

TBase 运维工程师手册



Tencent/大数据平台/TBase 研发组

版本号——V2

更新日期: 2019-07-08

1、TBase 数据库介绍及部署

1.1、使用 TBase 对运维工程师的基本要求

通常情况下，只要运维工程师有过 Postgresql, mysql, oracle, sql server, db2, sybase...等关系型数据库使用经验，使用起 TBase 来就没有什么困难，最佳的体验者为 Postgresql、pgxc、pgxl 运维人员。如果你没有这方面的经验，建议你先通读本文档。

1.2、什么是 TBase

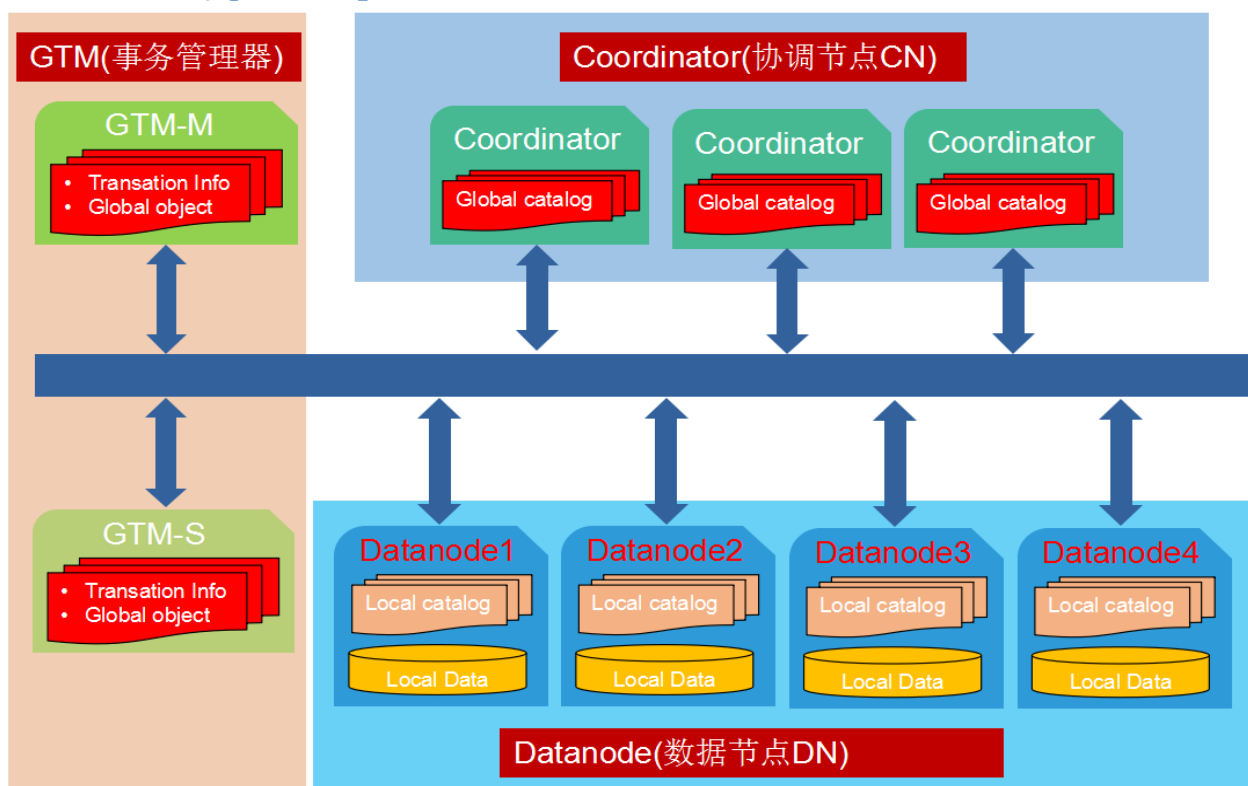
TBase 是一个提供写可靠性，多主节点数据同步的关系数据库集群平台。你可以将 TBase 配置一台或者多台主机上，TBase 数据存储在多台物理主机上面。数据表的存储有两种方式，分别是 distributed 或者 replicated，当向 TBase 发送查询 SQL 时，TBase 会自动向数据节点发出查询语句并获取数据。

1.3、TBase 设计目标

- 1、可以提供多个主节点同时处理来自应用端发出的 SQL 语句，这些节点称为 coordinator（协调节点）。
- 2、任何一个协调节点都有全局数据库视图，也就是说当任一协调节点接收更新数据并提交后，在另外的主节点可以迅速地看到变化的数据视图。
- 3、表以 distributed by shard 方式分布式存储，并且这对应用来说是透明的。
- 4、不需要架构师在应用层再构建 sharding 应用组件，大大的降低应用程序的复杂度，提高开发的效率。

1.4、TBase 集群构架

TBase集群架构：



■ Coordinator

Coordinator（简称 CN）是协调节点，是数据库服务的对外入口，负责数据的分发和查询规划，多个节点位置对等。业务请求发送给 CN 后，无需关心数据计算和存储的细节，由 CN 统一返回执行结果。CN 上只存储系统的元数据，并不存储实际的业务数据，可以配合支持业务接入增长动态增加。

■ Datanode

Datanode（简称 DN）是数据节点，执行协调节点分发的执行请求，实际存储业务数据。各个 DN 可以部署在不同的物理机上，也支持同物理机部署多个 DN 节点，注意互为主备 DN 不建议部署在同物理主机上。DN 节点存储空间彼此之间独立、隔离，是标准的 **share nothing** 存储拓扑结构。另外 TBase-V2 与 V1 最大的不同地方是 DN 与 DN 之间可以通信，互相交换数据。

■ GlobalTransactionManager

GlobalTransactionManager（简称 GTM），是全局事务管理器，负责全局事务管理。GTM 上不存储业务数据。

1.5、TBase 安装

1.5.1、运行环境硬件要求

用途（体验环境）	配置	数量(台)	部署组件
管理节点、协调节点、数据节点、集群管理平台、可视化开发工具	8 核 8G 200G	1	OssCenterMaster ConfdBMaster TStudio
管理高可用节点、协调节点、数据节点	8 核 8G 200G	1	OssCenterSlave ConfdBSlave Alarm

1.5.2、操作系统要求

支持系统版本	内核	glibc 版本
CentOS 7.3	Linux version 3.10.0	glibc-2.17

部署请使用最小化安装方式

Centos7.3minimal 发行版本，下载地址

http://mirror.neu.edu.cn/centos/7/isos/x86_64/CentOS-7-x86_64-Minimal-1611.iso

1.5.3、如何获取 TBase 安装包

请联系腾讯商务代表，或在 TBase 官网 <http://tbase.qq.com/download/> 获取 TBase 安装包和许可证文件。

1.5.4、安装过程

参考 TBase 部署文档

1.6 、配置 tbase 的客户端使用环境

Tbase-v2 是基于多租户设计，安装后所有机器都不会配置 tbase 的客户端默认使用环境，所以使用 psql 需要配置其环境变量，如下

```
export PGXZ_HOME=/usr/local/install/tbase_pgxz
export PATH=$PGXZ_HOME/bin:$PATH
export LD_LIBRARY_PATH=$PGXZ_HOME/lib:${LD_LIBRARY_PATH}
```

```
PGUSER=tbase
PGHOST=127.0.0.1
PGDATABASE=postgres
```

```
PGPORT=11000
```

```
export PGHOST PGUSER PGDATABASE PGPORT
```

这样就能使用 psql 客户端了，如果机器上面不准备安装任何节点，则你可以直接将上面的环境变量配置到

~/.bashrc 中

1.7 、TBase 每个 cn/dn 节点的系统限制

项目	值
单个数据库最大容量	无限制
单个表的最大容量	32TB
一行记录最大容量	1.6TB
一个字段的最大容量	1GB
一个表里最大记录行数	无限制
一个表里最大列数	250-1600，与列类型有关
一个表里最大索引个数	无限制
字段名长度	63

2、创建 TBase 默认的分布式使用环境

2.1、创建数据表默认的 default group

TBase 做为分布式数据库系统，使用前我们必需配置数据表默认分布的数据节点（DN），下面演示如何创建一个 default group

--切换为 tbase 用户

```
su tbase
```

--连接数据库--注意是连接到 cn 节点(后面没特别说明，所有数据库操作都是连接到 cn 节点)

```
psql -h 172.16.0.29 -p 15432 -U tbase -d postgres
```

```
psql (PostgreSQL 10 (TBase 2.01))
```

```
Type "help" for help.
```

--查询当前什么数据节点（DN）,这些 DN 节点就是上面初始化集群时建立的

```
postgres=# select * from pgxc_node where node_type='D';
```

node_name	node_type	node_port	node_host	nodeis_primary	nodeis_preferred	node_id	node_cluster_name
dn_0	D	23001	172.16.0.37	f	f	1485981022	tbase_cluster
dn_1	D	23002	172.16.0.42	f	f	-1300059100	tbase_cluster

(2 rows)

--建立数据表默认使用的 group

```
postgres=# create default node group default_group with(dn_0, dn_2);
```

2.2、为 default group 创建 shardmap

配置完成数据表默认使用的 DN 节点后，我们接下来需要配置记录的分区方案，shardmap 就是 TBase 各个哈希值与 DN 的对照表，下面演示如何创建一个 shardmap 给 default_group 使用

```
postgres=# create sharding group to group default_group;
postgres=# clean sharding;
```

至此我们就可以像单机一样使用 TBase 集群了。

2.3、更多 group 的使用方法

2.3.1、创建扩展 group

■ 建立 group

```
postgres=# create node group ext_group with(dn03,dn04);
CREATE NODE GROUP
```

■ 为 group 创建 shard

```
postgres=# create extension sharding group to group ext_group;
CREATE SHARDING GROUP
postgres=# clean sharding;
CLEAN SHARDING
postgres=#
```

2.3.2、删除 group

```
postgres=# drop sharding in group ext_group;
DROP SHARDING GROUP
postgres=# drop node group ext_group ;
DROP NODE GROUP
```

2.3.3、查询集群中 group 的数量

```
postgres=# select * from pgxc_group;
 group_name | default_group | group_members
-----+-----+-----
 default_group |          1 | 16386 16387
 ext_group    |          0 | 16388 16389
(2 rows)
```

3、管理及运维监控平台操作

3.1、系统管理平台

平台管理包含了资源管理（包括资源模板，软件包管理，服务器管理，服务器指标查看），租户管理，项目管理，系统信息查看，使用平台管理角色为平台管理员

3.1.1、登录方法

使用 TBase 一键部署好集群之后，监控平台会部署在你配置的那台机器上面,是一个运行着的 web 服务，默认的端口号为 8080，打开浏览器输入访问地址即可进入访问监控平台的登录页面

<http://xxx.xx.xx.xx:8080>

登录用户名：admin 登录密码：Admin@TbaseV2

3.1.2、资源管理-资源模板

■ 进入方法

平台管理-》资源管理-》资源模板

■ 相关功能

- 资源模板定义
- 资源模板备注信息修改
- 资源模板删除

3.1.2.1、添加资源模板

定义资源模板将来引添加服务器和申请集群时使用的，即使用资源模板把申请集群需要的资源与服务器关联起来，进入资源模板管理板块，点击“新建模板”按钮，弹出“新建模板”对话框，如下

TBase

资源管理

资源模板

软件包管理

机型管理

服务器管理

服务器指标

租户管理

实例管理

系统信息

证书信息

资源模板 / 新建模板

基本信息

名称只能包含大小写字母、数字和下划线（_），且只能以字母开头

* 模板名称: onecore_1ram_10ghdd

* 是否共享: ☒ 是 ☐ 否

模板描述: cpu: 1核心
内存: 1g
硬盘: 10g

资源信息

* CPU核数: 1 核

* 内存容量: 1 GB

* 磁盘容量: 10 GB

取消 新建

项目	说明
CPU 核数	为引用的项目节点分配 cpu 核心数
内存容量	为引用的项目节点分配内存数
磁盘容量	为引用的实例分配磁盘空间（需要服务器能识别出有这么多的 hdd 资源）

如果你机器数量够多，可以给 dn、cn、gtm 定义不同的模版。

3.1.2.2、修改资源模板备注

点击某个要修改备注信息的“资源模板”，录入修改信息后“编辑”完成保存。

3.1.2.3、删除资源模板

点击要删除“资源模板”右上角的“删除”按钮即可删除“资源模板”，已经被引用的“资源模板”无法删除

3.1.3、资源管理-软件包管理

■ 进入方法

平台管理-》资源管理-》软件包管理

■ 相关功能

- 上传软件包
- 删除软件包

3.1.3.1、添加软件包

添加的软件包将会在项目申请时使用，点击“添加软件包”在弹出的窗口输入软件包的 md5 值，选择要上传的软件包文件即可完成上传工作。一键安装后这里自动添加一键安装包里面的 tbase 软件包

3.1.3.2、删除软件包

点击要删除软件包后面的“删除”链接即可删除软件包，被项目引用的软件包无法删除

3.1.4、资源管理-机型管理

机型管理主要是为相同的机器定义好模板，这样可以集中的添加相同类型的机器，方便管理和提高添加机器的效率。

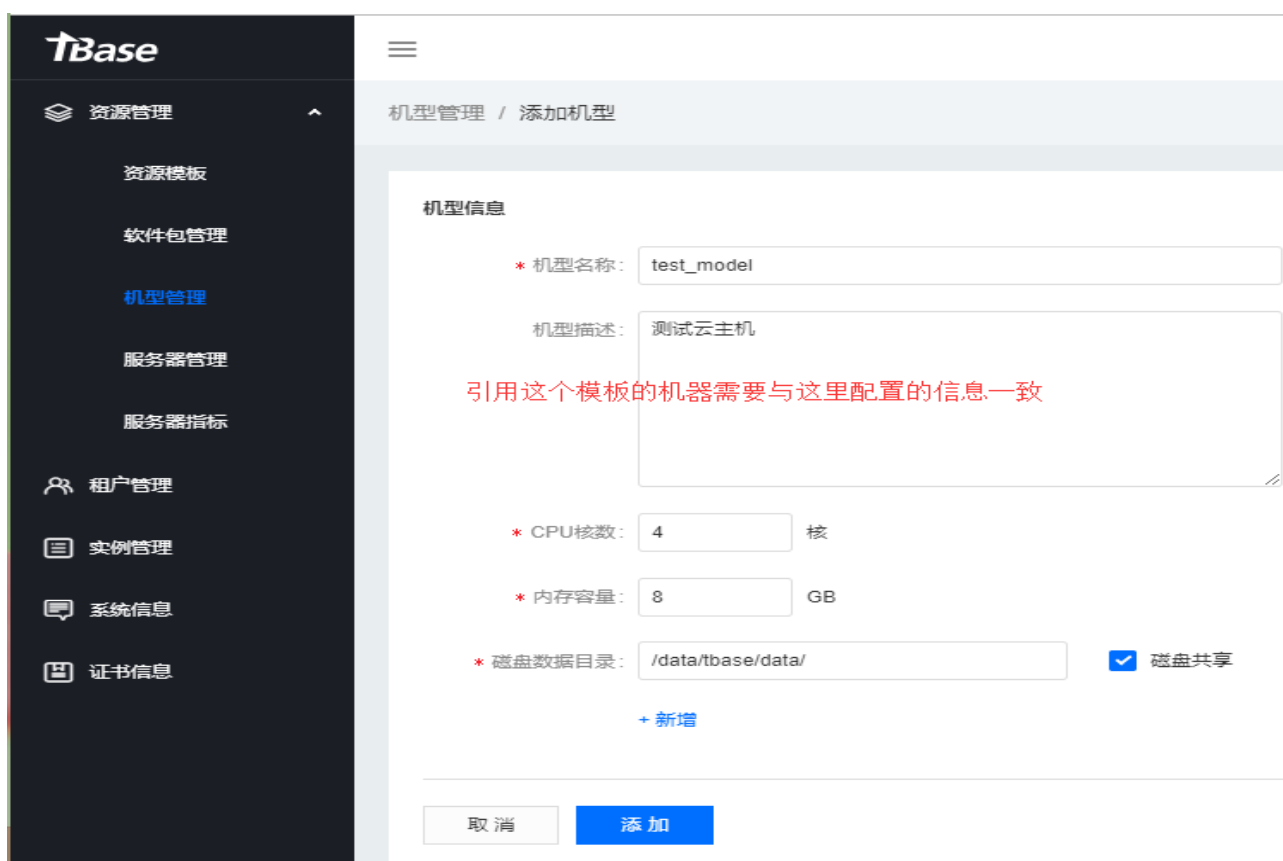
■ 进入方法

平台管理-》资源管理-》机型管理

■ 相关功能

- 添加机型
- 删除机型
- 查看机型详情

3.1.4.1、添加机型



编辑完成按“添加”按钮即可，后来只要机器与这样机型一样就可以直接这个机型模板

提示

- 1、填写完成后按“添加”保存。
- 2、“磁盘共享”表示准许在该目录下创建多个节点。
- 3、引用这个机型的机器需要与这里配置的信息一致。

3.1.4.2、查看机型信息

进入机型列表，点击要查询机型“详情”链接，即可查看该机型的详细信息，包括引用的机器



机型名称	CPU核数	内存容量	机型描述	操作
test_model	1核	1G	测试云主机	详情 删除

机型信息

机型名称: test_model

机型描述: 测试云主机

CPU核数: 1 核

内存容量: 1 G

磁盘数据目录: /data/tbase/data/

此机型的服务器

服务器IP	IDC名称	地域
暂无此机型的服务器		

3.1.4.3、删除机型信息

进入机型列表，点击“删除”按钮即可删除该机型，已经使用了的机型无法删除。

3.1.5、资源管理-服务器管理

■ 进入方法

平台管理-》资源管理-》服务器管理

■ 相关功能

- 添加服务器
- 删除服务器
- 查看服务器详情
- 添加新的磁盘数据目录

3.1.5.1、添加服务器前的准备工作

- 参照 5.1 把机器安装好 os 并且配置好
- 新服务器与 center 主，center 备做好互信（包括 tbase/root 用户）
- 配置 tbase 用户的 sudo 权限

3.1.5.2、添加服务器

添加的服务器将会在项目分配机器时使用，点击“添加服务器”

服务器信息

idc_1 local 172.16.0.3

* IDC 名称: idc_1 IDC名称在前面安装时配置的名称

地区: local

* 服务器IP地址: 172.16.0.3

* 服务器管理进程port: 8118

* 资源模板: onecore_1ram_10ghdd (1核CPU /... 资源模板从前面的定义中选择

* 机型: 无 这里不选择机型，如果选择机型则需要资源完全匹配

* 磁盘数据目录: /data/tbase/data/ 磁盘共享

+ 新增

+ 添加一个服务器

取消 完成

编辑完成按“完成”按钮即可

3.1.5.3、查看服务器信息

点击要查询服务器“详情”链接，即可查看服务器的所有详细信息

3.1.5.4、给已存在的服务器增加新的磁盘

在服务器“详情”页面点击添加新的磁盘图标，注意，一个磁盘对应一个目录

服务器管理 / 服务器详情

基本信息

IDC名称	idc_1
地域	local
服务器IP地址	172.16.0.13
服务器管理进程port	8118

硬件信息

CPU核数 (可用 / 总共)	1 / 5 核
内存总容量 (可用 / 总共)	3 / 7 GB
磁盘总容量 (可用 / 总共)	145 / 200 GB

磁盘信息

磁盘 1 x

点击，弹出添加新盘的对话框  +

磁盘名称	/dev/vdb1
磁盘类型	HDD
磁盘容量 (可用 / 总共)	146 / 200 GB
数据目录	/data/tbase/data/
是否为虚拟盘	否
是否共享	是

3.1.5.5、删除服务器

在服务器“详情”页，点击“删除”按钮即可删除服务器，已经使用了的服务器无法删除

3.1.6、资源管理-服务器指标

■ 进入方法

平台管理-》资源管理-》服务器指标

■ 相关功能

- 查看各服务器指标
- 服务器指标对比

3.1.6.1、查看服务器指标



3.1.7、租户管理

■ 进入方法

平台管理-》租户管理

■ 相关功能

- 添加租户
- 修改租户描述
- 删除租户

3.1.7.1、添加租户

租户即能使用这个平台的用户，一个租户可以申请多个集群，点击“添加租户”，如下



添加租户

* 租户账号: tbase_01

租户描述: xxx银行总账户

取消 确定

输入租户账号（账号需要全局唯一），和租户描述，按“确定”即可添加一个新的租户

3.1.7.2、修改租户描述

单击租户描述修改图标，录入修改内容，按“确定”即可完成修改

3.1.7.3、删除租户

单击要删除的租户右上角“删除”图标，在弹出的确认窗口点“确认”即可删除租户，已经使用的租户无法删除。

3.1.8、实例管理

■ 进入方法

平台管理-》实例管理

■ 相关功能

- 通过和确认实例申请
- 驳回实例申请
- 查看实例详情

3.1.8.1、通过和确认实例申请

租户申请实例提交后将会流转到这里进行审批，如下所示

实例管理

全部 待审批 已审批

请填写要搜索的内容

租户名称	实例名称	创建时间	节点个数	管理员	安全员	审计员	状态	操作
tencent	tbase01	09-13 14:44	全局事务管理器：2个 协调节点：2个 数据节点：1个	tbase01_admin	tt_mls	tt_audit	待审批	通过 驳回 详情
tencent	tencent_test	09-13 10:40	全局事务管理器：2个 协调节点：2个 数据节点：2个	tbase01_admin	tbase01_mls	tbase01_audit	分配成功	详情

共 2 条 < 1 > 10 条/页 跳至 页

3.1.8.2、驳回实例申请

点击上图的“驳回”链接即可驳回实例申请

3.1.8.3、查看实例详情

点击上图的“实例信息”即可查看实例的所有信息，包含选择软件包名称，各节点数量，租户名称，实例名称，实例管理员账号等；点击“节点详情”即可查看各节点的 ip 地址，port 和数据目录、bin 目录位置等信息。

实例管理 / 实例详情

实例信息 节点详情

基本信息

租户名称： tbase

实例名称： tbase01

实例描述：

实例类型： 分布式

软件包名称： tbase_pgxz-2.0.9-i.x86_64.rpm

初始化配置

节点类型	数量	模版资源（CPU / 内存 / 磁盘）
全局事务管理器	2个	onecore_1ram_10ghdd（1核 / 1G / 10G）
协调节点	2个	onecore_1ram_10ghdd（1核 / 1G / 10G）
数据节点	4个（2组，每组2个）	onecore_1ram_10ghdd（1核 / 1G / 10G）

角色配置

实例管理员： tbase_01_admin

实例审计员： tbase_01_audit

实例安全员： tbase_01_mls

普通用户： 无

实例信息 节点详情

节点类型 全部

节点名称	角色	节点IP	端口	资源模板	数据目录	bin目录
dn001	主	172.16.0.3	11002	oncore_1ram_10ghdd	/data/tbase/data/data_00/tbase01_1/data/dn001	/data/tbase/data/data_00/tbase01_1/2.0.9/install/tbase_pgiz/
dn001	备	172.16.0.13	11002	oncore_1ram_10ghdd	/data/tbase/data/data_00/tbase01_1/data/dn001	/data/tbase/data/data_00/tbase01_1/2.0.9/install/tbase_pgiz/
dn002	主	172.16.0.13	11004	oncore_1ram_10ghdd	/data/tbase/data/data_01/tbase01_1/data/dn002	/data/tbase/data/data_01/tbase01_1/2.0.9/install/tbase_pgiz/
dn002	备	172.16.0.3	11004	oncore_1ram_10ghdd	/data/tbase/data/data_01/tbase01_1/data/dn002	/data/tbase/data/data_01/tbase01_1/2.0.9/install/tbase_pgiz/
cn001	主	172.16.0.13	11000	oncore_1ram_10ghdd	/data/tbase/data/data_02/tbase01_1/data/cn001	/data/tbase/data/data_02/tbase01_1/2.0.9/install/tbase_pgiz/
cn002	主	172.16.0.3	11000	oncore_1ram_10ghdd	/data/tbase/data/data_02/tbase01_1/data/cn002	/data/tbase/data/data_02/tbase01_1/2.0.9/install/tbase_pgiz/
gim001	主	172.16.0.13	11006	oncore_1ram_10ghdd	/data/tbase/data/data_03/tbase01_1/data/gim001	/data/tbase/data/data_03/tbase01_1/2.0.9/install/tbase_pgiz/
gim002	备	172.16.0.3	11006	oncore_1ram_10ghdd	/data/tbase/data/data_03/tbase01_1/data/gim002	/data/tbase/data/data_03/tbase01_1/2.0.9/install/tbase_pgiz/

共 5 条 < 1 > 10 条/页 跳至 页

返回

3.1.9、系统信息

■ 进入方法

平台管理-》系统信息

■ 相关功能

➤ 查看平台的注册信息

系统信息

系统名称	TBase OSS
系统版本	2.0.9
系统管理员	Admin
系统创建日期	2018-11-14

3.1.10、证书信息

■ 进入方法

平台管理-》证书信息

■ 相关功能

- 查看证书信息
- 更新证书信息

3.1.10.1、查看当前证书信息

证书上显示的资源为用户可使用的最大资源，项目有：

- 所有集群 cpu 核心总数
- 每个 cn 最大连接数
- 单一集群 dn 最多个数
- 单一集群 cn 最多个数
- 证书的有效日期

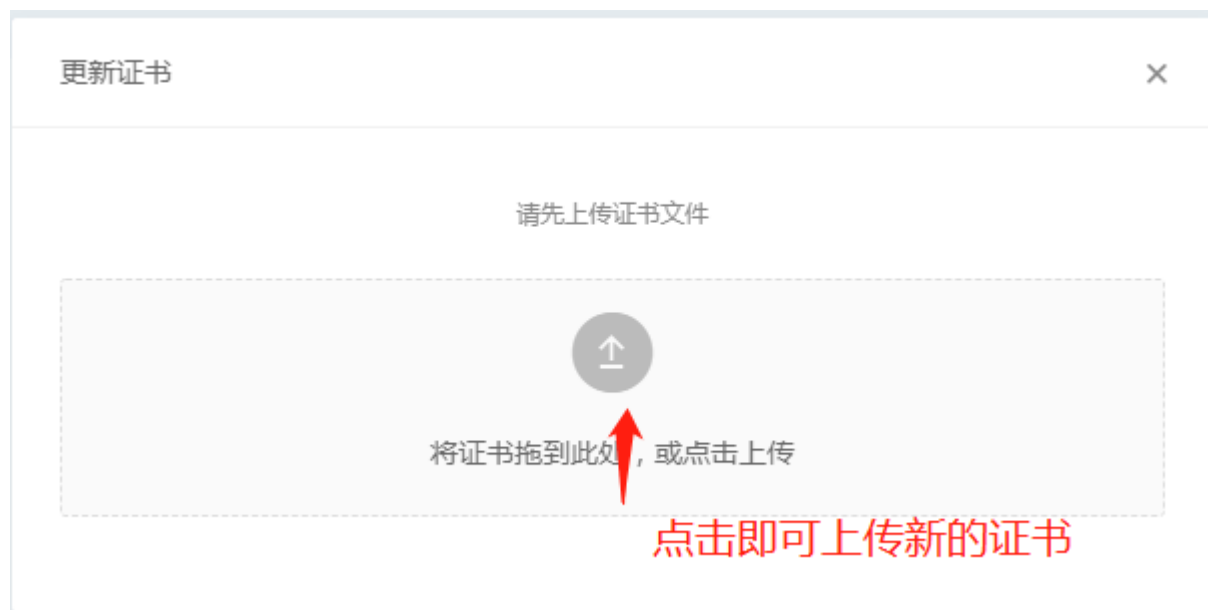
证书信息

证书ID	2000031
证书类型	Standard
创建日期	2018-10-22
有效期至	2099-12-31
最大CPU核数	65535
最大协调节点数	128
最大数据节点数	256
每个CN最大连接数	8000
用户ID	
申请人	
被授权集团名称	
被授权公司名称	
被授权项目名称	

更新证书

3.1.10.2、更新证书

在证书查看页面，点击“更新证书”，系统弹出证书上传窗口，选择新的证书上传即可。



3.2、租户管理平台

3.2.1、登录方法

打开浏览器输入访问地址即可进入访问监控平台的登录页面

http://xxx.xx.xx.xx:8080

登录用户名：租户名 登录密码：Tx2018@qq.com

3.2.2、实例管理

■ 进入方法

租户管理平台-》项目管理

■ 相关功能

- 添加实例
- 实例删除
- 查看实例详情

3.2.2.1、实例申请

点击“添加实例”

1 基本信息

2 节点配置

3 角色配置

基本信息

* 项目名称:

tbase_03

项目名称, 英文名/下划线/数字组合

项目描述:

xx银行总账户

* 项目类型:

TBase

项目类型选择tbase

* 软件包:

tbase_pgxz-2.0.6-i.x86_64.rpm

这里选择tbase软件包

取消

下一步

TBase

实例管理

用户管理

基本信息

2 节点配置

角色配置

节点配置

全局事务管理器

* 节点模板:

acore_1GRAM_10GHDD (1核 CPU / 1G ...

选择使用哪个资源模版

* 主节点:

gtm001

* 备节点:

gtm002

删除

gtm最少要一主一备, 也可以配置多个gtm备

+ 添加备节点

协调节点

* 节点模板:

acore_1GRAM_10GHDD (1核 CPU / 1G ...

* 节点名:

cn001

删除

cn002

删除

示例配置两个cn节点, cn节点不能少于2个, 不然实例无法创建成功

+ 添加节点

数据节点

* 节点模板:

acore_1GRAM_10GHDD (1核 CPU / 1G ...

* 备节点数:

1主1从

选择备节点数量, 1主1从是1个备节点, 1主2从则是两个备节点, 目前只提供这两种选择

* 节点名:

dn001

删除

dn002

删除

这里是配置两组dn, 根据1主1从, 总共是4个dn节点, dn节点名称可以自己手动修改

+ 添加节点

上一步

下一步

20

基本信息 节点配置 3 角色配置

角色配置

* 项目管理员:

* 项目审计员: 选择管理员，如果管理员不存在，则点击“添加用户”即可添加管理员

* 项目安全员:

+ 添加用户

上一步

提交审批

项目管理员以 admin，审计员以 audit，安全员以 mls 结尾，输入完成，点击“提交审批”，你的申请将会传给系统管理员进行审批。**注意：实例名称不要录入中文名或者特殊字符**

3.2.2.2、删除实例

点击实例的“删除”链接，在弹出的确认框输入你的密码，点击“确定”即可删除实例。

实例管理							
+ 添加实例							
请输入要搜索的内容							
实例名称	创建时间	节点个数	管理员	安全员	审计员	状态	操作
tbase01	09-13 14:44	全局事务管理器：2个 协调节点：2个 数据节点：1个	tbase01_admin	tl_mls	tl_audit	待审批	详情 删除
tencent_test	09-13 10:40	全局事务管理器：2个 协调节点：2个 数据节点：2个	tbase01_admin	tbase01_mls	tbase01_audit	分配成功	详情 删除
共 2 条 < 1 > 10 条/页 跳至 页							

3.2.2.3、实例详情查看

点击下图的“实例信息”即可查看实例的所有信息，包含选择软件包名称，各节点数量，租户名称，实例名称，实例管理员账号等；点击“节点详情”即可查看各节点的 ip 地址，port 和数据目录、bin 目录位置等信息。

实例信息

节点详情

基本信息

实例名称： tbase01

实例描述： 请填写实例描述信息

保存

实例类型： 分布式

软件包名称： tbase_pgxz-2.0.9-i.x86_64.rpm

初始化配置

节点类型	数量	模版资源 (CPU / 内存 / 磁盘)
全局事务管理器	2个	onecore_1ram_10ghdd (1核 / 1G / 10G)
协调节点	2个	onecore_1ram_10ghdd (1核 / 1G / 10G)
数据节点	4个 (2组, 每组2个)	onecore_1ram_10ghdd (1核 / 1G / 10G)

角色配置

实例管理员： tbase_01_admin

实例审计员： tbase_01_audit

实例安全员： tbase_01_mls

普通用户： 无

节点类型 全部

节点名称	角色	节点IP	端口	资源模版	数据目录	bin目录
dn001	主	172.16.0.3	11002	onecore_1ram_10ghdd	/data/tbase/data/data_00/tbase01_1/data/dn001	/data/tbase/data/data_00/tbase01_1/2.0.9/install/tbase_pgxz/
dn001	备	172.16.0.13	11002	onecore_1ram_10ghdd	/data/tbase/data/data_00/tbase01_1/data/dn001	/data/tbase/data/data_00/tbase01_1/2.0.9/install/tbase_pgxz/
dn002	主	172.16.0.13	11004	onecore_1ram_10ghdd	/data/tbase/data/data_01/tbase01_1/data/dn002	/data/tbase/data/data_01/tbase01_1/2.0.9/install/tbase_pgxz/
dn002	备	172.16.0.3	11004	onecore_1ram_10ghdd	/data/tbase/data/data_01/tbase01_1/data/dn002	/data/tbase/data/data_01/tbase01_1/2.0.9/install/tbase_pgxz/
cn001	主	172.16.0.13	11000	onecore_1ram_10ghdd	/data/tbase/data/data_02/tbase01_1/data/cn001	/data/tbase/data/data_02/tbase01_1/2.0.9/install/tbase_pgxz/
cn002	主	172.16.0.3	11000	onecore_1ram_10ghdd	/data/tbase/data/data_02/tbase01_1/data/cn002	/data/tbase/data/data_02/tbase01_1/2.0.9/install/tbase_pgxz/
gtn001	主	172.16.0.13	11006	onecore_1ram_10ghdd	/data/tbase/data/data_03/tbase01_1/data/gtn001	/data/tbase/data/data_03/tbase01_1/2.0.9/install/tbase_pgxz/

3.2.2.4、重新填写申请实例资料

点击上图的“重新申请”即可进入实例资料变更填写，流程跟实例申请一样，已经通过的实例无法再重新申请，除非管理员驳回。

3.2.3、用户管理

■ 进入方法

租户管理平台-》用户管理

■ 相关功能

- 添加用户
- 删除用户
- 修改用户备注

3.2.3.1、添加用户

点击“添加用户”



The image shows a 'Add User' dialog box with the following fields and options:

- * 用户名:** tbase_01_user1
- * 指派角色:** ☐ 不指派 ☐ 项目管理员 ☒ 普通用户
- 选择为项目管理员或者普通用户时需要选择项目**
- * 指派项目:** tbase_01 x
- 用户描述:** 集群管理员
- Buttons:** 取消 (Cancel), 确定 (Confirm)

录入信息后点击“确定”完成用户添加

3.2.3.2、删除用户

用户名	描述	所属项目	角色	操作
tbase_01_admin		tbase_01	管理员	编辑 删除
tbase_01_audit		tbase_01	审计员	编辑 删除
tbase_01_mls		tbase_01	安全员	编辑 删除

点击上图“删除”链接，在弹出的确认框点“确定”即可删除用户，如用户已经被项目引用，则无法删除。

3.2.3.3、修改用户备注

点击上图“编辑”链接，在弹出的编辑窗口录入备注信息后点击“确定”即可。

3.3、集群运维平台

3.3.1、登录方法

打开浏览器输入访问地址即可进入访问监控平台的登录页面

http://xxx.xx.xx.xx:8080

登录用户名：集群管理员账号 登录密码：Tx2018@qq.com

3.3.2、项目监控-集群监控

■ 进入方法

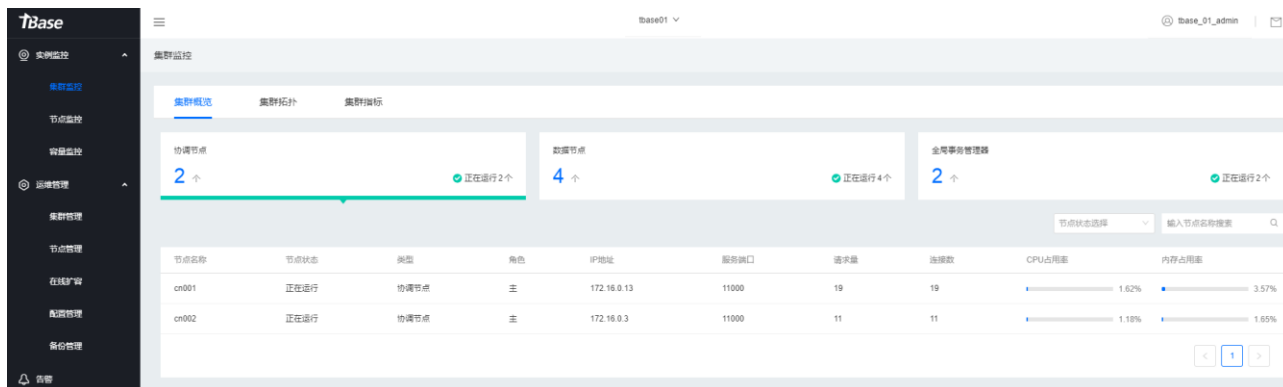
集群运维平台-》项目监控-》集群监控

■ 相关功能

- 集群概览查看
- 集群拓补查看
- 集群运行指标查看

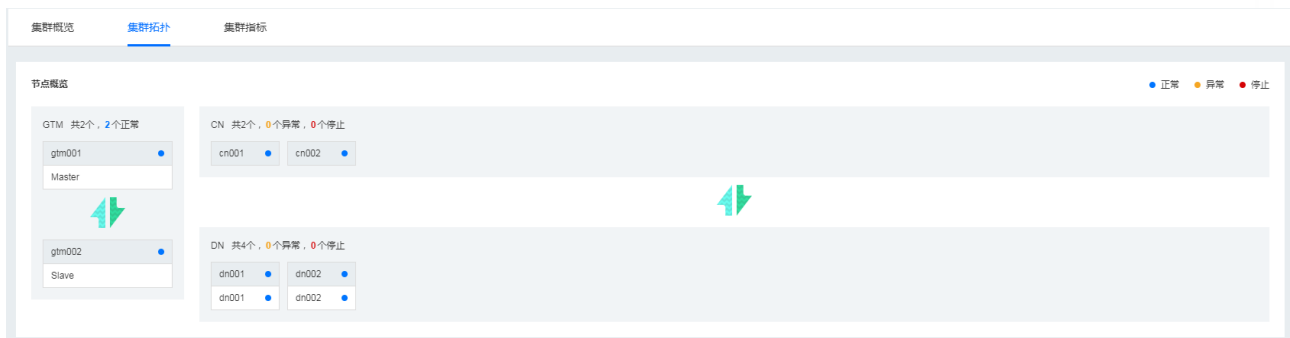
3.3.2.1、集群概览查看

点击集群概览页框，即可查看集群概览



3.3.2.2、集群拓补查看

点击集群拓补页框即可查看



3.3.2.3、集群运行指标查看

点击集群指标页框即可查看



3.3.3、项目监控-节点监控

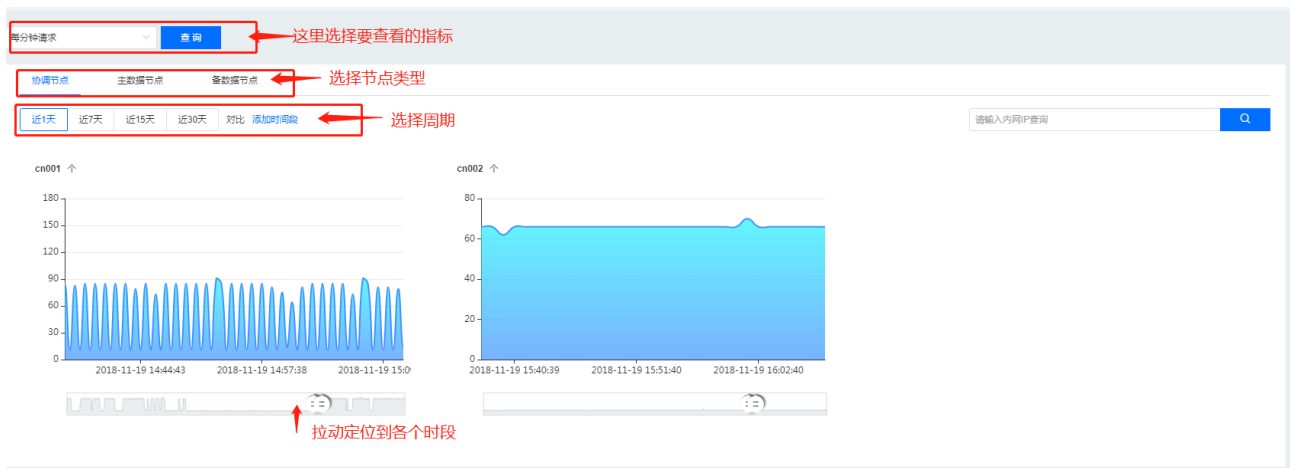
■ 进入方法

集群运维平台-》项目监控-》节点监控

■ 相关功能

➤ 查看节点运行指标

3.3.3.1、查看节点运行指标



3.3.4、项目监控-容量监控

■ 进入方法

集群运维平台-》项目监控-》容量监控

■ 相关功能

➤ 查看节点的空间点用情况

3.3.4.1、查看节点空间点用情况

节点名称	角色	节点IP	端口	类型	数据地址	挂数据盘	数据库目录容量	数据库容量	压缩率	wal日志容量
dn001	主	172.16.0.3	11002	数据节点	/data/tbase/data/tdata_00/tbase01_1/data/dn001	/dev/vdb1	2671.93MB	27.17MB	1	2448.63MB
dn001	备	172.16.0.13	11002	数据节点	/data/tbase/data/tdata_00/tbase01_1/data/dn001	/dev/vdb1	2703.63MB	27.32MB	1	2464.64MB
dn002	主	172.16.0.13	11004	数据节点	/data/tbase/data/tdata_01/tbase01_1/data/dn002	/dev/vdb1	2749.09MB	27.17MB	1	2448.63MB
dn002	备	172.16.0.3	11004	数据节点	/data/tbase/data/tdata_01/tbase01_1/data/dn002	/dev/vdb1	2638.20MB	27.32MB	1	2464.64MB

项目	说明
数据库目录容量	指 data 目录占用空间
数据库容量	指 pg_database_size()大小
Wal 日志容量	指/data/pg_wal 目录大小

3.3.5、运维管理-集群管理

■ 进入方法

集群运维平台-》运维管理-》集群管理

■ 相关功能

- 集群健康检查
- 集群启动/停止

3.3.5.1、集群健康检查

集群名: tbase_01_1 • 100分 健康检查 点击即可进行健康检查，第一次比较慢 停止集群

详细监控检查日志存储于“监控系统报目录/pgxz/healthchecklog”中

检查项	上次检查结果	检查结果
✓ 磁盘空间不足预警 数据库占用磁盘空间是否大于可用空间的80%	成功	成功
✓ 监听端口一致性 数据库分配端口与实际运行端口是否完全一致	成功	成功
✓ 操作系统参数一致性 设备操作系统运行参数是否与模板一致	成功	成功
✓ 集群拓扑元数据 DN和CN节点记录的集群拓扑架构是否完全一致	成功	成功
✓ 监控互信 监控系统是否能正常连通集群各节点	成功	成功
✓ CN与DN的连通性 CN是否能访问所有DN，并且正常执行检查命令	成功	成功

这是检查的结果

项目包括"磁盘空间不足预警","监听端口一致性","操作系统参数一致性","集群拓扑元数据","监控互信","CN 与 DN 的连通性"

3.3.5.2、集群停止和启动

集群名: tbase_01_1 • 100分 健康检查 停止集群

详细监控检查日志存储于“监控系统报目录/pgxz/healthchecklog”中

检查项	上次检查结果	检查结果
✓ 磁盘空间不足预警 数据库占用磁盘空间是否大于可用空间的80%	成功	成功
✓ 监听端口一致性 数据库分配端口与实际运行端口是否完全一致	成功	成功
✓ 操作系统参数一致性 设备操作系统运行参数是否与模板一致	成功	成功
✓ 集群拓扑元数据 DN和CN节点记录的集群拓扑架构是否完全一致	成功	成功
✓ 监控互信 监控系统是否能正常连通集群各节点	成功	成功
✓ CN与DN的连通性 CN是否能访问所有DN，并且正常执行检查命令	成功	成功

当集群在运行时，这里显示为“停止集群”，反之显示为“启动集群”

停止集群时，系统弹出密码输入框，输入你登录的密码

停止集群

1

是否确认停止该集群，确认停止请输入登录密码

* 登录密码：

取消

确定

按确认即可停止集群服务

3.3.5.2、多平面管理

多平面管理用于配置管理只读数据源，要配置只读数据源，需要创建备 cn,然后配置这个备 cn 路由指向备 dn,从而实现备机只读功能。

■ 创建只读数据源

实例监控

集群管理

节点监控

容量监控

运维管理

集群管理

节点管理

在线扩容

冷热迁移

健康检查

多平面管理

+ 创建只读平面

平面名	别名	创建时间	节点个数	状态	操作
只读平面1	只读平面1	2019-03-22 23:15:18	全局事务管理器: 2个 协调节点: 3个 数据节点: 4个	正常	详情

共 1 条 < 1 > 10 条/页 跳至 页

2、点击多平面管理

3、点击创建只读平面

1 点击集群管理

创建只读平面

备平面名：只读平面1

全局事务管理器：gtm001; gtm002

* dn001备节点：

dn001 (10.0.3.39:11002) ×

* dn002备节点：

dn002 (10.0.3.42:11004) ×

* 协调备节点：

cn001 (10.0.3.39:11008) ×

取消

确定

选择对应 dn 备节点和协调 (cn) 备节点，然后按确定即可，这样 10.0.3.39 就能提供只读能力
在 pgxc_node 可以看到的其拓补结构为双平面

```

[base@VM_3_39-centos ~]$ psql
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.

postgres=# select * from pgxc_node;
 node_name | node_type | node_port | node_host | nodeis_primary | nodeis_preferred | node_id | node_cluster_name
-----
 agtm_0    | G         | 11006     | 10.0.3.42 | t               | f                 | 475343005 | tbase_cluster
 cn001     | C         | 11000     | 10.0.3.42 | f               | f                 | -264077367 | tbase_cluster
 cn002     | C         | 11000     | 10.0.3.39 | f               | f                 | -674870440 | tbase_cluster
 dn001     | D         | 11002     | 10.0.3.42 | f               | f                 | 2142761564 | tbase_cluster
 dn002     | D         | 11004     | 10.0.3.39 | f               | f                 | -17499968  | tbase_cluster
-----
 cn001     | C         | 11008     | 10.0.3.39 | t               | t                 | -264077366 | tbase_cluster_01
 dn001     | D         | 11002     | 10.0.3.39 | f               | f                 | 2142761565 | tbase_cluster_01
 dn002     | D         | 11004     | 10.0.3.42 | f               | f                 | -17499967  | tbase_cluster_01
(8 rows)
  
```

读写

只读

■ 删除只读数据源

健康检查

多平面管理

+ 创建只读平面

平面名	别名	创建时间	节点个数	状态	操作
读/写平面	tbase_cluster	2019-03-22 23:15:18	全局事务管理器: 2个 协调节点: 2个 数据节点: 2个	正常	详情
只读平面1	tbase_cluster_01	2019-03-23 10:46:58	全局事务管理器: 2个 (与主平面共用) 协调节点: 1个 数据节点: 2个	正常	详情 删除

共 2 条

<1>

10 条/页 跳至

页

弹出删除确认窗口，输入当前操作用户密码，按“确定”按钮

删除只读平面

!

删除平面将导致无法通过这个平面访问集群数据，是否确认删除？确认请输入登录密码

* 登录密码：

.....

取消

确定

注意：DN 主节点扩容后，需要增加 DN 主对应备节点，然后重建只读数据源。

3.3.6、运维管理-节点管理

■ 进入方法

集群运维平台-》运维管理-》节点管理

■ 相关功能

- 节点服务启动/停止
- dn 节点主从切换
- 重做备机
- 添加节点（包括 GTM,CN,DN 主，DN 备）

3.3.6.1、节点服务启动/停止

节点列表		节点组							
+ 添加节点		请输入节点名称/内网ip搜索							
节点名称	主从类型	同步类型	节点类型	日志同步状态	状态	操作状态	更新时间	操作	
+ cn001 172.16.0.13:11000	主	异步复制	协调节点	不需要同步	运行	未操作	2018-11-19 21:19:23	点击“停止”即可停止节点服务	
+ cn002 172.16.0.3:11000	主	异步复制	协调节点	不需要同步	停止	停止成功	2018-11-19 21:19:24	启动 停止	
+ dn001 172.16.0.3:11002	主	异步复制	数据节点	日志同步状态正常	运行	切换成功	2018-11-19 21:19:14	点击“启动”即可启动节点服务	
+ dn001 172.16.0.13:11002	从	异步复制	数据节点	日志同步状态正常	运行	切换成功	2018-11-19 21:19:08	启动 停止 主从切换 重做备机	
+ dn002 172.16.0.3:11004	主	异步复制	数据节点	日志同步状态正常	运行	切换成功	2018-11-19 21:19:03	启动 停止	
+ dn002 172.16.0.3:11004	从	异步复制	数据节点	日志同步状态正常	运行	切换成功	2018-11-19 21:18:59	启动 停止 主从切换 重做备机	
+ gtm001 172.16.0.13:11006	主	异步复制	全局事务管理器	日志同步状态正常	运行	未操作	2018-11-19 21:19:03	启动 停止	
+ gtm002 172.16.0.3:11006	从	异步复制	全局事务管理器	日志同步状态正常	运行	未操作	2018-11-19 21:19:00	启动 停止 主从切换	

停止节点

您将停止【Coordinator 172.16.0.13:11000】，停止过程中，该节点不再接受新增请求，余下事务15秒未执行完将强制终止，停止后，节点数据不删除。

点击“停止”按钮即可停止服务

取消 停止



3.3.6.2、dn 节点主从切换



3.3.6.3、dn 备机重做





3.3.6.4、添加节点

节点列表		节点组							
+ 添加节点		请输入节点名称/内网ip搜索							
节点名称	主从类型	同步类型	节点类型	日志同步状态	状态	操作状态	更新时间	操作	
cn001 172.16.0.13:11000	主	异步复制	协调节点	不需要同步	运行	未操作	2018-11-19 21:21:53	启动 停止	
cn002 172.16.0.13:11000	主	异步复制	协调节点	不需要同步	运行	启动成功	2018-11-19 21:22:15	启动 停止	
dn001 172.16.0.13:11002	主	异步复制	数据节点	日志同步状态正常	运行	切换成功	2018-11-19 21:22:14	启动 停止	
dn001 172.16.0.13:11002	从	异步复制	数据节点	日志同步状态正常	运行	切换成功	2018-11-19 21:22:08	启动 停止 主从切换 重做备机	
dn002 172.16.0.13:11004	主	异步复制	数据节点	日志同步状态正常	运行	切换成功	2018-11-19 21:22:13	启动 停止	
dn002 172.16.0.13:11004	从	异步复制	数据节点	日志同步状态正常	运行	切换成功	2018-11-19 21:21:59	启动 停止 主从切换 重做备机	
gtm001 172.16.0.13:11006	主	异步复制	全局事务管理器	日志同步状态正常	运行	未操作	2018-11-19 21:22:03	启动 停止	

● 添加 CN 节点

节点信息

CN003

* 节点类型:

CN

* 节点名称:

CN003

* 资源模板:

e,1GRAM,1G DISK S...

* 节点IP:

172.16.0.3

项目	说明
节点名称	名称规则按 cn001,cn002,cn003 至 cn999
资源模板	需要选择正确的模板

节点 IP	这里的列表是上面选择模板后自动生成
-------	-------------------

- 添加备 CN 节点

节点信息 ⓘ

cn001

* 节点类型:

备CN

* 所属平面:

读/写平面

* 主节点:

cn001

* 资源模板:

gtm_cn_dn (8G / 0G / 50G)

* 节点IP:

172.16.0.10

+ 添加节点

取消

完成

项目	说明
所属平面	选择读/写平面
主节点	添加备 cn 需选择 cn 主节点
资源模板	需要选择正确的模板
节点 IP	这里的列表是上面选择模板后自动生成

- 添加备 GTM

GTM003

* 节点类型:

备GTM

* 节点名称:

GTM003

* 资源模板:

re,1GRAM,1G DISK S...

* 节点IP:

172.16.0.3

项目	说明
节点名称	名称规则按 gtm001,gtm002,gtm003 至 gtm999
资源模板	需要选择正确的模板
节点 IP	这里的列表是上面选择模板后自动生成

- 添加备 DN

dn003

* 节点类型:

备DN

* 主节点:

dn003

* 资源模板:

ore,1GRAM,1G DISK S...

* 节点IP:

172.16.0.13

项目	说明
主节点	添加 dn 备需选择 dn 主节点
资源模板	需要选择正确的模板
节点 IP	这里的列表是上面选择模板后自动生成

- 添加主 DN

节点信息

DN005

* 节点类型:

DN

* 节点名称:

DN005

* 资源模板:

通用模板1core,1GRAM,1G DISK S...

* 节点IP:

172.16.0.13

所属节点组:

default_group

这里一定要选择，除非你添加的节点用于新的GROUP

+ 添加节点

取消 完成

项目	说明
节点名称	名称规则按 dn001,dn002,dn003 至 dn999
资源模板	需要选择正确的模板
节点 IP	这里的列表是上面选择模板后自动生成
所属节点组	这里一定要选择正确，除非你添加的节点用于新的 Group

3.3.7、运维管理-在线扩容

■ 进入方法

集群运维平台-》运维管理-》在线扩容

■ 相关功能

- 数据迁移
- 空间清理

3.3.7.1、数据迁移

迁移任务

空间清理

+ 创建任务

点击创建任务，弹出数据迁移对话框

请编写要搜索的内容

任务名	创建时间	源节点	目标节点	状态	操作
dn003_2_dn004	2018-08-07 10:37:07	dn003	dn004	迁移完成	详情
dn001_2_dn003	2018-08-06 22:08:07	dn001	dn003	迁移完成	详情

共 2 条

< 1 >

10 条页

跳至

页

创建任务

* 任务名称: dn001_2_dn003

* 源节点: dn001
选择要搬迁数据的节点后系统自动在下面显示可以迁移的shard列表

* 目标节点: dn003

1 项 选择迁移的shard

请输入搜索内容

2234 (1000001条数据, 占用81.0...)

1 项 已选的shard

请输入搜索内容

3318 (1000000条数据, 占用81.0...)

这里是选择那些shard要搬迁

这里显示可以迁移的shardid

取消 确定

选择完成后点击“确定”，剩下的工作就是系统自己搬迁数据了。执行数据搬迁要求所有表都要有对应的逻辑复制标识，规则如下

- 有主键的表

不需要建立

- 有唯一索引的表

```
postgres=# alter table t1 replica identity using index t1_id_uidx;
```

```
ALTER TABLE
```

```
postgres=# \d+ t1
```

Table "public.t1"

Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
id	integer		not null		plain		
mc	text				extended		

Indexes:

"t1_id_uidx" UNIQUE, btree (id) **REPLICA IDENTITY**

Distribute By: SHARD(id)

Location Nodes: ALL DATANODES

```
postgres=#
```

注意：做为 **replaice identity** 字段，必需为 **not null** 约束

- 无主键或无唯一索引的表

```
postgres=# alter table t replica identity full;
```

```
ALTER TABLE
```

```
postgres=# \d+ t
```

Table "public.t"

Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
id	integer				plain		
mc	text				extended		

Replica Identity: FULL

Distribute By: SHARD(id)

Location Nodes: ALL DATANODES

```
postgres=#
```

3.3.7.2、清理数据

迁移任务 **空间清理**

[+ 创建任务](#) ← 点击“创建任务”弹出空间清理配置窗口

请填写要搜索的内容

清理节点	清理时间	状态	操作
dn004	2018-08-09 16:42:43	清理完成	
dn003	2018-08-07 10:38:35	清理完成	
dn001	2018-08-06 22:09:03	清理完成	

共 3 条 < 1 > 10 条/页 跳至 页

创建任务

* 清理节点: dn001

* 清理开始时间: 00:00:00

* 清理结束时间: 23:00:00

清理时间段尽可能选择在业务低峰时间段

取消 确定

3.3.8、运维管理-冷热迁移

3.3.8.1、选择迁移粒度

TBase 默认按月迁移数据，而数据库中只能存在一种迁移粒度的冷热表，所以要先选择好迁移粒度，迁移粒度分为 year、month、day，由 cold_hot_separation_mode 控制，系统默认按月分区，所以 cold_hot_separation_mode= 'month'，如果要切换其它迁移粒度，需要先修改各个 cn、dn 节点参数

cold_hot_separation_mode, 修改后重启集群。

3.3.8.2、定义冷热节点

集群中最少要有两个 group，分别由冷节点、热节点组成。如果集群中已经有 default node group，那么只要再创建一个扩展 group 即可。扩展 group 中的 DN 节点必须是 default node group 以外的 DN 节点，如果没有就要添加新的 DN 节点用来创建扩展 group。

```
postgres=# create node group cold_group with(dn003,dn004);
CREATE NODE GROUP
postgres=# create extension sharding group to group cold_group;
CREATE SHARDING GROUP
postgres=# clean sharding;
CLEAN SHARDING
```

DN 节点默认为热节点，所以 psql 分别连接 dn003，dn004 节点，执行以下 sql，将其定义为冷节点：

```
postgres=# select pg_set_node_cold_access();
```

#如果冷节点定义错误，取消定义的冷节点：

```
postgres=# select pg_clear_node_cold_access();
```

3.3.8.3、创建冷热表

```
postgres=# CREATE TABLE cold_move_test(
    id int not null,
    identifynumber text NOT NULL,
    inserttimeforhis timestamp without time zone NOT NULL,
    primary key(id,identifynumber,inserttimeforhis))
partition by range (inserttimeforhis)
    begin (timestamp without time zone '2017-01-01')
    step (interval '1 month')
    partitions (24)
```

DISTRIBUTE BY SHARD (identifynumber,inserttimeforhis) to GROUP default_group cold_group;

冷热表必须是时间范围分区表，必须带有主键（发布订阅要求），shard key 必须带有时间范围分区字段。

3.3.8.4、创建 Replica identity

冷热迁移必须创建 Replica identity

```
postgres=# ALTER TABLE cold_move_test REPLICA IDENTITY DEFAULT;
```

如果要取消 Replica identity，命令如下：

```
postgres=# ALTER TABLE cold_move_test REPLICA IDENTITY nothing;
```

3.3.8.5、创建冷热迁移任务

The screenshot shows the '创建任务' (Create Task) dialog box with the following fields and annotations:

- * 任务名称:** A text input field for the task name.
- * 迁移粒度:** Radio buttons for '年' (Year), '月' (Month), and '日' (Day). An annotation points to the '年' option: '迁移类型，默认按年迁移，实例中只能存在其中一种迁移类型的冷热分区表'.
- * 迁移时间:** Two date pickers showing '1970-01'. An annotation points to the first date: '系统默认迁移开始时间'. Another annotation points to the second date: '迁移结束时间，不能超过当前系统时间'.
- 自动切换:** A toggle switch. An annotation points to it: '路由自动切换开关'.
- 自动清理:** A toggle switch. An annotation points to it: '是否自动清理热节点过期数据和发布订阅等信息'.
- Buttons: '取消' (Cancel) and '确定' (Confirm).

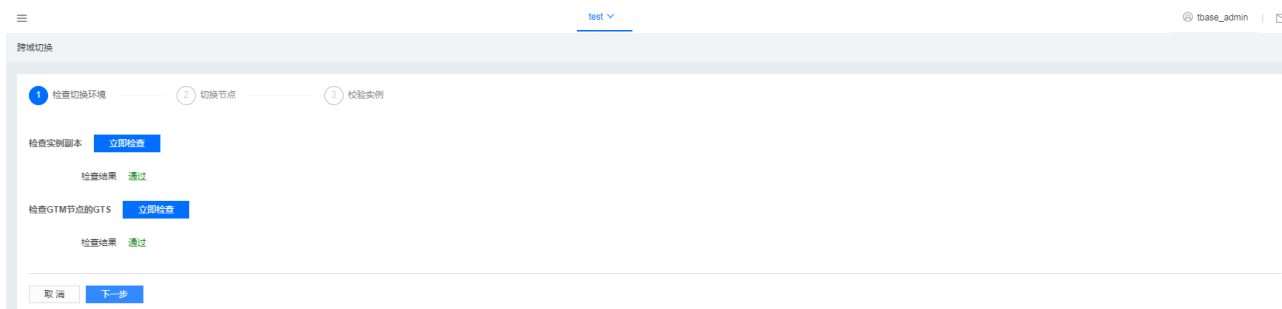
填写相关信息后点击确定即开始冷热迁移任务，等待迁移完成即可，如果迁移报错，请查看界面报错提示。

3.3.9、运维管理-跨域切换

3.3.9.1、检查切换环境

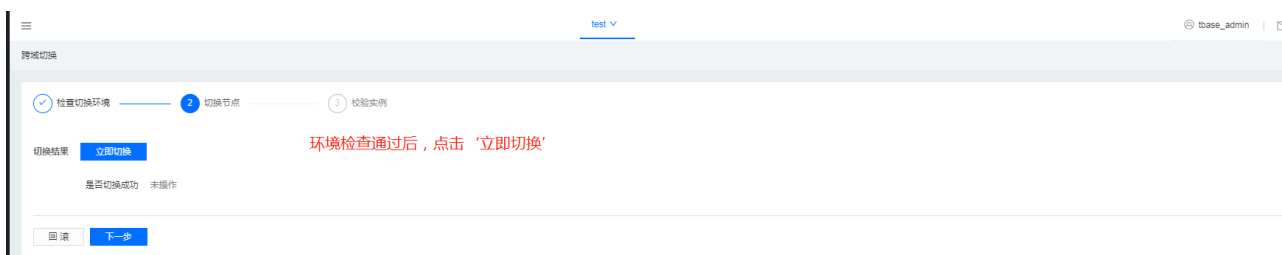
The screenshot shows the '跨域切换' (Cross-Domain Switch) page in the TBase console. The '检查切换环境' (Check Switch Environment) step is active, with the following details:

- 检查实例副本:** A button labeled '立即检查' (Check Immediately). An annotation points to it: '这里检查备region中副本数量是否足够，不能少于两个'.
- 检查结果:** 未检查 (Not Checked).
- 检查GTM节点的GTS:** A button labeled '立即检查' (Check Immediately). An annotation points to it: '检查gtm主备服务是否正常，gts是否一致'.
- 检查结果:** 未检查 (Not Checked).
- Buttons: '取消' (Cancel) and '下一步' (Next Step).



如果检查失败，根据界面报错提示去 Center 主日志中查找具体原因。

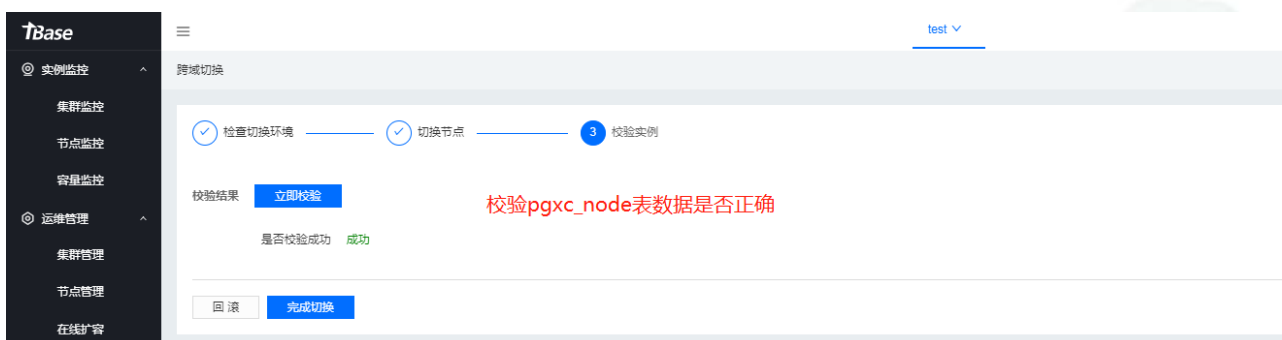
3.3.9.2、切换节点



切换时，gtm 备可能会出现瞬时性的失联，导致切换失败，只要再次点击‘立即切换’重试即可。



3.3.9.3、校验实例



3.3.10、运维管理-配置管理

■ 进入方法

集群运维平台-》运维管理-》配置管理

■ 相关功能

- Postgresql.conf 参数配置
- Pg_hba.conf 参数配置

3.3.10.1、postgresql.conf 参数配置

The screenshot displays the '数据库参数 (postgresql.conf)' configuration page. At the top, there are tabs for '数据库参数 (postgresql.conf)' and '验证配置 (pg_hba.conf)'. Below the tabs, a search bar contains the text 'log'. A table lists various configuration parameters with columns for '配置项', 'Context', '重启生效', '当前值', '描述', and '操作'. Parameters like 'log_checkpoints' and 'log_connections' are selected. A red box highlights the '配置项' column, with a note '这里勾选多个要配置的参数'. Another red box highlights the '操作' column, with a note '点击编辑只修改选择的这个参数'. A third red box highlights the 'log' search bar, with a note '这里过滤要配置的项目'. Below the table, there are pagination controls showing page 2 of 5.

The second part of the image shows a modal titled '修改数据库参数'. It contains a table with columns for '配置项', 'Context', '重启生效', '当前值', '修改值', '描述', and '操作'. Two parameters are shown: 'log_checkpoints' and 'log_connections'. The '修改值' column has input fields where 'on' is entered. A red box highlights the '修改值' column, with a note '配置值在这里修改'. Another red box highlights the '操作' column, with a note '不想修改的参数可以删除'. At the bottom left, there is a section '配置应用到' with a list of nodes: 'cn001', '172.16.0.37:11000', 'cn002', 'dn001', 'dn002', 'dn003', and 'dn004'. A red box highlights this section, with a note '这里配置要修改的节点'. At the bottom right, there are '取消' and '确定' buttons.

注意，需要重启的参数会中断业务使用

3.3.10.2、pg_hba.conf 参数配置

数据库参数 (postgresql.conf) [鉴权配置 \(pg_hba.conf \)](#)

系统默认配置 [用户配置](#)

选择节点实例: cn001 172.16.0.37:11000

[编辑](#) ← 点击编辑

Type	Database	User	CIDR-Address	Method
host	all	all	0.0.0.0/0	trust

< 1 >

数据库参数 (postgresql.conf) [鉴权配置 \(pg_hba.conf \)](#)

系统默认配置 [用户配置](#)

选择节点实例: cn001 172.16.0.37:11000

[编辑](#) [新增](#) [保存配置并下发](#) ← 修改完成点“保存配置并下发”即可生效

Type	Database	User	CIDR-Address	Method	操作
host	all	all	0.0.0.0/0	md5	删除

< 1 >

3.3.11、运维管理-备份管理

■ 进入方法

集群运维平台-》运维管理-》备份管理

■ 相关功能

- 配置备份规则
- 查看备份执行情况

3.3.11.1、配置备份规则

备份设置

备份检查

HDFS全局设置

HDFS 地址：

hdfs://172.16.0.12:9000/tmp

请以 hdfs://[address]:[port]/[directory] 格式填入地址（不能以 / 结尾），DN数据节点有读写该目录的权限

全量备份设置

状态：

☒

开始时间：

01

时

00

分

全量备份执行的时间间隔，建议选择2-5天

间隔时间：

3

天

全量备份保留份数取决于存储空间大小，建议选择大于2份

保留份数：

3

执行全量备份时通过限制带宽保证业务不受影响

带宽限制：

50

Mbit/s

增量备份设置

状态：

☒

并发数：

11

执行增量备份的并发数量，建议5-15

保存并应用

取消



点击保存并应用即可

3.3.11.2、查看备份执行情况

备份设置 备份检查

全量备份状态 增量备份状态 ← 点击这里显示增量备份执行情况

↑ 点击这里显示全量备份执行情况

请输入节点名称 Q

DN节点名称	备份开始状态	最近备份执行状态	最近备份开始时间	最近备份结束时间	操作
dn001	已经关闭	unknown	2018-07-07T21:00:06.762916+08:00	2018-07-07T21:00:06.762916+08:00	查看备份记录
dn001	已经关闭	unknown	2018-07-07T21:00:06.762916+08:00	2018-07-07T21:00:06.762916+08:00	查看备份记录

3.3.12、运维管理-在线升级

3.3.12.1、上传升级包

TBase

资源管理 软件包管理

+ 添加软件包 点击添加可用的软件包

请输入搜索的内容 Q

软件包名	添加时间	软件类型	包类型	版本号	操作
tbase_ggiz-2.13-x86_64.rpm	07-02 11:07	TbaseV2	安装包	2.13	详情 删除
tbase_ggiz-2.15-x86_64.rpm	07-01 15:05	TbaseV2	安装包	2.15	详情 删除

共 2 条 < 1 > 10 条页 跳至 页

TBase

资源管理 软件包管理 / 添加软件包

软件包信息

* 软件类型: Tbase V2 选择软件包类型，分为TBaseV2、TBaseV3、PostgreSQL三种类型

* 包类型: 安装包 包类型选择 安装包

* MD5校验值: 请填写MD5校验值 填写安装包的md5值

上传软件包:

将文件拖到此处，或点击上传

只能上传 .rpm 格式的文件，且文件命名规则为：
包名+版本号，示例：包名v1.03.rpm

取消

3.3.12.2、创建版本升级任务



创建升级任务

1. 版本变更涉及集群可用性，请谨慎确认后再操作。

2. 为确保版本变更后集群可用性，系统会**自动升级所选节点的相关节点**。

* 任务名称: UpgradeTo_2.13_6p6maRN

系统自动生成任务名称，也可以自定义任务名称

* 升级版本: 2.13

选择升级版本，如果升级版本与当前实例一样不会显示升级节点信息，可以低版本升高版本，也可以高版本回退到低版本

* 选择需要升级的节点

节点名称

gtm001
172.16.0.8:11008

gtm002
172.16.0.12:11008

gtm003
172.16.0.5:11008

gtm004
172.16.0.108:11008

cn001
172.16.0.5:11379

角色

备

备

备

主

主

原版本

2.15

2.15

2.15

2.15

2.15

选择升级节点，一般全选，升级整个实例所有节点

搜索节点 (名称、IP、端口、Q)

共 16 条

< 1 2 3 4 >

取消 下一步



创建升级任务



- ① 1.版本变更涉及集群可用性，请谨慎确认后再操作。
2.为确保版本变更后集群可用性，系统会**自动升级所选节点的相关节点**。

升级目标版本：2.13

升级节点汇总：
数据节点（主）2个
数据节点（备）6个
协调节点（主）1个
全局事务管理器（主）1个
全局事务管理器（备）3个
协调节点（备）3个

升级节点信息汇总

您选择的节点：

节点名称	角色	原版本
gtm001 172.16.0.8:11008	备	2.15
gtm002 172.16.0.12:11008	备	2.15
gtm003 172.16.0.5:11008	备	2.15
gtm004 172.16.0.108:11008	主	2.15
cn001 172.16.0.5:11379	主	2.15

< 1 2 3 4 >

系统选择的关联节点：

节点名称	角色	原版本
如果两个版本xlog不兼容，或者通信协议不一致，升级部分节点的话系统会自动把关联节点升级，这里显示关联节点信息，不需要用户去判断选择		

上一步

升级



任务名称	创建时间	当前进度	目标版本	状态	操作
UpgradeTo_2_13_6p6maRN	2019-07-05 15:10:53	0%	2.13	运行中	查看详情

最后等待升级完成即可，如果报错可以点击错误查看错误信息，定位原因。

3.3.13、运维管理-数据多活

■ 进入方法

集群运维平台-》运维管理-》数据多活

■ 相关功能

- 创建发布
- 创建订阅
- 查看发布状态
- 查看订阅状态

3.3.13.1、创建发布



发布名称	对象名称	创建时间	运行状态	支持数据变更操作	存储节点数量	操作
------	------	------	------	----------	--------	----



新建发布

选择按数据库发布或者按数据表发布

* 发布类型: ☒ 发布数据库 ☐ 发布数据表

* 发布对象名称: postgres

取消 确定

新建发布

* 发布类型: ☒ 发布数据库 ☐ 发布数据表

* 发布对象名称: postgres

若选择“发布数据库”，使用默认的数据库名称postgres

取消 确定

新建发布

* 发布类型: ☐ 发布数据库 ☒ 发布数据表

* 发布对象名称: public.name_tbl

若选择“发布数据表”，填写数据表名称（含schema作为前缀）

取消 确定

新建发布

* 发布类型: ☒ 发布数据库 ☐ 发布数据表

* 发布对象名称: postgres

点击“确定”完成发布的创建

取消 确定

3.3.13.2、创建订阅

创建订阅前，需要将订阅端实例所有 cn 节点（包括主备）加入到发布端实例所有 dn 节点（包括主备）tbase 用户的 pg_hba.conf 白名单中，可以在“运维管理-配置管理”中配置。如：

配置管理

数据库参数 (postgresql.conf) 鉴权配置 (pg_hba.conf) 定时任务

系统默认配置 用户配置

选择节点实例: dn001 10.53.62.223:11002 (读/写)

编辑 新增 保存配置并下发

Type	Database	User	CIDR-Address	Method	操作
host	all	tbase	10.53.62.6/32	trust	删除
host	all	tbase	10.53.62.27/32	trust	删除
host	all	tbase	10.53.62.30/32	trust	删除
host	all	tbase	10.53.62.31/32	trust	删除

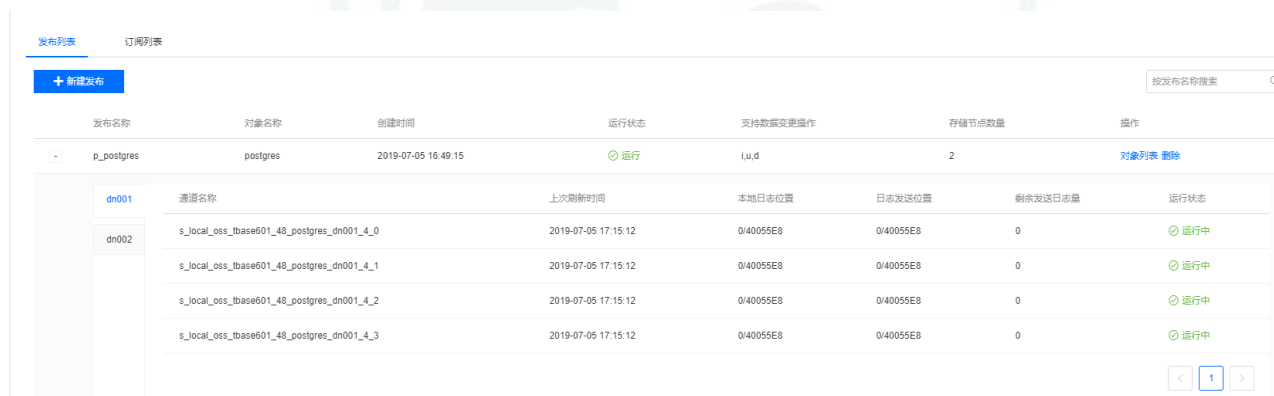
开始创建订阅：



如果要跨 OSS 创建订阅，需要先配置 OSS 联邦（参考后面的第 5 小节）

3.3.13.3、查看发布状态

发布创建后，可查看发布的状态：



3.3.13.4、查看订阅状态

订阅创建后，可查看订阅的状态：

发布列表		订阅列表					
+ 新建订阅		按订阅名称搜索					
订阅名称	对象名称	创建时间	运行状态	主键冲突告警	通道并行数量	操作	
- s_local_oss_tbase601_48_postgres	postgres	2019-07-05 17:04:16	运行	yes	4	对象列表 删除	
cn001	通道名称	上次更新时间	日志接收位置	日志重做位置	接收与重做差异	运行状态	
cn002	s_local_oss_tbase601_48_postgres_dn002_4_0	2019-07-05 17:16:15	0/4005608	0/4005608	0	运行中	
	s_local_oss_tbase601_48_postgres_dn002_4_1	2019-07-05 17:16:15	0/4005608	0/4005608	0	运行中	
	s_local_oss_tbase601_48_postgres_dn002_4_2	2019-07-05 17:16:15	0/4005608	0/4005608	0	运行中	
	s_local_oss_tbase601_48_postgres_dn002_4_3	2019-07-05 17:16:15	0/4005608	0/4005608	0	运行中	

3.3.14、告警

■ 进入方法

集群运维平台-》运维管理-》告警

■ 相关功能

- 查看告警规则
- 配置告警规则
- 查看告警记录并处理
- 配置告警信息接收信息

3.3.12.1、查看告警规则

告警列表

告警记录

告警管理

事件告警

指标告警

告警列表分两种类型，分别是事件和指标

新增告警

全部状态

输入告警名称搜索

Q

告警名称	告警指标名	指标描述	接收人	接收方式	告警开关	告警状态	屏蔽IP列表	操作
Center 存活检测失败	PGXZ_CHECK_CENTER_AVAILABLE_WARNING	Center 存活检测失败	无	☐ ☐ ☐		正常		编辑 删除
Agent 原主掉告警	PGXZ_CREATE_SLAVE_FAIL_WARNING	Agent 原主掉告警	无	☐ ☐		正常		编辑 删除
Agent 冷备告警	PGXZ_COLD_BACKUP_WARNING	Agent 冷备告警	无	☐ ☐		正常		编辑 删除
打开关闭 Agent AutoPull 超时告警	PGXZ_CONTROL_AGENT_RSP_WARNING	打开关闭 Agent AutoPull 超时告警	无	☐ ☐		正常		编辑 删除
Center 角色校验失败	PGXZ_CENTER_WARNING_ROLE	Center 角色校验失败	无	☐ ☐ ☐		正常		编辑 删除
Shard迁移任务告警	PGXZ_SHARD_WARNING	Shard迁移任务告警	无	☐ ☐		正常		编辑 删除
自动化任务告警	PGXZ_CLIENT_TASK_WARNING	自动化任务告警	无	☐ ☐		正常		编辑 删除

3.3.12.2、配置告警规则

告警列表告警记录告警管理

事件告警指标告警

新增告警

全部状态连接数

告警名称	告警条件	接收人	接收方式	告警开启	告警状态	屏蔽IP列表	操作
当前连接数 - DN Slave	同比告警 当前连接数 的阈...	无	☑ ☐ ☐	🔴	正常		编辑删除
当前连接数 - DN Master	同比告警 当前连接数 的阈...	无	☑ ☐ ☐	🔴	正常	点击编辑弹出配置窗口	编辑删除
当前连接数百分比	同比告警 当前连接数百分比...	无	☑ ☐ ☐	🔴	正常		编辑删除
TCP 连接数	同比告警 当前连接数 TCP 连接数 的阈...	无	☑ ☐ ☐	🔴	正常		编辑删除
当前连接数 - CN	同比告警 当前连接数 的阈...	无	☑ ☐ ☐	🔴	正常		编辑删除

共 5 条110 条/页 到第 页

编辑告警

* 告警名称: 当前连接数 - DN Master

* 告警指标: 节点告警 当前连接数

* 指标维度: Datanode Master

* 算法类型: 阈值

* 告警条件: 大于 大于 20

* 告警开启: 🔴

检测频率: 1分钟

屏蔽IP列表: 新增

接收人: 请选择接收人

接收方式: ☑ 邮件 ☑ 短信 ☑ 电话

* 是否轮转: 🔴

取消确定

根据需要设置相应的值，后按“确定”按钮即可。

3.3.12.3、告警记录查看并处理

告警列表 告警记录 告警管理

事件告警 规则告警

这里配置条件过滤告警记录

输入告警名称搜索 Q 待处理 最近一周

批量处理 批量处理告警记录

告警名称	告警指标名	异常信息	接收人	接收方式	告警时间	状态	处理说明
☒ Shard迁移任务告警	PGXZ_SHARD_WARNING	【PGXZ告警】 172.16.0.37 在 201...	无	☐ ☐	2018-08-09 17:03:45	待处理	单条处理告警记录
☒ Shard迁移任务告警	PGXZ_SHARD_WARNING	【PGXZ告警】 172.16.0.37 在 201...	无	☐ ☐	2018-08-09 16:53:14	待处理	
☐ Shard迁移任务告警	PGXZ_SHARD_WARNING	【PGXZ告警】 172.16.0.37 在 201...	无	☐ ☐	2018-08-09 16:42:28	待处理	
☐ Shard迁移任务告警	PGXZ_SHARD_WARNING	【PGXZ告警】 172.16.0.37 在 201...	无	☐ ☐	2018-08-09 16:40:15	待处理	

3.3.12.4、配置告警信息接收信息

告警列表 告警记录 告警管理

创建用户 点击“创建用户”即可弹出新创用户窗口

输入用户名搜索 Q

用户名	邮箱	手机号码	操作
admin	tomchen@tencent.com	13828000000	编辑 删除 点击删除，删除选定记录

共 1 条 < 1 > 10 条/页 跳至 页

创建用户

* 用户名: admin

* 邮箱: tomchen@tencent.com

* 手机号码: 13828000000

取消 确定

4、OSS 组件

4.1、OSS 是什么

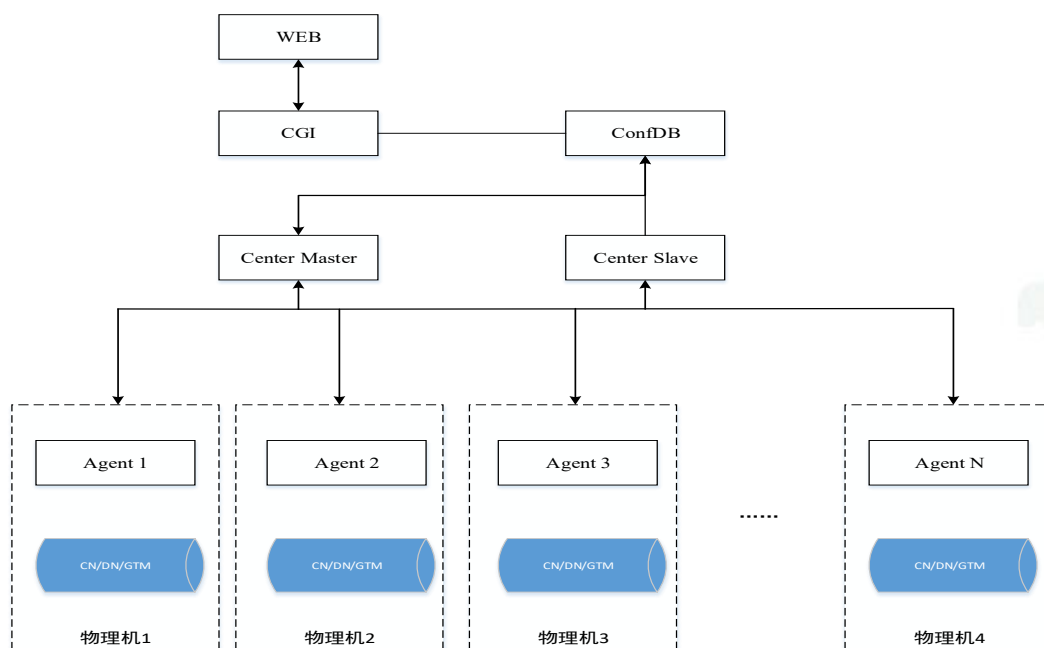
OSS 系统是 TBase 分布式数据库系统配套使用的平台管理系统，集租户管理，服务器资源管理，项目管理，监控运维管理。其中租户管理，服务器资源管理，项目管理是实现多租户管理配套组件。监控运维系统设计的主要目的是监控和维护 TBase 数据库，包括指标实时监控、告警，部分故障自动修复，在线扩容，数据搬迁等功能，与 TBase 数据库组成一套完整的体系，形成一个高效、稳定、可靠的分布式数据库系统。

4.2、OSS 设计目标

- 多租户管理，实现多个租户共用同一套管理平台
- 租户管理，实现单一租户管理多个集群项目
- 物理资源隔离管理，实现 tbase 云化能力
- 可对指定的 TBase 数据库系统进行实时指标监控及异常告警，告警形式多样且可选。
- 可通过 OSS 系统对配套的 TBase 数据库系统进行各类运维操作，其中包括集群起停、节点部署、升级、在线扩容/缩容、数据搬迁、数据备份等。
- 可对 TBase 数据库系统部分故障做自恢复处理。
- OSS 系统自身具备一定的健壮性和故障自恢复能力。

4.3、OSS 工作图及原理说明

OSS 系统与 TBase 系统的部署模式如下图所示



OSS 系统包括 5 个组件：WEB 页面、CGI、ConfdB、Center 和 Agent。其中 Center 包括 Center Master 和 Center Slave,ConfdB 包括 ConfdBMaster 和 ConfdBSlave,WEB、CGI、ConfdB 和 Center 都可以独立于 TBase 集群部署，其中 Agent 是与 TBase 集群的节点混合部署，每台部署了 TBase 数据库节点的物理机器上都需要部署一个 Agent。OSS 系统通过 Agent 实现对 TBase 集群中每个节点的状态监控和操作，并将采集的指标反馈给 Center，Center 将状态存储在 ConfdB 中，前端通过接口从 ConfdB 中取相关的状态数据，并显示在页面中。用户对 TBase 集群及其节点的操作，通过 web 下发到 Center，再下发到各个 Agent 或节点进行动作执行。

4.4、OSS 各组件说明

4.4.1、查看 OSS 各组件部署节点分布情况

```
[root@VM_0_3_centos conf]# pwd
/root/tbase_mgr/conf
[root@VM_0_3_centos conf]# cat role.info
#eth ip root password role( OssCenterMaster OssCenterSlave OssAgent ConfdBMaster ConfdBSlave Alarm
TStudio)
#Notice: The machine that will be installed OssCenterMaster or OssCenterSlave can't be installed OssAgent

eth0 172.16.0.37 root xxxxxxxx OssCenterMaster|ConfdBMaster|TStudio
eth0 172.16.0.42 root xxxxxxxx OssCenterSlave|ConfdBSlave|Alarm
eth0 172.16.0.3 root xxxxxxxx OssAgent
eth0 172.16.0.13 root xxxxxxxx OssAgent
[root@VM_0_3_centos conf]#
```

进入 TBase 安装目录下的 conf 目录，查看 role.info，里面有保留了当初的部署配置信息。

如果后期有添加新的机器，则查询 confdb 的 pgxzm_oss.tbl_oss_agent_info 可以得到最新的 center,agent 机器分布情况

```
[tbaseConfdB@VM_0_37_centos tbase_mgr]$ psql -h 127.0.0.1 -U postgres -d postgres -p 54321
psql (9.6.2)
Type "help" for help.
```

```
postgres=# select ip,port,role from pgxzm_oss.tbl_oss_agent_info ;
```

```
   ip      | port |  role
-----+-----+-----
 172.16.0.37 | 8118 | Master
 172.16.0.42 | 8118 | Slave
 172.16.0.3  | 8118 | Agent
 172.16.0.13 | 8118 | Agent
(4 rows)
```

```
postgres=#
```

4.4.2、confdb 说明

4.4.2.1、Confdb 服务介绍

Confdb 是一个单机版 Postgresql 数据库，采用主备高可用运行方式，主备节点部署在集群中的两部主机上面。Confdb 记录了当前 OSS 管理下的所有信息，如有哪些租户，哪些机器，哪些实例，实例的各种状态信息，当前任务，任务进度等内容。通过查询 Confdb 下的各个表，可以完全了解机器和实例信息。Confdb 的内容非常重要，当 OSS 重启时，Center 需要依赖 Confdb 中的元数据信息恢复当前实例情况，资源分配情况，任务进度情况等。**当需要手动修改 Confdb 内容时，须提前做好备份。**多个集群可以共用一个 Confdb 实例，每个集群对应该实例下的一个指标 Database

4.4.2.2、Confdb 服务程序

■ 安装路径

/data/tbase/confdb/postgres

4.4.2.3、Confdb 服务管理

■ 基于 OSS 服务管理脚本

启动服务

```
[root@VM_0_3_centos tbase_mgr]# sh tbase_mgr.sh start Confdb
```

停止服务

```
[root@VM_0_3_centos tbase_mgr]# sh tbase_mgr.sh stop Confdb
```

重启服务

```
[root@VM_0_3_centos tbase_mgr]# sh tbase_mgr.sh restart Confdb
```

查询服务状态

```
[root@VM_0_3_centos tbase_mgr]# sh tbase_mgr.sh status Confdb
```

4.4.2.4、Confdb 服务默认端口号

Confdb 服务默认的端口号 54321

4.4.2.5、Confdb 运行日志所在路径

/data/tbase/confdb/postgres/pg_log

4.4.2.6、连接 Confdb 服务

```
psql -h 172.16.0.3 -U postgres -d postgres -p 54321
```

Password for user postgres:

```
psql (10.6)
```

Type "help" for help.

```
postgres=#
```

然后就可以操作 Confdb 上面的业务数据，**当需要手动修改 Confdb 内容时，须提前做好备份。**

4.4.3、Center 说明

4.4.3.1、Center 服务介绍

Center 是 OSS 的决策中心，设计为一主多备。备 center 仅负责存储实例状态等信息，用于主 center 在需要的时候进行校验。通常 center 只负责发起任务，监控任务进度，而具体任务的执行会由对应的 agent 完成。Center 的作用如下

- 负责接受 cgi 的命令，并进行相应处理。如申请、初始化、回收、升级、重启等命令。
- 负责管理所有机器资源，管理各实例的资源消耗
- 负责收集 agent 上报的所有实例信息，并向 confdb 升级。
- 负责裁决是否发起主备切换。当主备 center 都正常时，才能发起主备切换。
- 负责发起全量备份

4.4.3.2、Center 服务程序

- 安装路径

```
/data/tbase/install/tbase_oss
```

4.4.3.3、Center 配置文件

- 配置文件所在路径

```
/data/tbase/install/tbase_oss/config
```

- 配置文件 tbase_oss_conf.ini 说明

这里配置 CenterMaster 和 CenterSlave 的 IP 地址和 PORT，CGI 的 IP 地址和 PORT，Center 连接 Confdb 的参数也在这里配置，如

```
;section for oss web service
```

```
;oss_ip 是 center 主备 ip，也是 CGI 的 ip，oss_port 是 CGI port
```

```
[oss]
```

```
oss_ip=172.16.0.37,172.16.0.42
```

```
oss_port=8080
```

```
;section for local
```

```
;role=Master 表示运行角色为 CenterMaster
```

```
;role=Slave 表示运行角色为 CenterSlave
```

```
;role=Agent 表示其运行角色为 Agent
```

```
;bind_net_frame=eth0 表示监控的网卡设备为 eth0
```

```
[local]
```

```
local_ip=172.16.0.37
```

```
local_port=8118
```

```
role=Master
```

```
os_user=tbase
```

```
bind_net_frame=eth0
```

```
;section for confdb, 下面是 confdb 连接配置信息
```

```
[confdb]
```

```
db_host=172.16.0.37
```

```
db_user=postgres
```

```
db_password=postgres
```

```
db_name=postgres
```

```
db_port=54321
```

■ 配置文件 shardmove_conf.ini 说明

这里配置数据迁移使用的，不需要修改

4.4.3.4、Center 服务管理

■ 基于 OSS 服务管理脚本

启动服务

```
/data/tbase/shell/center_start.sh
```

停止服务

```
/data/tbase/shell/center_stop.sh
```

查询服务状态

```
/data/tbase/shell/center_status.sh
```

应用程序升级

```
/data/tbase/shell/center_upgrade.sh
```

■ 基于 Center 服务管理脚本

启动服务

```
/data/tbase/install/tbase_oss/tools/op/start.sh
```

停止服务

```
/data/tbase/install/tbase_oss/tools/op/stop.sh
```

查询服务状态

```
/data/tbase/install/tbase_oss/tools/op/status.sh
```

4.4.3.5、Center 服务默认端口号

Center 服务默认的端口号 8118，CGI 的默认 port 为 8080

4.4.3.6、Center 运行日志所在路径

```
/data/tbase/install/tbase_oss/logs
```

4.4.4、Agent 说明

4.4.4.1、Agent 服务介绍

Agent 是 OSS 具体任务的执行者。在每台物理机上都部署一个 agent（仅一个）。具体操作实例的任务都需要由 Agent 来完成，如初始化、备份、重启、升级、回收、监控信息上报等。Agent 的作用如下

- 负责完成 Center 通知的任务，并回复 Center 任务进度
- 负责监控 Center 通知监控的实例，并实时上报 Center 具体实例状态情况
- 负责收集实例监控数据（cpu、磁盘等消耗），并上报给监控平台。
- 负责当实例异常时，对实例进行自动拉起。

4.4.4.2、Agent 服务程序

- 安装路径

```
/data/tbase/install/tbase_oss
```

4.4.4.3、Agent 配置文件

- 配置文件所在路径

```
/data/tbase/install/tbase_oss/config
```

■ 配置文件 agent_mcd0.conf 说明

■ 配置文件所在路径

/data/tbase/install/tbase_oss/config

■ 配置文件 tbase_oss_conf.ini 说明

这里配置 CenterMaster 和 CenterSlave 的 IP 地址和 PORT, CGI 的 IP 地址和 PORT, Center 连接 ConfdB 的参数也在这里配置, 如

```
;section for oss web service
```

;oss_ip 是 center 主备 ip, 也是 CGI 的 ip, oss_port 是 CGI port

```
[oss]
```

```
oss_ip=172.16.0.37,172.16.0.42
```

```
oss_port=8080
```

```
;section for local
```

```
;role=Master 表示运行角色为 CenterMaster
```

```
;role=Slave 表示运行角色为 CenterSlave
```

```
;role=Agent 表示其运行角色为 Agent
```

```
;bind_net_frame=eth0 表示监控的网卡设备为 eth0
```

```
[local]
```

```
local_ip=172.16.0.37
```

```
local_port=8118
```

```
role=Agent
```

```
os_user=tbase
```

```
bind_net_frame=eth0
```

;section for confdb, 下面是 confdb 连接配置信息

```
[confdb]
```

```
db_host=172.16.0.37
```

```
db_user=postgres
```

```
db_password=postgres
```

```
db_name=postgres
```

```
db_port=54321
```

4.4.4.4、Agent 服务管理

■ 基于 OSS 服务管理脚本--当前操作集群

启动服务

```
/data/tbase/shell/agent_start.sh
```

停止服务

```
/data/tbase/shell/agent_stop.sh
```

查询服务状态

```
/data/tbase/shell/agent_status.sh
```

■ 基于 OSS 服务管理脚本--所有 Agent

启动服务

```
/data/tbase/shell/agent_all_start.sh
```

停止服务

```
/data/tbase/shell/agent_all_stop.sh
```

查询服务状态

```
/data/tbase/shell/agent_all_status.sh
```

升级 agent

```
/data/tbase/shell/agent_upgrade.sh
```

■ 基于 Agent 服务管理脚本

启动服务

```
/data/tbase/install/tbase_oss/tools/op/start.sh
```

停止服务

```
/data/tbase/install/tbase_oss/tools/op/stop.sh
```

查询服务状态

```
/data/tbase/install/tbase_oss/tools/op/status.sh
```

4.4.4.5、Agent 运行日志所在路径

```
/data/tbase/install/tbase_oss/logs
```

4.4.5、Alarm 说明

4.4.5.1、Alarm 服务介绍

Alarm 是 OSS 告警服务管理者，分为事件告警和指标告警，设备只需要在一台物理机上部署，一般情况下 CGI 服务部署一起即可，Alarm 作用如上

- 负责告警项目管理
- 生成告警记录
- 发送告警通知（邮件，短信都可以）

4.4.5.2、Alarm 服务程序

■ 安装路径

/usr/local/pgxz_monitor

4.4.7.3、Alarm 配置文件

■ 事件告警服务 pgxzEventAlarmService

/usr/local/pgxz_monitor/pgxzEventAlarmService/config/monitor01.properties

/usr/local/pgxz_monitor/pgxzEventAlarmService/qcloud/pgxz_qcloud.conf

这里用于配置 confdb 的连接信息

pgsql.dbhost=172.16.0.37

pgsql.port=54321

pgsql.user=postgres

pgsql.password=

pgsql.dbname=tbase

pgsql.flushtime=300

pgsql.cleardate=90

#server 端口

http.port=9996

#与 center 通信接口

alarm.interface=http://172.16.0.37:8080/api/v1/alarm/

/etc/sysconfig/pgxzeventalarmd

这里用于配置 jdk 环境变量

JAVA_HOME=/usr/local/install/jdk

JDK_PATH=\${JAVA_HOME}/bin:\${JAVA_HOME}/jre/bin

JDK_CLASSPATH=\${JAVA_HOME}/lib:\${JAVA_HOME}/jre/lib

PATH=\${JDK_PATH}:\${PATH}

CLASSPATH=.:\${JDK_CLASSPATH}:\${CLASSPATH}

OPTS="-Xms512m -Xmx512m"

■ 指标告警服务 pgxzMetricAlarmService

/usr/local/pgxz_monitor/pgxzMetricAlarmService/config/metricAlarm.properties

/usr/local/pgxz_monitor/pgxzMetricAlarmService/qcloud/pgxz_qcloud.conf

这里用于配置 confdb 的连接信息及 CGI url 地址

```
#db
pgsql.dbhost=172.16.0.37
pgsql.port=54321
pgsql.user=postgres
pgsql.password=
pgsql.dbname=tbase
pgsql.flushtime=300
pgsql.cleardate=90

#server 端口
http.port=9997

#与 center 的通信接口
pgsql.inf.url=http://172.16.0.37:8080/api/v1/alarm/

#others
analysis.ThreadPoolSize=5
#与 center 的通信接口
alarm.interface=http://172.16.0.37:8080/api/v1/alarm/
```

/etc/sysconfig/pgxzeventalarmd

这里用于配置 jdk 环境变量

```
JAVA_HOME=/usr/local/install/jdk
JDK_PATH=${JAVA_HOME}/bin:${JAVA_HOME}/jre/bin
JDK_CLASSPATH=${JAVA_HOME}/lib:${JAVA_HOME}/jre/lib

PATH=${JDK_PATH}:${PATH}
CLASSPATH=.:${JDK_CLASSPATH}:${CLASSPATH}

OPTS="-Xms512m -Xmx512m"
```

■ web 服务程序

/usr/local/pgxz_monitor/pgxzWebService/bin/application.properties

这里用于配置 confdb 的连接信息，事件服务器地址，指标服务器地址及与 center 的通信接口

```
spring.datasource.url=nothing
spring.datasource.dbhost=172.16.0.37
spring.datasource.port=54321
spring.datasource.username=postgres
```

```

spring.datasource.password=
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.max-idle=10
spring.datasource.max-wait=10000
spring.datasource.min-idle=3
spring.datasource.initial-size=3
spring.datasource.validation-query=SELECT 1
spring.datasource.test-on-borrow=false
spring.datasource.test-while-idle=true
spring.datasource.time-between-eviction-runs-millis=18800

```

```

# web server port
server.port=8082

```

```

# pgxzm interface url
pgxzm.interface.url=http://172.16.0.37:8080/api/v1/alarm/

```

```

# alarm module ip and port
alarm.module.uri=172.16.0.42:9996,172.16.0.42:9997

```

/etc/sysconfig/pgxzwebservice

```

OPTS="-Xms1024m -Xmx2048m"

```

```

JAVA_HOME=/usr/local/install/jdk
JDK_PATH=${JAVA_HOME}/bin:${JAVA_HOME}/jre/bin
JDK_CLASSPATH=${JAVA_HOME}/lib:${JAVA_HOME}/jre/lib

```

```

PATH=${JDK_PATH}:${PATH}
CLASSPATH=.:${JDK_CLASSPATH}:${CLASSPATH}

```

4.4.7.4、Alarm 服务管理

■ 基于 OSS 服务管理脚本

启动服务

```
[root@VM_0_3_centos tbase_mgr]# sh tbase_mgr.sh start Alarm
```

停止服务

```
[root@VM_0_3_centos tbase_mgr]# sh tbase_mgr.sh stop Alarm
```

重启服务

```
[root@VM_0_3_centos tbase_mgr]# sh tbase_mgr.sh restart Alarm
```

查询服务状态

```
[root@VM_0_3_centos tbase_mgr]# sh tbase_mgr.sh status Alarm
```

■ 基于 Alarm 服务管理脚本

启动服务

```
[root@VM_0_3_centos tbase_mgr]# /etc/rc.d/init.d/pgxzeventalarmd start
[root@VM_0_3_centos tbase_mgr]# /etc/rc.d/init.d/pgxzmetricalarmd start
[root@VM_0_3_centos tbase_mgr]# /etc/rc.d/init.d/pgxzwebserviced start
```

停止服务

```
[root@VM_0_3_centos tbase_mgr]# /etc/rc.d/init.d/pgxzeventalarmd stop
[root@VM_0_3_centos tbase_mgr]# /etc/rc.d/init.d/pgxzmetricalarmd stop
[root@VM_0_3_centos tbase_mgr]# /etc/rc.d/init.d/pgxzwebserviced stop
```

查询服务状态

```
[root@VM_0_3_centos tbase_mgr]# /etc/rc.d/init.d/pgxzeventalarmd status
[root@VM_0_3_centos tbase_mgr]# /etc/rc.d/init.d/pgxzmetricalarmd status
[root@VM_0_3_centos tbase_mgr]# /etc/rc.d/init.d/pgxzwebserviced status
```

4.4.7.5、Alarm 运行日志所在路径

■ 事件告警服务 pgxzEventAlarmService

```
/usr/local/pgxz_monitor/pgxzEventAlarmService/log
```

■ 指标告警服务 pgxzMetricAlarmService

```
/usr/local/pgxz_monitor/pgxzMetricAlarmService/log
```

■ web 服务程序

```
/usr/local/pgxz_monitor/pgxzWebService/log
```

5、TBase 组件管理

TBase 组件管理主要是给运维的同学详细的介绍在没有监控运维平台的情况下如何搭建和运维一个 TBase 集群。

5.1、部署机器准备

5.1.1、测试数据盘的随机同步写入性能

```
time dd if=/dev/zero of=test bs=8k count=51200 oflag=dsync
```

```
51200+0 records in
51200+0 records out
419430400 bytes (419 MB) copied, 3.55592 s, 118 MB/s
```

```
real 0m3.557s
user 0m0.009s
sys 0m0.994s
```

参考，普通 sata ssd 单盘写入能力大约是 30-50mb/s，12 个 sas 1.2T 盘+带电带缓存的 raid 卡，写入能力大约 30-50mb/s，ts85 机型可以到达 100mb/s

5.1.2、创建运行环境

■ 创建操作系统运行用户

```
[root@VM_0_3_centos ~]# adduser -d /data/tbase tbase
```

■ 配置操作系统用户密码

```
[root@VM_0_3_centos pgxztmp]# passwd tbase
```

■ 创建 pgxz 应用程序运行目录

```
[root@VM_0_3_centos pgxz]# su tbase
[tbase@VM_0_3_centos install]$ mkdir -p /data/tbase/install/pgxz
```

■ 创建 pgxz 数据目录

```
[root@VM_0_3_centos pgxz]# su tbase
[tbase@VM_0_3_centos install]$ mkdir -p /data/tbase/data/pgxz
```

■ 上传 pgxz 应用程序到目录/data/tbase/install/pgxz 下面，目录的结构为

```
[tbase@VM_0_3_centos pgxz]$ ll /data/pgxz/install/pgxz
total 4
drwxrwxr-x 2 pgxz pgxz 4096 Nov 10 04:33 bin
drwxrwxr-x 4 pgxz pgxz 189 Nov 10 04:33 include
drwxrwxr-x 4 pgxz pgxz 172 Nov 10 04:33 lib
drwxrwxr-x 3 pgxz pgxz 24 Oct 1 14:54 share
[tbase@VM_0_3_centos pgxz]$
```

■ 配置 pgxz 用户的环境变量

```
[tbase@VM_0_3_centos ~]$ vim /data/tbase/.bashrc
# .bashrc
```

```
# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
```

fi

```
export TBASE_HOME=/data/tbase/install/pgxz
export PATH=$TBASE_HOME/bin:$PATH
export LD_LIBRARY_PATH=$TBASE_HOME/lib:${LD_LIBRARY_PATH}
PGHOST=127.0.0.1
PGUSER=tbase
PGDATABASE=postgres
PGPORT=15432
```

```
export PGHOST PGUSER PGDATABASE PGPORT
```

```
alias l='ls -lrt'
```

5.1.3、校准机器的时间

```
[root@VM_0_3_centos pgxz]# crontab -e
*/20 * * * * /usr/sbin/ntpdate ntpupdate.tencentyun.com >/dev/null &
```

校准服务器可自行选择

5.1.4、关闭 selinux

```
[root@VM_0_3_centos pgxz]# vim /etc/sysconfig/selinux
SELINUX=disabled
```

5.1.5、配置防火墙

■ iptables 配置(centos6 默认 firewall)

下面提供多种方案自己按需要选择

1)、不启用防火墙

```
[root@VM_0_3_centos pgxz]# service iptables stop
```

2)、配置 5432 port 可以对外提供服务

```
[root@VM_0_3_centos pgxz]# vim /etc/sysconfig/iptables
-A INPUT -m state --state NEW -m tcp -p tcp --dport 5432 -j ACCEPT
[root@VM_0_3_centos pgxz]# service iptables restart
```

3)、配置某个网段可以访问服务

```
[root@VM_0_3_centos pgxz]# vim /etc/sysconfig/iptables
-A INPUT -s 192.168.0.0/24 -j ACCEPT
[root@VM_0_3_centos pgxz]# service iptables restart
```

4)、配置某个 ip 可以访问服务

```
[root@VM_0_3_centos pgxz]# vim /etc/sysconfig/iptables
-A INPUT -s 192.168.0.231/32 -p tcp --dport 5432 -j ACCEPT
[root@VM_0_3_centos pgxz]# service iptables restart
■ firewalld 配置 (centos7 默认 firewall)
```

1)、关闭防火墙

```
systemctl stop firewalld.service
```

2) 禁止 firewall 开机启动

```
systemctl disable firewalld.service
```

5.1.6、修改 limits.conf 参数

```
[root@VM_0_3_centos pgxz]# vim /etc/security/limits.conf
#准许单个进程打开文件数
* soft    nofile    131072
* hard    nofile    131072
#准放运行的进程数
* soft    nproc     131072
* hard    nproc     131072
#限制内核文件的大小
* soft    core      unlimited
* hard    core      unlimited
```

```
[root@VM_0_3_centos pgxz]# rm /etc/security/limits.d/* -f
```

5.1.7、修改内核参数 sysctl.conf

```
[root@VM_0_3_centos pgxz]# vim /etc/sysctl.conf
#所有在消息队列中的消息总和的最大值(msgmnb=64k)
kernel.msgmnb = 65536
#指定内核中消息队列中消息的最大值(msgmax=64k)
kernel.msgmax = 65536
#共享内存总量，以页为单位（4K/页），默认这个值足够大了
kernel.shmall = 4294967296
```

#单个共享内存段的最大值，这个值需要大到让应用的内存段不用分成多个创建，否则性能会下降

```
kernel.shmmax=137438953472
```

#可以创建的内存段数，默认这个值足够大了

```
kernel.shmmni = 4096
```

#4 个数据分别对应:SEMMSL、SEMMNS、SEMOPM、SEMMNI 这四个核心参数

#SEMMSL：用于控制每个信号集的最大信号数量，建议最小值为 postgresql 最大连接数+10

#SEMOPM：每个 semop 系统调用可以执行的信号操作的数量。建议设置 SEMOPM 等于 SEMMSL。

#SEMMNI：内核参数用于控制整个 Linux 系统中信号集的最大数量，建议 128

#SEMMNS：用于控制整个 Linux 系统中信号（而不是信号集）的最大数，为 SEMMSL * SEMMNI

```
kernel.sem = 50100 64128000 50100 1280
```

#准许系统所有进程一共可以打开的文件数量

```
fs.file-max = 6553600
```

#系统允许的最大异步 IO 队列长度

```
fs.aio-max-nr = 1048576
```

#增大 locale port 数量，默认是从 32768 开始

```
net.ipv4.ip_local_port_range = 1024 65535
```

#keepalive 认定探测失效时间，默认是 2 小时，7200 秒

```
net.ipv4.tcp_keepalive_time = 60
```

#在认定连接失效之前，发送多少个 TCP 的 keepalive 探测包。缺省值是 9

```
net.ipv4.tcp_keepalive_probes = 6
```

#每隔多少秒重新发送 keepalive 探测包，默认是 75 秒

```
net.ipv4.tcp_keepalive_intvl = 10
```

```
net.ipv6.conf.all.forwarding=1
```

```
net.ipv4.ip_forward=1
```

```
vm.dirty_background_bytes = 102400000
```

上面的所有部署完成后需要重启机器生效

5.1.8、修改/etc/ssh/sshd_config 配置

```
vi /etc/ssh/sshd_conf
```

```
UseDNS no
```

修改完成后重启 sshd 服务

```
systemctl restart sshd.service
```

5.1.9、配置机器之间互信

■ 生成 id_rsa 和 id_rsa.pub 文件

```
ssh-keygen
```

■ 分发 id_rsa.pub 文件

```
ssh-copy-id 172.16.0.29
```

```
ssh-copy-id 172.16.0.47
```

每台机器都需要配置，配置完成后需要人工登录一次

5.1.10、配置 tbase 用户的 sodu 权限

```
vim /etc/sudoers.d/tbase
```

内容如下

```
tbase  ALL=(ALL)      NOPASSWD: ALL
```

如果有基本配置文件存也，也可以该内容配置到其它文件中即可

5.2、GTM “主” 组件

5.2.1、初始化

```
mkdir -p /data/tbase/data/pgxz
```

```
initgtm -Z gtm -D /data/tbase/data/pgxz/gtm
```

5.2.2、运行参数配置

■ gtm.conf

#配置 gtm 节点名称

```
nodename = gtm_1
```

#允许服务监听范围，*允许监听所有 IP 地址

```
listen_addresses = '*'
```

#监听端口号

```
port = 21000
```

#做为 GTM 主节点提供服务

```
startup = ACT
```

#客户端 idle60 秒后进行存活检查

```
keepalives_idle = 60
```

#每轮检查间隔为 10 秒

```
keepalives_interval = 10
```

#检查总次数为 6 次

```
keepalives_count = 6
```

#gtm 节点日志文件保存地点，可以使用绝对路径也可以使用相对路径保存于 gtm 节点目录下
log_file = 'gtm.log'

#日志级别配置，级别越高，记录的日志就越多

#默认为 WARNING.

可用值有: DEBUG, DEBUG5, DEBUG4, DEBUG3,

DEBUG2, DEBUG1, INFO, NOTICE, WARNING,

ERROR, LOG, FATAL, PANIC

log_min_messages = WARNING

#状态是否同步到 GTM 备节点

synchronous_backup = on

5.2.3、服务管理

■ 启动服务

```
gtm_ctl -Z gtm -D /data/tbase/data/pgxz/gtm start
server starting
[tbase@VM_0_3_centos install]$ TBase create 6 worker thread.
Start sever loop start thread count 10 running thread count 10.
TBase GTM is ready to go!!
```

■ 关闭服务

```
gtm_ctl -Z gtm -D /data/tbase/data/pgxz/gtm stop
waiting for server to shut down.... done
server stopped
```

■ 显示状态

```
gtm_ctl -Z gtm -D /data/tbase/data/pgxz/gtm status -H 127.0.0.1 -P 21000
gtm_ctl: server is running (PID: 8602)
current node [primary] gts[1633541452] xlog [0/0]
"-D" "/data/tbase/data/pgxz/gtm"
```

如果有备 gtm 存在则显示为

```
gtm_ctl: server is running (PID: 8602)
current node [primary] gts[1747689305] xlog [0/0]
[standby/sync] host[172.16.0.47:21000] gts [2047689083] xlog[0/0]
"-D" "/data/tbase/data/pgxz/gtm"
```

5.3、GTM “备” 组件

5.3.1、初始化

```
mkdir -p /data/tbase/data/pgxz
initgtm -Z gtm -D /data/tbase/data/pgxz/gtm
```

5.3.2、运行参数配置

■ gtm.conf

#配置 gtm 节点名称

```
nodename = gtm_2
```

#允许服务监听范围，*允许监听所有 IP 地址

```
listen_addresses = '*'
```

#监听端口号

```
port = 21000
```

#做为 GTM 备节点提供运行

```
startup = STANDBY
```

#指主 GTM 主节点信息

```
active_host = '172.16.0.29'
```

```
active_port = 21000
```

#客户端 idle60 秒后进行存活检查

```
keepalives_idle = 60
```

#每轮检查间隔为 10 秒

```
keepalives_interval = 10
```

#检查总次数为 6 次

```
keepalives_count = 6
```

#gtm 节点日志文件保存地点，可以使用绝对路径也可以使用相对路径保存于 gtm 节点目录下

```
log_file = 'gtm.log'
```

#日志级别配置，级别越高，记录的日志就越多

#默认为 WARNING.

可用值有: DEBUG, DEBUG5, DEBUG4, DEBUG3,

DEBUG2, DEBUG1, INFO, NOTICE, WARNING,

ERROR, LOG, FATAL, PANIC

```
log_min_messages = WARNING
```

5.3.3、服务管理

■ 启动服务

```
gtm_ctl -Z gtm -D /data/tbase/data/pgxz/gtm start
TBase create 6 worker thread.
Start sever loop start thread count 7 running thread count 8.
TBase GTM is ready to go!!
```

■ 关闭服务

```
gtm_ctl -Z gtm -D /data/tbase/data/pgxz/gtm stop
```

■ 显示状态

```
[tbase@VM_0_47_centos gtm]$ gtm_ctl -Z gtm -D /data/tbase/data/pgxz/gtm status -H 127.0.0.1 -P 21000
gtm_ctl: server is running (PID: 7498)
current node [standby] gts[2143234085] xlog [0/0]
"-D" "/data/tbase/data/pgxz/gtm"
```

5.4、condinator(cn)组件

5.4.1、初始化

```
initdb --locale=zh_CN.UTF-8 -U tbase -E utf8 -D /data/tbase/data/pgxz/cn01 --nodename=cn01 --nodetype=coordinator
--master_gtm_nodename gtm_1 --master_gtm_ip 172.16.0.29 --master_gtm_port 21000
```

5.4.2、运行参数配置

■ postgresql.conf 参数

```
#每个 cn 节点名字不同
pgxc_node_name = 'cn01'
#允许服务监听范围, *允许监听所有 IP 地址
listen_addresses = '0.0.0.0'
#监听端口号, 5432 是 tbase 默认的监听 port, 如果与 cn 或者备机安装在一起的话, 注意 port 不能相同
port = 15432

#配置 cn 连接池监听端口号
pooler_port = 22001
#配置连接池大小
# max_pool_size >= (max_connection * dn 节点数 + cn 节点数); 按实际计算
max_pool_size = 65535

#数据库最大的并发连接数
max_connections = 2000
```

```

#需要保留 super 的操作连接数
superuser_reserved_connections = 20

#指定不活动多少秒之后通过 TCP 向客户端发送一个 keepalive 消息
tcp_keepalives_idle = 60
#指定在多少秒之后重发一个 keepalive 消息
tcp_keepalives_interval = 10
#指定与客户端的服务器连接被认为死掉之前发送的次数
tcp_keepalives_count = 6

#设置数据库服务器将使用的共享内存缓冲区大小，这个值一般建议是不要超过 1/4 物理内存大小
#调整该值可以根据查询的命中率来调整
shared_buffers = 16GB

#二阶段提交最大数目，要大于 max_connections
max_prepared_transactions = 20000

#配置内部排序操作或哈希表使用最大内存量，超出这个值会写到临时磁盘文件
#这个默认值不宜过大，特别是并发连接数在高的应用，session 中视需要可以单独调整
work_mem = 4MB

#指定在维护性操作（例如 CREATE INDEX）中使用的 最大的内存量，这个值可以在 session 中配置
maintenance_work_mem = 1024MB

#配置 xlog 日志级别，其值为 minimal, replica, or logical
#日志写入量为 logical>replica>minimal，主备复制配置为 replica，逻辑复制配置成 logical
wal_level = logical
#指定来自后备服务器或流式基础备份客户端的并发连接的最大数量
#数据并发迁移时需要较大数量的 sender 进程
max_wal_senders = 64
#指定服务器可以支持的复制槽,把它的值设置为高于 max_wal_senders
max_replication_slots = 64
#指定逻辑复制工作的最大数量
max_logical_replication_workers = 64
#每个订阅的最大同步工作者数量
max_sync_workers_per_subscription = 4
#为后备服务器保留的 xlog 文件数
wal_keep_segments = 1000
#指定一个支持同步复制的后备服务器的列表，如果使用异步复制，则这里不能打开
synchronous_standby_names = ''

#控制一个事务是否需要等待 WAL 记录被写入磁盘
#当性能比完全确保事务的持久性更重要时，关闭 synchronous_commit 可以作为一个有效的代替手段。
#参数值为 off, local, remote_write 和 on,如果 synchronous_standby_names 被设置时
#这个参数也控制事务提交是否将等待事务的 WAL 记录被复制到后备服务器上。
#该参数可以在 session 中进行配置，即应用能够控制不同业务使用不同的同步方式

```

#当设置为 off 时，在向客户端报告成功和真正事务保证之间会有延迟

#（最大的延迟是 wal_writer_delay 的三倍）。不同于 fsync，

#将这个参数设置为 off 不会产生数据库不一致性的风险

synchronous_commit = on

#控制在一个检查点之后的页面的第一次修改期间将每个页面的全部内容写到 WAL 中。

#把这个参数关闭会加快正常操作，但是在系统失败后可能导致不可恢复的数据损坏

full_page_writes = on

#用于控制还未写入磁盘的 WAL 数据的共享内存量

wal_buffers = 16MB

#启用归档

archive_mode = on

#归档执行命令，实际命令参考备份归档文档,如

#DIR=/home/postgres/archive/'date +%F'; test ! -d \$DIR && mkdir \$DIR; test ! -f \$DIR/%f && cp %p \$DIR/%f

#如果暂时不需要归档可以这样配置

archive_command = 'bin/date'

#配置多少秒后如果还有产生新的 xlog，则服务器将切换到一个新的段文件，0 表示禁用

archive_timeout = 0

#指定备角色期间是否可以连接查询

hot_standby = on

#延长备机查询与归档恢复冲突时 timeout 时间，默认是 30s，值 -1 允许后备机一直等到冲突查询结束

max_standby_archive_delay = 1800s

#延长备机查询与流复制冲突时 timeout 时间，默认是 30s 值 -1 允许后备机一直等到冲突查询结束

max_standby_streaming_delay = 1800s

#在每次写入脏缓冲区后，后台写入器进程休眠时间，默认为 200ms

bgwriter_delay = 10ms

#控制每次最大多少个缓冲区将被后台写入器写出

bgwriter_lru_maxpages = 500

#用于计算每次多少个缓冲区被后台写入器写出

bgwriter_lru_multiplier = 2.0

#设置顺序扫描磁盘页面，获取一次数据的代价

seq_page_cost = 1.0

#设置非顺序扫描磁盘页面，获取一次数据的代价,如果是 ssd 的话，这个值可以设置为 2

random_page_cost = 4.0

#自动 WAL 检查点之间的最长时间，以秒计，增大这个参数可以降低 xlog 的写入量

checkpoint_timeout = 10min

#自动 WAL 检查点之间的最大的 xlog 文件数

#建议值 max_wal_size=shared_buffers * 0.75，min_wal_size=max_wal_size*0.25

max_wal_size = 12GB

```

min_wal_size = 3GB
#控制 checkpoint 对 shared buffers 的写入完成时间，如两次 checkpoint 的时间差是 60 秒
#则由于 checkpoint 而引发的 IO 操作会被平均分配在 60*0.5=30s 之内完成
#这个参数值大则 IO 负载变得平滑，避免在做 checkpoint 操作时出现的 IO 尖峰
#但持续时间长 IO 使用率变得过低，合理的参数值可以在日志表中收集一下 checkpoint 参数后进行调整
#checkpoint complete: wrote 6518 buffers (1.2%);
#0 transaction log file(s) added, 0 removed, 2 recycled;
#write=149.047 s, sync=1.485 s, total=150.582 s; sync files=70, longest=0.192 s, average=0.021 s
#理想的值为
checkpoint_completion_target = 0.93
#两次 checkpoint 发生的时间并为多少秒时，那么就向服务器日志写一个消息
checkpoint_warning = 30s

#告诉服务器，系统可能使用到的内存数
effective_cache_size = 250GB

#日志相关配置
#用户访问日志格式
log_destination = 'csvlog'
#启用用户访问日志收集器
logging_collector = on
#日志存放目录，可以是相对路径（相对了 data 目录）或绝对路径
log_directory = 'log'
#设置日志文件的权限
log_file_mode = 0600
#按我先前经常是只有 log_rotation_age 这个值有效果
log_truncate_on_rotation = 'on'
#120 分钟切换一次
log_rotation_age = 120
#1024MB 切换一次
log_rotation_size = 1024MB
#会话的当前执行命令保留长度，用于在字段 pg_stat_activity.query 中显示
track_activity_query_size = 4096
#配置 sql 语句执行超过多少毫秒数时，语句将被记录,值为-1 时禁用，0 时记录所有语句
log_min_duration_statement = 1000
#是否记录 checkpoint
log_checkpoints = on
#是否记录连接
log_connections = on
#是否记录断开连接
log_disconnections = on
#是否记录 autovacuum 执行日志
log_autovacuum_min_duration = 0
#是否记录所有语句的执行时间，值为 on 时将单独记录所有语句的执行时间，这里不记录语句
log_duration = off
#配置日志记录的内容

```

```

log_line_prefix = '%m %a %r %d %u %p %x'
#控制那些类型的语句被记录, 有效值是 none (off)、ddl、mod 和 all (所有语句)。
#如果 log_min_duration_statement 值为 0 时, 则 log_statement 什么值的效果都一样
#如果 log_min_duration_statement 值大于 0, 并且 log_statement 为 ddl 则 ddl 语句全部表被记录, 为两条 log
#dml 超时才被记录, 为一条记录
log_statement = 'none'
#设置在服务器日志中写入的时间戳的时区
log_timezone = 'PRC'
log_filename = 'postgresql-%A-%H.log'

#打开 autovacuum,进行垃圾数据自动回收
#打开垃圾回收及系统信息统计, 注意 /etc/hosts 里面的 localhost 解释要打开
track_counts = on
autovacuum = on
#是否记录 autovacuum 执行日志
log_autovacuum_min_duration = 0
#最大的 autovacuum worker 数, 一个库只能有一个 worker 在工作
autovacuum_max_workers = 3
#autovacuum launcher process 进程的一个轮循的总睡眠时间, 而不是针对单个库的睡眠时间
autovacuum_naptime = 1min
#autovacuum 进程进行 vacuum 操作的阈值条件一, 指修改, 删除的记录数。
autovacuum_vacuum_threshold = 50
#autovacuum 进程进行 analyze 操作的阈值条件一, 指插入, 修改, 删除的记录数。
autovacuum_analyze_threshold = 50
#autovacuum 进程进行 vacuum 操作的阈值条件二表
#表(update,delete 记录)/pg_class.reltuples> autovacuum_vacuum_scale_factor,
autovacuum_vacuum_scale_factor = 0.002
#autovacuum 进程进行 analyze 操作的阈值条件二表
#表(insert,update,delete 记录)/pg_class.reltuples> autovacuum_vacuum_scale_factor
autovacuum_analyze_scale_factor = 0.001
#autovacuum 执行超过 autovacuum_vacuum_cost_limit 值时 vacuum 进程休眠时间
autovacuum_vacuum_cost_delay = 20ms
#配置自动 VACUUM 操作中的代价限制值
autovacuum_vacuum_cost_limit = -1
#一个表的 pg_class.relFrozenxid 域能保持的最大年龄就强制 vacuum freeze
autovacuum_freeze_max_age = 200000000
vacuum_delta = 100000

#中止任何使用了超过指定毫秒数的语句, 零值 (默认) 将关闭这个参数
statement_timeout = 0

#配置表锁等待日志
log_lock_waits = on
#配置等待超过多少秒才记录日志
deadlock_timeout = 1s

```

```
#配置默认时区
datestyle = 'iso, ymd'
timezone = 'PRC'

#动态库预加载，需要初始化共享内存，如没有可以为空
#shared_preload_libraries = 'pg_stat_statements,pg_errcode_stat'
shared_preload_libraries = "
pg_stat_statements.max = 1000
pg_stat_statements.track = all

pg_errcode_stat.max_cnt = 1000
pg_errcode_stat.errmsg_size = 512
pg_errcode_stat.track_enable = true
pg_errcode_stat.save = true
pg_errcode_stat.log_min = WARNING

#配置 oracle 兼容
enable_oracle_compatible=off

#pooler 池满时时退出
enable_pooler_stuck_exit = off

#oltp 集群应用时配置
prefer_olap = 0
olap_optimizer = on

#脱敏开关
enable_data_mask = off
#行安全开关
enable_cls = off
#加密开关
enable_transparent_crypt = off
#审计开关
enable_audit = off
#细粒度审计开关
enable_fga = off

#并发计算
max_worker_processes = 1024
#总并行计算池
max_parallel_workers = 64
#单个业务最大的并行计算数
max_parallel_workers_per_gather = 0

#是否在使用扩展协议时将\0 的结束符替换成空格
enable_null_string = off
```

```
#禁用白名单功能, 2.07
enable_key_value = off

#限制使用 pg_basebackup 的速度为 1000MB/S 这个是落盘速度
base_backup_limit=1000

#配置冷热分区时间, 2.07
#manual_hot_date = '2018-09-13'

#每个进程最大打开文件数, 如果 pool 并发大的话需要增大, 默认是 1000
max_files_per_process = 65535

#是否将 not in(select xxx)转换成 hash join,注意, 如果子查询存在 null, 则返回结果是不一致的
enable_pullup_subquery = off
```

■ pg_hba.conf

#认证配置格式如下

# TYPE	DATABASE	USER	ADDRESS	METHOD
--------	----------	------	---------	--------

#本地 socket 连接配置

local	all	all		trust
-------	-----	-----	--	-------

tcp/ip 本地连接配置

host	all	all	127.0.0.1/32	trust
------	-----	-----	--------------	-------

#配置集群节点互相信任

host	replication	all	172.16.0.29/32	trust
------	-------------	-----	----------------	-------

host	all	all	172.16.0.29/32	trust
------	-----	-----	----------------	-------

host	replication	all	172.16.0.47/32	trust
------	-------------	-----	----------------	-------

host	all	all	172.16.0.47/32	trust
------	-----	-----	----------------	-------

#外面其它连接使用密码认证

host	all	all	0.0.0.0/0	md5
------	-----	-----	-----------	-----

5.4.3、服务管理

■ 启动服务

```
pg_ctl -Z coordinator -D /data/tbase/data/pgxz/cn01 start
```

■ 关闭服务

```
pg_ctl -Z coordinator -D /data/tbase/data/pgxz/cn01 stop
```

■ 快速关闭服务

```
pg_ctl -Z coordinator -D /data/tbase/data/pgxz/cn01 stop -m f
```

■ 强制关闭服务

```
pg_ctl -Z coordinator -D /data/tbase/data/pgxz/cn01 stop -m i
```

■ 参数重载

```
pg_ctl -Z coordinator -D /data/tbase/data/pgxz/cn01 reload
```

■ 显示状态

```
pg_ctl -Z coordinator -D /data/tbase/data/pgxz/cn01 status
```

```
pg_ctl: server is running (PID: 19233)
```

```
/data/tbase/install/pgxz/bin/postgres "--coordinator" "-D" "/data/tbase/data/pgxz/cn01"
```

5.5、datanode(dn) “主” 组件

5.5.1、初始化

```
initdb --locale=zh_CN.UTF-8 -U tbase -E utf8 -D /data/tbase/data/pgxz/dn01 --nodename=dn01 --nodetype=datanode  
--master_gtm_nodename gtm_1 --master_gtm_ip 172.16.0.29 --master_gtm_port 21000
```

5.5.2、运行参数配置

■ postgresql.conf 参数

#每个 dn 节点名字不同

```
pgxc_node_name = 'dn01'
```

#允许服务监听范围，*允许监听所有 IP 地址

```
listen_addresses = '0.0.0.0'
```

#监听端口号,如果与 cn 或者备机安装在一起的话，注意 port 不能相同

```
port = 23001
```

#数据库最大的并发连接数

#max_pool_size >= (max_connection*dn 节点数+cn 节点数); 按实际计算

```
max_connections = 2000
```

#需要保留 super 的操作连接数

```
superuser_reserved_connections = 20
```

#配置 dn 连接池监听端口号

```
pooler_port = 25001
```

#配置连接池大小

max_pool_size >= (max_connection*max_datanodes+max_coordinators); 按实际计算

```
max_pool_size = 65535
```

#指定不活动多少秒之后通过 TCP 向客户端发送一个 keepalive 消息

```
tcp_keepalives_idle = 60
```

#指定在多少秒之后重发一个 keepalive 消息

```

tcp_keepalives_interval = 10
#指定与客户端的服务器连接被认为死掉之前发送的次数
tcp_keepalives_count = 6

#设置数据库服务器将使用的共享内存缓冲区大小，这个值一般建议是不要超过 1/4 物理内存大小
#调整该值可以根据查询的命中率来调整
shared_buffers = 16GB

#二阶段提交最大数目，要大于 max_connections
max_prepared_transactions = 20000

#配置内部排序操作或哈希表使用最大内存量，超出这个值会写到临时磁盘文件
#这个默认值不宜过大，特别是并发连接数在高的应用，session 中视需要可以单独调整
work_mem = 4MB

#指定在维护性操作（例如 CREATE INDEX）中使用的 最大的内存量，这个值可以在 session 中配置
maintenance_work_mem = 2048MB

#配置 xlog 日志级别，其值为 minimal, replica, or logical
#日志写入量为 logical>replica>minimal，主备复制配置为 replica，逻辑复制配置成 logical
wal_level = logical

#指定来自后备服务器或流式基础备份客户端的并发连接的最大数量
#数据并发迁移时需要较大数量的 sender 进程
max_wal_senders = 64
#指定服务器可以支持的复制槽,把它的值设置为高于 max_wal_senders
max_replication_slots = 64
#指定逻辑复制工作的最大数量
max_logical_replication_workers = 64
#每个订阅的最大同步工作者数量
max_sync_workers_per_subscription = 4

#为后备服务器保留的 xlog 文件数
wal_keep_segments = 10000
#指定一个支持同步复制的后备服务器的列表，如果使用异步复制，则这里不能打开
synchronous_standby_names = ""

#控制一个事务是否需要等待 WAL 记录被写入磁盘
#当性能比完全确保事务的持久性更重要时，关闭 synchronous_commit 可以作为一个有效的代替手段。
#参数值为 off, local, remote_write 和 on,如果 synchronous_standby_names 被设置时
#这个参数也控制事务提交是否将等待事务的 WAL 记录被复制到后备服务器上。
#该参数可以在 session 中进行配置，即应用能够控制不同业务使用不同的同步方式
#当设置为 off 时，在向客户端报告成功和真正事务保证之间会有延迟
#（最大的延迟是 wal_writer_delay 的三倍）。不同于 fsync，
#将这个参数设置为 off 不会产生数据库不一致性的风险
synchronous_commit = on

```

#控制在一个检查点之后的页面的第一次修改期间将每个页面的全部内容写到 WAL 中。
 #把这个参数关闭会加快正常操作，但是在系统失败后可能导致不可恢复的数据损坏
 full_page_writes = on

#用于控制还未写入磁盘的 WAL 数据的共享内存量
 wal_buffers = 16MB

#启用归档
 archive_mode = on
 #归档执行命令，实际命令参考备份归档文档,如
 #DIR=/home/postgres/archive/'date +%F'; test ! -d \$DIR && mkdir \$DIR; test ! -f \$DIR/%f && cp %p \$DIR/%f
 #如果暂时不需要归档可以这样配置
 archive_command = 'bin/date'
 #配置多少秒后如果还有产生新的 xlog，则服务器将切换到一个新的段文件，0 表示禁用
 archive_timeout = 0

#指定备角色期间是否可以连接查询
 hot_standby = on
 #延长备机查询与归档恢复冲突时 timeout 时间，默认是 30s，值 -1 允许后备机一直等到冲突查询结束
 max_standby_archive_delay = 1800s
 #延长备机查询与流复制冲突时 timeout 时间，默认是 30s 值 -1 允许后备机一直等到冲突查询结束
 max_standby_streaming_delay = 1800s

#在每次写入脏缓冲区后，后台写入器进程休眠时间，默认为 200ms
 bgwriter_delay = 10ms
 #控制每次最大多少个缓冲区将被后台写入器写出
 bgwriter_lru_maxpages = 500
 #用于计算每次多少个缓冲区被后台写入器写出
 bgwriter_lru_multiplier = 2.0

#设置顺序扫描磁盘页面，获取一次数据的代价
 seq_page_cost = 1.0
 #设置非顺序扫描磁盘页面，获取一次数据的代价,如果是 ssd 的话，这个值可以设置为 2
 random_page_cost = 4.0

#自动 WAL 检查点之间的最长时间，以秒计，增大这个参数可以降低 xlog 的写入量
 checkpoint_timeout = 10min
 #自动 WAL 检查点之间的最大的 xlog 文件数
 #建议值 max_wal_size=shared_buffers * 0.75，min_wal_size=max_wal_size*0.25
 max_wal_size = 12GB
 min_wal_size = 3GB
 #控制 checkpoint 对 shared buffers 的写入完成时间，如两次 checkpoint 的时间差是 60 秒
 #则由于 checkpoint 而引发的 IO 操作会被平均分配在 60*0.5=30s 之内完成
 #这个参数值大则 IO 负载变得平滑，避免在做 checkpoint 操作时出现的 IO 尖峰
 #但持续时间长 IO 使用率变得过低，合理的参数值可以在日志表中收集一下 checkpoint 参数后进行调整

```

#checkpoint complete: wrote 6518 buffers (1.2%);
#0 transaction log file(s) added, 0 removed, 2 recycled;
#write=149.047 s, sync=1.485 s, total=150.582 s; sync files=70, longest=0.192 s, average=0.021 s
#理想的值为
checkpoint_completion_target = 0.93
#两次 checkpoint 发生的时间并为多少秒时，那么就向服务器日志写一个消息
checkpoint_warning = 30s

#告诉服务器，系统可能使用到的内存数
effective_cache_size = 250GB

#日志相关配置
#用户访问日志格式
log_destination = 'csvlog'
#启用用户访问日志收集器
logging_collector = on
#日志存放目录，可以是相对路径（相对了 data 目录）或绝对路径
log_directory = 'log'
#设置日志文件的权限
log_file_mode = 0600
#按我先前经常是只有 log_rotation_age 这个值有效果
log_truncate_on_rotation = 'on'
#120 分钟切换一次
log_rotation_age = 120
#64MB 切换一次
log_rotation_size = 64MB
#会话的当前执行命令保留长度，用于在字段 pg_stat_activity.query 中显示
track_activity_query_size = 4096
#配置 sql 语句执行超过多少毫秒数时，语句将被记录,值为-1 时禁用，0 时记录所有语句
log_min_duration_statement = 1000
#是否记录 checkpoint
log_checkpoints = on
#是否记录连接
log_connections = on
#是否记录断开连接
log_disconnections = on
#是否记录 autovacuum 执行日志
log_autovacuum_min_duration = 0
#是否记录所有语句的执行时间，值为 on 时将单独记录所有语句的执行时间，这里不记录语句
log_duration = off
#配置日志记录的内容
log_line_prefix = '%m %a %r %d %u %p %x'
#控制那些类型的语句被记录，有效值是 none (off)、ddl、mod 和 all（所有语句）。
#如果 log_min_duration_statement 值为 0 时，则 log_statement 什么值的效果都一样
#如果 log_min_duration_statement 值大于 0，并且 log_statement 为 ddl 则 ddl 语句全部表被记录，为两条 log
#dml 超时才被记录，为一条记录

```

```

log_statement = 'none'
#设置在服务器日志中写入的时间戳的时区
log_timezone = 'PRC'
log_filename = 'postgresql-%A-%H.log'

#打开 autovacuum,进行垃圾数据自动回收
#打开垃圾回收及系统信息统计, 注意 /etc/hosts 里面的 localhost 解释要打开
track_counts = on
autovacuum = on
#是否记录 autovacuum 执行日志
log_autovacuum_min_duration = 0
#最大的 autovacuum worker 数, 一个库只能有一个 worker 在工作
autovacuum_max_workers = 3
#autovacuum launcher process 进程的一个轮循的总睡眠时间, 而不是针对单个库的睡眠时间
autovacuum_naptime = 1min
#autovacuum 进程进行 vacuum 操作的阈值条件一, 指修改, 删除的记录数。
autovacuum_vacuum_threshold = 50
#autovacuum 进程进行 analyze 操作的阈值条件一, 指插入, 修改, 删除的记录数。
autovacuum_analyze_threshold = 50
#autovacuum 进程进行 vacuum 操作的阈值条件二表
#表(update,delete 记录)/pg_class.reltuples> autovacuum_vacuum_scale_factor,
autovacuum_vacuum_scale_factor = 0.002
#autovacuum 进程进行 analyze 操作的阈值条件二表
#表(insert,update,delete 记录)/pg_class.reltuples> autovacuum_vacuum_scale_factor
autovacuum_analyze_scale_factor = 0.001
#autovacuum 执行超过 autovacuum_vacuum_cost_limit 值时 vacuum 进程休眠时间
autovacuum_vacuum_cost_delay = 20ms
#配置自动 VACUUM 操作中的代价限制值
autovacuum_vacuum_cost_limit = -1
#一个表的 pg_class.relFrozenxid 域能保持的最大年龄就强制 vacuum freeze
autovacuum_freeze_max_age = 200000000
vacuum_delta = 100000

#中止任何使用了超过指定毫秒数的语句, 零值 (默认) 将关闭这个参数
statement_timeout = 0

#配置表锁等待日志
log_lock_waits = on
#配置等待超过多少秒才记录日志
deadlock_timeout = 1s

#配置默认时区
datestyle = 'iso, ymd'
timezone = 'PRC'

#动态库预加载, 需要初始化共享内存, 如没有可以为空

```

```
#shared_preload_libraries = 'pg_stat_statements,pg_errcode_stat'
shared_preload_libraries = "
pg_stat_statements.max = 1000
pg_stat_statements.track = all

pg_errcode_stat.max_cnt = 1000
pg_errcode_stat.errmsg_size = 512
pg_errcode_stat.track_enable = true
pg_errcode_stat.save = true
pg_errcode_stat.log_min = WARNING

#配置 oracle 兼容
enable_oracle_compatible=off

#pooler 池满时时退出
enable_pooler_stuck_exit = off

#oltp 集群应用时配置
prefer_olap = 0
olap_optimizer = false

#脱敏开关
enable_data_mask = off
#行安全开关
enable_cls = off
#加密开关
enable_transparent_crypt = off
#审计开关
enable_audit = off
#细粒度审计开关
enable_fga = off

#并发计算
max_worker_processes = 1024
#总并行计算池
max_parallel_workers = 64
#单个业务最大的并行计算数
max_parallel_workers_per_gather = 0

#是否在使用扩展协议时将\0 的结束符替换成空格
enable_null_string = off

#禁用白名单功能，2.07
enable_key_value = off

#限制使用 pg_basebackup 的速度为 1000MB/S 这个是落盘速度
```

```
base_backup_limit=1000
```

```
#配置冷热分区时间, 2.07
```

```
#manual_hot_date = '2018-09-13'
```

```
#每个进程最大打开文件数, 如果 pool 并发大的话需要增大, 默认是 1000
```

```
max_files_per_process = 65535
```

```
#是否将 not in(select xxx)转换成 hash join,注意, 如果子查询存在 null, 则返回结果是不一致的
```

```
enable_pullup_subquery = off
```

■ pg_hba.conf

```
#认证配置格式如下
```

```
# TYPE DATABASE USER ADDRESS METHOD
```

```
#本地 socket 连接配置
```

```
local all all trust
```

```
# tcp/ip 本地连接配置
```

```
host all all 127.0.0.1/32 trust
```

```
#配置集群节点互相信任
```

```
host replication all 172.16.0.29/32 trust
```

```
host all all 172.16.0.29/32 trust
```

```
host replication all 172.16.0.47/32 trust
```

```
host all all 172.16.0.47/32 trust
```

```
#外面其它连接使用密码认证
```

```
host all all 0.0.0.0/0 md5
```

5.5.3、服务管理

■ 启动服务

```
pg_ctl -Z datanode -D /data/tbase/data/pgxz/dn01 start
```

■ 关闭服务

```
pg_ctl -Z datanode -D /data/tbase/data/pgxz/dn01 stop
```

■ 快速关闭服务

```
pg_ctl -Z datanode -D /data/tbase/data/pgxz/dn01 -m f
```

■ 强制关闭服务

```
pg_ctl -Z datanode -D /data/tbase/data/pgxz/dn01 stop -m i
```

■ 参数重载

```
pg_ctl -Z datanode -D /data/tbase/data/pgxz/dn01 reload
```

■ 显示状态

```
pg_ctl -Z datanode -D /data/tbase/data/pgxz/dn01 status
```

```
pg_ctl: server is running (PID: 20354)
```

```
/data/tbase/install/pgxz/bin/postgres "--datanode" "-D" "/data/tbase/data/pgxz/dn01"
```

5.6、datanode(dn) “备” 组件

5.6.1、备节点生成

```
pg_basebackup -p 23001 -h 172.16.0.29 -U tbase -D /data/tbase/data/pgxz/dn01 -X f -P -v
```

生成备节点注意事项参考 3.2.2.5

5.6.2、运行参数配置

■ postgresql.conf 参数

参考 DN 主节点之 postgresql.conf 运行参数配置，注意备机是否要开启归档，如果不需要，请按下面参数修改

#其它注意整理

#如果备机禁用归档，则做如下配置

```
archive_mode = off
```

#如果主备混布，需要注意配置 cn 连接池监听端口号不要冲突到

```
pooler_port = 22001
```

■ pg_hba.conf 参数

请参考 dn 主的 pg_hba.conf 配置

■ recovery.conf 参数

#节点是否做为备机运行

```
standby_mode = 'on'
```

#主节点连接参数，发果是同步复制，则需要配置 application_name

```
primary_conninfo = 'host=172.16.0.29 port=23001 user=postgres password=pgsql application_name=sync'
```

#如果备机流复制追不上的话，并且有归档日志存在，则可以使用 restore_command 来同步数据

```
#restore_command = 'cp /home/postgres/archive/20[1-9][0-9]-[0-1][0-9]-[0-3][0-9]/%f %p;'
```

```
#同步到最新的时间线
recovery_target_timeline = 'latest'
```

5.6.3、服务管理

■ 启动服务

```
pg_ctl -Z datanode -D /data/pgxz/data/pgxz/dn01 start
```

■ 关闭服务

```
pg_ctl -Z datanode -D /data/pgxz/data/pgxz/dn01 stop
```

■ 快速关闭服务

```
pg_ctl -Z datanode -D /data/pgxz/data/pgxz/dn01 -m f
```

■ 强制关闭服务

```
pg_ctl -Z datanode -D /data/pgxz/data/pgxz/dn01 stop -m i
```

■ 参数重载

```
pg_ctl -Z datanode -D /data/pgxz/data/pgxz/dn01 reload
```

■ 显示状态

```
pg_ctl -Z datanode -D /data/pgxz/data/pgxz/dn01 status
```

```
pg_ctl: server is running (PID: 1156)
```

```
/pgxz/pgxz/install/pgxz/bin/postgres "--datanode" "-D" "/data/pgxz/data/pgxz/dn01"
```

5.7、配置 cn/dn 节点的拓补结构

5.7.1、cn pgxc_node 参数修改

5.7.1.1、修改自身参数

CN 节点初始化后其拓补结构如下所示

```
postgres=# select * from pgxc_node;
```

node_name	node_type	node_port	node_host	nodeis_primary	nodeis_preferred	node_id	node_cluster_name
cn01	C	5432	localhost	f	f	53994174	tbase_cluster
gtm_1	G	21000	172.16.0.29	t	f	-1500995709	tbase_cluster

(2 rows)

使用 `alter node` 命令可以修改其配置参数，如下所示

```
postgres=# alter node cn01 with (host='172.16.0.29',port=15432);
```

```
ALTER NODE
```

```
postgres=# select * from pgxc_node;
```

node_name	node_type	node_port	node_host	nodeis_primary	nodeis_preferred	node_id	node_cluster_name
gtm_1	G	21000	172.16.0.29	t	f	-1500995709	tbase_cluster
cn01	C	15432	172.16.0.29	f	f	53994174	tbase_cluster

(2 rows)

5.7.1.2、添加其它 cn 节点

```
postgres=# create node cn02 with (type=coordinator,host='172.16.0.47',port=15432,primary=false,preferred=false);
```

```
CREATE NODE
```

```
postgres=# select * from pgxc_node;
```

node_name	node_type	node_port	node_host	nodeis_primary	nodeis_preferred	node_id	node_cluster_name
gtm_1	G	21000	172.16.0.29	t	f	-1500995709	tbase_cluster
cn01	C	15432	172.16.0.29	f	f	53994174	tbase_cluster
cn02	C	15432	172.16.0.47	f	f	-613255070	tbase_cluster

(3 rows)

5.7.1.3、添加其它 dn 节点

```
postgres=# create node dn01 with (type=datanode,host='172.16.0.29',port=23001,primary=false,preferred=false);
```

```
CREATE NODE
```

```
postgres=# create node dn02 with (type=datanode,host='172.16.0.47',port=23002,primary=false,preferred=false);
```

```
CREATE NODE
```

```
postgres=# select * from pgxc_node;
```

node_name	node_type	node_port	node_host	nodeis_primary	nodeis_preferred	node_id	node_cluster_name
gtm_1	G	21000	172.16.0.29	t	f	-1500995709	tbase_cluster
cn01	C	15432	172.16.0.29	f	f	53994174	tbase_cluster
cn02	C	15432	172.16.0.47	f	f	-613255070	tbase_cluster
dn01	D	23001	172.16.0.29	f	f	-1085152094	tbase_cluster
dn02	D	23002	172.16.0.47	f	f	-506537247	tbase_cluster

(5 rows)

```
postgres=# select pgxc_pool_reload();
```

```
pgxc_pool_reload
```

```
-----
t
(1 row)
```

5.7.2、dn pgxc_node 参数修改

5.7.2.1、修改自身参数

DN 节点初始化后其拓补结构如下所示

```
postgres=# select * from pgxc_node;
```

node_name	node_type	node_port	node_host	nodeis_primary	nodeis_preferred	node_id	node_cluster_name
dn01	D	5432	localhost	f	f	-1085152094	tbase_cluster
gtm_1	G	21000	172.16.0.29	t	f	-1500995709	tbase_cluster

(2 rows)

使用 alter node 命令可以修改其配置参数，如下所示

```
postgres=# alter node dn01 with(host='172.16.0.29',port=23001);
ALTER NODE
```

```
postgres=# select * from pgxc_node;
```

node_name	node_type	node_port	node_host	nodeis_primary	nodeis_preferred	node_id	node_cluster_name
gtm_1	G	21000	172.16.0.29	t	f	-1500995709	tbase_cluster
dn01	D	23001	172.16.0.29	f	f	-1085152094	tbase_cluster

(2 rows)

```
postgres=#
```

5.7.2.2、添加其它 cn 节点

```
postgres=# create node cn01 with (type=coordinator,host='172.16.0.29',port=15432,primary=false,preferred=false);
CREATE NODE
```

```
postgres=# create node cn02 with (type=coordinator,host='172.16.0.47',port=15432,primary=false,preferred=false);
CREATE NODE
```

```
postgres=# select * from pgxc_node;
```

node_name	node_type	node_port	node_host	nodeis_primary	nodeis_preferred	node_id	node_cluster_name
gtm_1	G	21000	172.16.0.29	t	f	-1500995709	tbase_cluster
dn01	D	23001	172.16.0.29	f	f	-1085152094	tbase_cluster
cn01	C	15432	172.16.0.29	f	f	53994174	tbase_cluster
cn02	C	15432	172.16.0.47	f	f	-613255070	tbase_cluster

(4 rows)

5.7.2.3、添加其他 dn 节点

```
postgres=# create node dn02 with (type=datanode,host='172.16.0.47',port=23002,primary=false,preferred=false);
CREATE NODE
```

```
postgres=# select * from pgxc_node;
```

node_name	node_type	node_port	node_host	nodeis_primary	nodeis_preferred	node_id	node_cluster_name
gtm_1	G	21000	172.16.0.29	t	f	-1500995709	tbase_cluster
dn01	D	23001	172.16.0.29	f	f	-1085152094	tbase_cluster
cn01	C	15432	172.16.0.29	f	f	53994174	tbase_cluster
cn02	C	15432	172.16.0.47	f	f	-613255070	tbase_cluster
dn02	D	23002	172.16.0.47	f	f	-506537247	tbase_cluster

```
(5 rows)
```

```
postgres=# select pgxc_pool_reload();
```

```
pgxc_pool_reload
```

```
-----
```

```
t
```

```
(1 row)
```

```
postgres=#
```

5.7.3、让修改的拓补结构生效

```
postgres=# select pgxc_pool_reload();
```

```
pgxc_pool_reload
```

```
-----
```

```
t
```

```
(1 row)
```

注意，这些都需要在每个 coordinator，datanode 节点操作，因为 coordinator 上创建 node 的操作不会在所有的 coordinator 中同步（也不能同步），需要手工操作。到这里整个集群就可以使用了，记得还要创建 **group** 和 **shard map** 哦，具体参考第二节。

5.7.4、gtm 节点管理方法

5.7.4.1、添加 gtm 节点

```
create gtm node gtm_1 with (host='172.16.0.29',port=21000);
```

5.7.4.2、删除 gtm 节点

```
drop gtm node gtm_1;
```

5.7.4.3、修改单一 cn 或者 dn 的 gtm 配置信息

```
alter gtm node gtm_1 with (host='172.16.0.29',port=21000);
```

5.7.4.4、修改所有 cn 和 dn 的 gtm 配置信息

```
alter cluster gtm node gtm_1 with (host='172.16.0.29',port=21000);
```

5.8、密码文件.pgpass、PGPASSFILE

5.8.1、什么是密码文件

密码就是为 TBase 客户端连接认证时提供密码服务的文件，他是一个文本文件

5.8.2、为什么需要密码文件

因为在项目的实际应用中经常需要非人工的应用程序交互，这时如果有密码认证的话，就需要输入密码，但机器无法这样做，这就需要有一个有保存密码的地方让应用自动去获取

5.8.3、密码文件叫什么名字

Linux 下叫 **.pgpass** 或者环境变量 PGPASSFILE 引用的一个文件

5.8.4、密码文件存入于什么地方

- 存放于 /data/tbase/.pgpass
- 存放于 PGPASSFILE 引用文件的位置

5.8.5、密码文件内容格式

```
hostname:port:database:username:password
127.0.0.1:15432:*:tbase:123
```

5.8.6、密码文件使用实例

```
mv /data/tbase/.pgpass /data/tbase/.pgpass.bak
psql -h 127.0.0.1 -U tbase -p 15432 -d postgres
Password for user postgres:这里要录入密码
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=# \q
echo 127.0.0.1:15432:*.pgxz:123>/data/tbase/pgpass.txt
export PGPASSFILE=/data/tbase/pgpass.txt
```

```
psql -h 127.0.0.1 -U tbasep 15432 -d postgres
```

WARNING: password file "/data/tbase/pgpass.txt" has group or world access; permissions should be u=rw (0600) or less --密码文件的权限需要是 0600

Password for user postgres: 使用不上密码文件，所以还需要录入

```
psql (PostgreSQL 10 (TBase 2.01))
```

Type "help" for help.

```
postgres=# \q
```

```
[tbase@VM_0_3_centos center]$ chmod 0600 /data/tbase/pgpass.txt
```

```
[tbase@VM_0_3_centos center]$ psql -h 127.0.0.1 -U pgxz -p 15432 -d postgres
```

```
psql (PostgreSQL 10 (TBase 2.01))
```

Type "help" for help.

```
postgres=#
```

5.9、SSL 加密连接配置

TBase 支持多种认证方式，不加密，单向 SSL 和双向 SSL。双向认证 SSL 协议要求服务器和用户双方都有证书。单向认证 SSL 协议不需要客户拥有 CA 证书，由于单向认证和双向认证的区别仅在于创建连接阶段，数据的传输均为加密的数据，如果有第三方攻击，获得的只是加密的数据，第三方要获得有用的信息，就需要对加密的数据进行解密，这时候的安全就依赖于密码方案的安全。而幸运的是，目前所用的密码方案，只要通讯密钥长度足够的长，就足够的安全。这也是我们强调要求使用最少 128 位加密通讯的原因，TBase 使用的是 SHA256 加密算法。因此一般情况下客户端与 TBase 服务端的连接采取 SSL 单向认证即可，即仅在 TBase Server 端配置 SSL 证书。另外特别强调的是客户端到底采用那个认证方式是由服务器端来决定的。

5.9.1、创建服务器端私钥

```
[tbase@VM_0_3_centos cn01]$ cd /data/tbase/data/pgxz/cn01
```

```
[tbase@VM_0_3_centos cn01]$ openssl genrsa -des3 -out server.key 1024
```

Generating RSA private key, 1024 bit long modulus

```
.++++++
```

```
.....++++++
```

e is 65537 (0x10001)

Enter pass phrase for server.key:

Verifying - Enter pass phrase for server.key:

```
[tbase@VM_0_3_centos cn01]$
```

#删除在上面添加的密码

```
[tbase@VM_0_3_centos cn01]$ openssl rsa -in server.key -out server.key
```

Enter pass phrase for server.key:

writing RSA key

#私钥文件设置最小访问权限

```
[tbase@VM_0_3_centos cn01]$ chmod 0400 server.key
```

5.9.2、创建服务器端证书

```
[tbase@VM_0_3_centos cn01]$ openssl req -new -key server.key -days 3650 -out server.crt -x509 -subj
'/C=CN/ST=guangdong/L=shenzhen/O=Tencent/CN=tbase.qq.com/emailAddress=tbase@tencent.com'
```

5.9.3、创建根证书

```
[tbase@VM_0_3_centos cn01]$ cp server.crt root.crt
```

5.9.4、配置 pg_hba.conf

在 pg_hba.conf 文件中增加下面一行

```
hostssl      all             tbase      0.0.0.0/0   md5 clientcert=0
```

说明 clientcert=0 表示准许客户端单向 SSL 连接，clientcert=1 表示只准许客户端双向 SSL 连接

5.9.5、配置 postgresql.conf

在 postgresql.conf 文件中增加下面两行

```
ssl = on
ssl_ca_file = 'root.crt'
```

5.9.6、重启服务生效

```
[tbase@VM_0_3_centos cn01]$ /data/tbase/install/pgxz/bin/pg_ctl -D /data/tbase/data/pgxz/cn01/ stop -m f
waiting for server to shut down.... done
server stopped
[tbase@VM_0_3_centos cn01]$ /data/tbase/install/pgxz/bin/pg_ctl -D /data/tbase/data/pgxz/cn01/ start -Z
coordinator
```

5.9.7、客户端验证 ssl 连接是否生效

```
[tbase@VM_0_3_centos pgxz]$ psql -h 172.16.0.29 -p 15432 -d postgres -U tbase
Password for user tbase:
psql (PostgreSQL 10 (TBase 2.01))
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.
```

```
postgres=>
```

或者这样连接

```
[tbase@VM_0_3_centos .postgresql]$ psql postgresql://tbase@172.16.0.29:15432/postgres?sslmode=verify-ca
Password:
psql (PostgreSQL 10 (TBase 2.01))
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.
```

postgres=>

5.9.8、创建客户端证书

5.9.8.1、创建客户端私钥

```
[tbase@VM_0_3_centos cn01]$ openssl genrsa -des3 -out /tmp/postgresql.key 1024
Generating RSA private key, 1024 bit long modulus
...+++++
.....+++++
e is 65537 (0x10001)
Enter pass phrase for /tmp/postgresql.key:
Verifying - Enter pass phrase for /tmp/postgresql.key:

#删除在上面添加的密码

[tbase@VM_0_3_centos cn01]$ openssl rsa -in /tmp/postgresql.key -out /tmp/postgresql.key
Enter pass phrase for /tmp/postgresql.key:
writing RSA key
```

5.9.8.2、创建客户端证书

为 TBase 数据库用户(tbase)创建 SSL 证书并且使用服务器上的 root.crt 文件来签名

```
[tbase@VM_0_3_centos cn01]$ openssl req -new -key /tmp/postgresql.key -out /tmp/postgresql.csr -subj
'/C=CN/ST=guangdong/L=shenzhen/O=Tencent/CN=tbase'
[tbase@VM_0_3_centos cn01]$ openssl x509 -req -in /tmp/postgresql.csr -CA root.crt -CAkey server.key -out
/tmp/postgresql.crt -CAcreateserial
Signature ok
subject=/C=CN/ST=guangdong/L=shenzhen/O=Tencent/CN=tbase
Getting CA Private Key
```

subj 参数中的通用性数据可以修改为你的个人资料数据，但是 Common Name (/CN=) 必须是在服务器上第一次配置证书时设置的数据数据库用户名 (本文示例中是 tbase)root.crt 和 server.key 文件必须与执行命令的目录；否则就应该使用完整 的路径来指定。

5.9.8.3、修改 pg_hba.conf

```
hostssl      all             tbase      0.0.0.0/0   md5 clientcert=1
```

5.9.8.4、reload TBase 服务

```
[tbase@VM_0_3_centos cn01]$ /data/tbase/install/pgxz/bin/pg_ctl -D /data/tbase/data/pgxz/cn01/ reload
server signaled
[tbase@VM_0_3_centos cn01]$
```

5.9.8.5、部署证书到客户端

复制 postgresql.key, postgresql.crt, root.crt 到家目录下的 .postgresql 目录，目录不存在就手动建立，并且配置 postgresql.key 的访问权限为 0600。Winxp 下复制到

linux 下用，存放于 /data/tbase/.postgresql，权限 0600

windows xp 存放于 C:\Documents and Settings\启动 postgresql 服务的 os 用户名\Application Data\postgresql\ 目录下，例如操作系统用户名叫 postgres，则存放于

C:\Documents and Settings\postgres\Application Data\postgresql

windows 7/8/10 存放于 C:\Users\启动 postgresql 服务的 os 用户名\AppData\Roaming\postgresql\ 目录下，例如操作系统用户名叫 postgres，则存放于

C:\Users\postgres\AppData\Roaming\postgresql

5.9.8.6、psql 客户端证书的使用

- 证书存放于默认目录，证书名称为 postgresql.key 和 root.crt

```
[tbase@VM_0_3_centos .postgresql]$ psql postgresql://tbase@172.16.0.29:15432/postgres?sslmode=verify-ca
Password:
psql (PostgreSQL 10 (TBase 2.01))
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.
```

postgres=>

默认情况下不需要指定证书位置

- 自定义证书位置

```
[tbase@VM_0_3_centos ~]$ psql 'host=172.16.0.29 port=15432 dbname=postgres user=tbase sslmode=verify-ca
```

```
sslcert=postgresql.crt sslkey=postgresql.key sslrootcert=root.crt'
```

Password:

psql (PostgreSQL 10 (TBase 2.01))

SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)

Type "help" for help.

```
postgres=> \q
```

```
[tbase@VM_0_3_centos ~]$
```

也可以这样写

```
[tbase@VM_0_13_centos
```

```
~]$
```

```
psql
```

```
"postgresql://tbase@172.16.0.29:15432/postgres?sslmode=verify-ca&sslcert=postgresql.crt&sslkey=postgresql.ke  
y&sslrootcert=root.crt"
```

Password:

psql (PostgreSQL 10 (TBase 2.01))

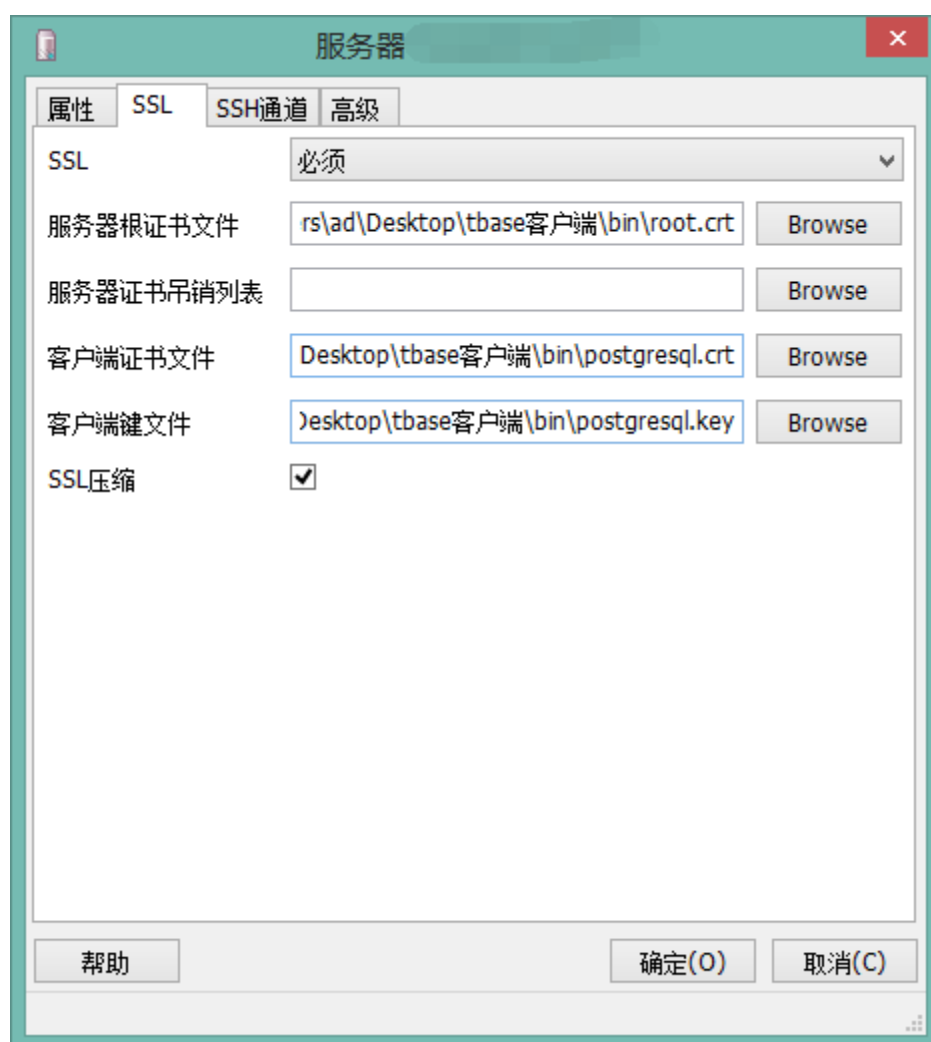
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)

Type "help" for help.

```
postgres=>
```

上面连接指定证书位于连接客户端 psql 当前目录下

5.9.8.7、pgadmin-III 客户端证书的使用



6、TBase 工具使用

6.1、pg_dump 数据导出工具

6.1.1、pg_dump 介绍

pg_dump 是用于备份 Tbase 单个数据库的工具。即使数据库正在被并发使用，它也能创建一致的备份。pg_dump 不阻塞其他用户访问数据库（读取或写入）。

备份的数据可以是脚本或归档文件格式。脚本内容包含 SQL 命令的纯文本数据，它们可以用来重建数据库到它被备份时的状态。恢复一个这样的脚本，需要使用 psql 客户端工具。脚本文件还可以被用来在其他机器上重构数据库。在经过一些修改后，还可以在其他 SQL 数据库产品上重构数据库。另一种可选

的归档文件格式必须与 `pg_restore` 配合使用来重建数据库。`pg_dump` 提供了一种灵活的归档和传输机制。`pg_dump` 可以被用来备份整个数据库，然后 `pg_restore` 可以被用来检查归档并/或选择数据库的哪些部分要被恢复。`pg_restore` 支持并行恢复并且默认是压缩的。

6.1.2、连接数据库参数介绍

`-d dbname`

`--dbname=dbname`

指定要连接到的数据库名，不指定该参数时默认是从 `PGDATABASE` 环境变量中取得（如果被设置）。

`-h host`

`--host=host`

指定要连接服务器机器的主机名或者 ip 地址。不指定该参数时默认是从 `PGHOST` 环境变量中取得（如果被设置）。

`-p port`

`--port=port`

指定要连接服务器监听的 TCP 端口，不指定该参数时默认是从 `PGPORT` 环境变量中（如果被设置），否则使用编译在程序中的默认值。

`-U username`

`--username=username`

指定要连接服务的用户名，不指定该参数时默认是从 `PGUSER` 环境变量中（如果被设置），否则使用当前操作系统用户名做为默认值。

`-w`

`--no-password`

从不发出一个口令提示。如果服务器要求口令认证并且没有其他方式提供口令（例如一个 `.pgpass` 文件），那么连接尝试将失败。这个选项对于批处理任务和脚本有用，因为在其中没有一个用户来输入口令。

`-W`

`--password`

强制 `pg_dump` 在连接到一个数据库之前提示要求一个口令。

6.1.3、pg_dump 经典使用方法

6.1.3.1、在脚本中应用

■ 生成文本格式

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -w >/data/tbase/tbase.sql
```

■ 生成归档格式

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -w -Fc >/data/tbase/tbase.tar.gz
```

加上小写 w 是告诉程序 pg_dump 在无法正确获取 tbase 用户密码时直接退出，而不是给出密码输入框

6.1.3.2、在人工操作中的应用

■ 生成文本格式

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -W >/data/tbase/tbase.sql
```

■ 生成归档格式

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -W -Fc >/data/tbase/tbase.tar.gz
```

加上大写 W 时强制弹出密码输入框

6.1.3.3、配置了环境变量后使用方法

```
export PGHOST=127.0.0.1
export PGDATABASE=postgres
export PGUSER=tbase
export PGPORT=15432
pg_dump -W > postgres.sql
```

6.1.4、pg_dump 扩展功能

下面 pg_dump 例子默认使用了环境变量 PGHOST,PGDATABASE,PGUSER,PGPORT

6.1.4.1、只备份指定模式下的对象

备份 tbase 模式下的对象

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -n tbase >/data/tbase/tbase.sql
```

备份 tbase 和 pgxz 模式下的对象，多个模式使用多组 -n

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -n tbase -n pgxz >/data/tbase/tbase.sql
```

6.1.4.2、不备份指定模式下的对象

不备份 tbase 模式下的对象

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -N tbase >/data/tbase/tbase.sql
```

不备份 tbase 和 pgxz 模式下的对象，多个模式使用多组 -N

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -N tbase -N pgxz >/data/tbase/tbase.sql
```

6.1.4.3、只备份某些数据表

备份数据表 tbase.t

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -t 'tbase.t' >/data/tbase/tbase.sql
```

备份数据表 tbase.t, pgxz.t

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -t 'tbase.t' -t 'pgxz.t1' >/data/tbase/tbase.sql
```

备份模式 tbase 下所有 t 开头的数据表

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -t 'tbase.t*' >/data/tbase/tbase.sql
```

6.1.4.4、不备份某些数据表

不备份数据表 tbase.t

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -T 'tbase.t' >/data/tbase/tbase.sql
```

不备份数据表 tbase.t, pgxz.t1

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -T 'tbase.t' -T 'pgxz.t1' >/data/tbase/tbase.sql
```

不备份模式 tbase 下所有 t 开头的数据表

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -T 'tbase.t*' >/data/tbase/tbase.sql
```

6.1.4.5、只备份出对象定义

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -s >/data/tbase/tbase.sql
```

6.1.4.6、只备份出数据

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 -a >/data/tbase/tbase.sql
```

6.1.4.7、备份出的数据格式为 insert values(xxx)

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 --inserts >/data/tbase/tbase.sql
```

6.1.4.8、备份出的数据格式为 insert(xxx) values(xxx)

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 --column-inserts >/data/tbase/tbase.sql
```

6.1.4.9、备份带 group 信息

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 --include-nodes >/data/tbase/tbase.sql
```

6.1.4.10、不备份带非日志表数据

```
pg_dump -h 127.0.0.1 -U tbase -d postgres -p 15432 --no-unlogged-table-data >/data/tbase/tbase.sql
```

6.2、pg_dumpall 数据导出工具

6.2.1、pg_dumpall 介绍

pg_dumpall 是用于备份 Tbase 所有数据库的工具，所以使用 pg_dumpall 时不能指定数据库。即使数据库正在被并发使用，它也能创建一致的备份。pg_dumpall 不阻塞其他用户访问数据库（读取或写入）。与 pg_dump 不同的地方是 pg_dumpall 可以用于备份 TBase 全局对象，如用户和表空间定义。pg_dumpall 备份时无法指定备份数据的格式，即无法使用 -Fc 这样的参数。

6.2.2、连接数据库参数介绍

-h host

--host=host

指定要连接服务器机器的主机名或者 ip 地址。不指定该参数时默认是从 PGHOST 环境变量中取得（如果被设置）。

-p port

--port=port

指定要连接服务器监听的 TCP 端口，不指定该参数时默认是从 PGPORT 环境变量中（如果被设置），否则使用编译在程序中的默认值。

-U username

--username=username

指定要连接服务的用户名，不指定该参数时默认是从 PGUSER 环境变量中（如果被设置），否则使用当前操作系统用户名做为默认值。

-w

--no-password

从不发出一个口令提示。如果服务器要求口令认证并且没有其他方式提供口令（例如一个 .pgpass 文件），那么连接尝试将失败。这个选项对于批处理任务和脚本有用，因为在其中没有一个用户来输入口令。

-W

--password

强制 pg_dump 在连接到一个数据库之前提示要求一个口令。

-l dbname

--database=dbname

指定要连接到哪个数据库备份全局对象以及发现要备份哪些其他数据库。如果没有指定，将会使用 postgres

数据库，如果 postgres 不存在，就使用 template1。

6.2.3、pg_dumpall 经典使用方法

6.2.3.1、在脚本中应用

```
pg_dumpall -h 127.0.0.1 -U tbase -p 15432 -w>/data/tbase/tbase.sql
```

加上小写 w 是告诉程序 pg_dumpall 在无法正确获取 pgxz 用户密码时直接退出，而不是给出密码输入框

6.2.3.2、在人工操作中的应用

```
pg_dumpall -h 127.0.0.1 -U tbase -p 15432 -W>/data/tbase/tbase.sql
```

加上大写 W 时强制弹出密码输入框

6.2.3.3、配置了环境变量后使用方法

```
export PGHOST=127.0.0.1
export PGUSER=tbase
export PGPORT=15432
pg_dumpall -W > postgres.sql
```

6.2.4、pg_dumpall 扩展功能

6.2.4.1、只备份全局对象

```
pg_dumpall -h 127.0.0.1 -U tbase -p 15432 -g>/data/tbase/tbase.sql
```

6.2.4.2、只备份用户角色

```
pg_dumpall -h 127.0.0.1 -U tbase -p 15432 -r>/data/tbase/tbase.sql
```

6.2.4.3、只备份表空间

```
pg_dumpall -h 127.0.0.1 -U tbase -p 15432 -t>/data/tbase/tbase.sql
```

6.2.4.4、只备份对象定义

```
pg_dumpall -h 127.0.0.1 -U tbase -p 15432 -s>/data/tbase/tbase.sql
```

6.2.4.5、只备份出数据

```
pg_dumpall -h 127.0.0.1 -U tbase -p 15432 -a >/data/tbase/tbase.sql
```

6.2.4.6、备份出的数据格式为 insert values(xxx)

```
pg_dumpall -h 127.0.0.1 -U tbase -p 15432 --inserts >/data/tbase/tbase.sql
```

6.2.4.7、备份出的数据格式为 insert(xxx) values(xxx)

```
pg_dumpall -h 127.0.0.1 -U tbase -p 15432 --column-inserts >/data/tbase/tbase.sql
```

6.2.4.8、备份带 group 信息

```
pg_dumpall -h 127.0.0.1 -U tbase -p 15432 --include-nodes >/data/tbase/tbase.sql
```

6.2.4.9、不备份带非日志表数据

```
pg_dumpall -h 127.0.0.1 -U tbase -p 15432 --no-unlogged-table-data >/data/tbase/tbase.sql
```

6.3、pg_restore 数据恢复工具

6.3.1、pg_restore 介绍

pg_restore 是一个用来从 pg_dump 创建的非文本格式归档恢复 PostgreSQL 数据库的工具。这些归档文件还允许 pg_restore 选择恢复哪些内容。pg_restore 可以在两种模式下操作。如果指定了一个数据库名称，pg_restore 会连接那个数据库并且把归档内容直接恢复到该数据库中。否则，会创建一个脚本，其中包含着重建该数据库所必要的 SQL 命令，它会被写入到一个文件或者标准输出。这个脚本输出等效于 pg_dump 的纯文本输出格式。因此，一些控制输出的选项与 pg_dump 的选项类似。

6.3.2、连接参数介绍

-d dbname

--dbname=dbname

指定要连接到的数据库名。

-h host

--host=host

指定要连接服务器机器的主机名或者 ip 地址。不指定该参数时默认是从 PGHOST 环境变量中取得（如果被设置）。

-p port

`--port=port`

指定要连接服务器监听的 TCP 端口，不指定该参数时默认是从 PGPORT 环境变量中（如果被设置），否则使用编译在程序中的默认值。

`-U username`

`--username=username`

指定要连接服务的用户名，不指定该参数时默认是从 PGUSER 环境变量中（如果被设置），否则使用当前操作系统用户名做为默认值。

`-w`

`--no-password`

从不发出一个口令提示。如果服务器要求口令认证并且没有其他方式提供口令（例如一个 .pgpass 文件），那么连接尝试将失败。这个选项对于批处理任务和脚本有用，因为在其中没有一个用户来输入口令。

`-W`

`--password`

强制 pg_restore 在连接到一个数据库之前提示要求一个口令。

6.3.3、pg_restore 经典使用方法

6.3.3.1、在脚本中应用

```
pg_restore -h 127.0.0.1 -U tbase -p 15432 -d tbase -w tbase.dump
```

加上小写 w 是告诉程序 pg_restore 在无法正确获取 pgxz 用户密码时直接退出，而不是给出密码输入框。上面语句实现的功能是把 tbase.dump 备份档恢复进数据库 tbase 中。

6.3.3.2、在人工操作中的应用

```
pg_restore -h 127.0.0.1 -U tbase -p 15432 -d tbase -W tbase.dump
```

加上大写 W 时强制弹出密码输入框

6.3.3.3、配置了环境变量后使用方法

```
export PGHOST=127.0.0.1
```

```
export PGUSER=tbase
```

```
export PGPORT=15432
```

```
pg_restore -d tbase tbase.dump
```

6.3.4、pg_restore 扩展功能

6.3.4.1、输出等效于 pg_dump 的纯文本格式

```
pg_restore tbase.dump
```

命令中不要使用 -d dbname 参数，则 pg_restore 输出等效于 pg_dump 的纯文本输出格式。

6.3.4.2、输出文本重定向于某个文件中

```
pg_restore tbase.dump -f tbase.sql
```

或者得

```
pg_restore tbase.dump > tbase.sql
```

将输出的文本重定向于 tbase.sql 文件中。

6.3.4.3、恢复时先创建数据库

```
pg_dump -h 127.0.0.1 -p 15432 -U tbase -d tbase -Fc > tbase.dump
```

```
dropdb -h 127.0.0.1 -p 15432 -U tbase tbase
```

```
pg_restore -h 127.0.0.1 -U tbase -p 15432 -C -d postgres tbase.dump
```

在使用这个选项时，-d 提到的数据库只被用于发出初始的 DROP DATABASE 和 CREATE DATABASE 命令。所有要恢复到该数据库名中的数据都出现在归档中

6.3.4.4、只恢复 ddl

```
pg_restore -h 127.0.0.1 -U tbase -p 15432 -d tbase -s tbase.dump
```

6.3.4.5、只恢复数据

```
pg_restore -h 127.0.0.1 -U tbase -p 15432 -d tbase -a tbase.dump
```

6.3.4.6、在重新创建数据库对象之前清除

```
pg_restore -h 127.0.0.1 -U tbase -p 15432 -d tbase -c tbase.dump
```

6.3.4.7、只恢复某些模式

```
pg_restore -h 127.0.0.1 -U tbase -p 15432 -n 'public' -d tbase tbase.dump
```

6.3.4.8、只恢复某些表

#恢复所有模式下的 t_txt 表

```
pg_restore -h 127.0.0.1 -U tbase -p 15432 -t 't_txt' -d tbase tbase.dump
```

#恢复所有模式下的 t_txt 表和 t 表

```
pg_restore -h 127.0.0.1 -U tbase -p 15432 -t 't_txt' -t 't' tbase.dump
```

#恢复 public 模式下的 t_txt 表

```
pg_restore -h 127.0.0.1 -U tbase -p 15432 -n 'public' -t 't_txt' -d tbase tbase.dump
```

6.3.4.9、列出归档的内容

```
pg_restore -h 127.0.0.1 -U tbase -p 15432 -d tbase -l tbase.dump
```

6.4、psql 客户端工具使用

6.4.1、psql 介绍

psql 是一个基于终端的 TBase 客户端工具，类似 Oracle 中的命令行工具 sqlplus，但比 sqlplus 强太多了。它让你能交互式地键入查询，把它们发送给 TBase，并且查看查询结果。或者，输入可以来自于一个文件或者命令行参数。此外，psql 还提供一些元命令和多种类似 shell 的特性来为编写脚本和自动化多种任务提供便利。psql 还支持命令补全，历史命令回找。

6.4.2、连接数据库参数介绍

-d dbname

--dbname=dbname

指定要连接到的数据库名，不指定该参数时默认是从 PGDATABASE 环境变量中取得（如果被设置）。

-h host

--host=host

指定要连接服务器机器的主机名或者 ip 地址。不指定该参数时默认是从 PGHOST 环境变量中取得（如果被设置）。

-p port

--port=port

指定要连接服务器监听的 TCP 端口，不指定该参数时默认是从 PGPORT 环境变量中（如果被设置），否则使用编译在程序中的默认值。

-U username

--username=username

指定要连接服务的用户名，不指定该参数时默认是从 PGUSER 环境变量中（如果被设置），否则使用当前

操作系统用户名做为默认值。

-w

--no-password

从不发出一个口令提示。如果服务器要求口令认证并且没有其他方式提供口令（例如一个.pgpass 文件），那么连接尝试将失败。这个选项对于批处理任务和脚本有用，因为在其中没有一个用户来输入口令。

-W

--password

强制 pg_dump 在连接到一个数据库之前提示要求一个口令。

6.4.3、psql 功能介绍

6.4.3.1、连接到一个数据库

■ 使用参数连接

```
[pgxz@VM_0_3_centos root]$ psql -h 172.16.0.29 -p 15432 -U tbase -d postgres
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=#
```

■ 使用 conninfo 字符串或者一个 URI

```
[pgxz@VM_0_3_centos root]$ psql postgresql://pgxz@172.16.0.29:15432/postgres
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=#
```

■ 配置证书连接

```
[pgxz@VM_0_3_centos ~]$ psql 'host=172.16.0.29 port=15432 dbname=postgres user=tbase sslmode=verify-ca
sslcert=postgresql.crt sslkey=postgresql.key sslrootcert=root.crt'
Password:
psql (PostgreSQL 10 (TBase 2.01))
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=> \q
```

■ 配置环境变量后快捷连接

```
[pgxz@VM_0_3_centos root]$ PGUSER=tbase
```

```
[pgxz@VM_0_3_centos root]$ PGHOST=127.0.0.1
[pgxz@VM_0_3_centos root]$ PGDATABASE=postgres
[pgxz@VM_0_3_centos root]$ PGPORT=15432
[pgxz@VM_0_3_centos root]$ export PGHOST PGUSER PGDATABASE PGPORT
```

```
[pgxz@VM_0_3_centos root]$ psql
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=#
```

变些环境变量可以配置在

```
[pgxz@VM_0_3_centos root]$ cat ~/.bashrc
# .bashrc
PGUSER=tbase
PGHOST=127.0.0.1
PGDATABASE=postgres
PGPORT=15432
export PGHOST PGUSER PGDATABASE PGPORT
```

6.4.3.2、建立一个新连接

- 连接到另外一个库（也可以是当前库）

```
postgres=# select pg_backend_pid();
 pg_backend_pid
-----
          2408
(1 row)
```

```
postgres=# \c
You are now connected to database "postgres" as user "pgxz".
postgres=# select pg_backend_pid();
 pg_backend_pid
-----
          2426
(1 row)
```

```
postgres=# \c template1
You are now connected to database "template1" as user "pgxz".
template1=#
```

- 连接到外一台服务

```
postgres=# \c postgres pgxz 172.16.0.47 15432
```

You are now connected to database "postgres" as user "pgxz" on host "172.16.0.47" at port "15432".

6.4.3.3、显示和设置该连接当前运行参数

■ 显示当前连接的运行参数

```
postgres=# SELECT CURRENT_USER;
current_user
-----
pgxz
(1 row)
```

```
postgres=# show search_path ;
search_path
-----
"user",public
(1 row)
```

```
postgres=# show work_mem ;
work_mem
-----
4MB
(1 row)
```

■ 设置当前连接的运行参数

```
postgres=# set search_path = "user",public,pg_catalog;
SET
postgres=# set work_mem = '8MB';
SET
```

■ 打开和关闭显示每个 sql 语句执行的时间

```
postgres=# \timing on
Timing is on.
postgres=# select count(1) from tbase;
count
-----
10000
(1 row)
```

Time: 5.139 ms

```
postgres=# \timing off
Timing is off.
postgres=# select count(1) from tbase;
count
```

```
-----
10000
(1 row)
```

■ 打开和关闭显示每个快捷操作符实际运行的 sql 语句

```
postgres=# \set ECHO_HIDDEN on
postgres=# \dt
***** QUERY *****
SELECT n.nspname as "Schema",
       c.relname as "Name",
       CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'm' THEN 'materialized view' WHEN
'i' THEN 'index' WHEN 'S' THEN 'sequence' WHEN 's' THEN 'special' WHEN 'f' THEN 'foreign table' END as
"Type",
       pg_catalog.pg_get_userbyid(c.relowner) as "Owner"
FROM pg_catalog.pg_class c
      LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('r',"
       AND n.nspname <> 'pg_catalog'
       AND n.nspname <> 'information_schema'
       AND n.nspname !~ '^pg_toast'
       AND pg_catalog.pg_table_is_visible(c.oid)
ORDER BY 1,2;
*****
```

```
          List of relations
Schema |      Name      | Type | Owner
-----+-----+-----+-----
public | t_time_range | table | pgxz
public | tbase         | table | pgxz
(2 rows)
```

```
postgres=# \set ECHO_HIDDEN off
postgres=# \dt
          List of relations
Schema |      Name      | Type | Owner
-----+-----+-----+-----
public | t_time_range | table | pgxz
public | tbase         | table | pgxz
(2 rows)
```

■ 配置输出结果为 HTML 格式

```
postgres=# \pset format html
Output format is html.
postgres=# \d tbase
```

```
<table border="1">
  <caption>Table "public.tbases"</caption>
  <tr>
    <th align="center">Column</th>
    <th align="center">Type</th>
    <th align="center">Modifiers</th>
  </tr>
  <tr valign="top">
    <td align="left">id</td>
    <td align="left">integer</td>
    <td align="left">&nbsp;</td>
  </tr>
  <tr valign="top">
    <td align="left">mc</td>
    <td align="left">text</td>
    <td align="left">&nbsp;</td>
  </tr>
</table>
```

恢复为对齐模式

```
postgres=# \set format aligned
```

Output format is aligned.

```
postgres=# \d tbases
```

Table "public.tbases"

Column	Type	Modifiers
id	integer	
mc	text	

■ 配置行列显示格式

```
postgres=# \x on
```

Expanded display is on.

```
postgres=# select * from tbases where id=1;
```

```
-[ RECORD 1 ]
```

```
id | 1
```

```
mc | 1
```

```
-[ RECORD 2 ]
```

```
id | 1
```

```
mc | 2
```

```
-[ RECORD 3 ]
```

```
id | 1
```

```
mc | 2
```

```
postgres=# \x off
```

Expanded display is off.

```
postgres=# select * from tbase where id=1;
 id | mc
----+----
  1 | 1
  1 | 2
  1 | 2
(3 rows)
```

■ 显示和配置客户端编码

```
postgres=# \encoding
UTF8
```

配置客户端编码为 SQL_ASCII

```
postgres=# \encoding sql_ascii
postgres=# \encoding
SQL_ASCII
```

6.4.3.4、退出连接

```
postgres=# \q
```

6.4.3.5、psql 执行一个 sql 命令

■ 显示标题

```
[pgxz@VM_0_3_centos root]$ psql -h 172.16.0.29 -p 15432 -U tbase -d postgres -c "select count(1) from pg_class"
 count
-----
    317
(1 row)
```

■ 不显示标题

```
[pgxz@VM_0_3_centos root]$ psql -h 172.16.0.29 -p 15432 -U tbase -d postgres -t -c "select count(1) from pg_class"
317
```

6.4.3.6、psql 执行一个 sql 文件中所有命令

■ 在外部执行

```
[pgxz@VM_0_3_centos ~]$ cat /data/tbase/tbase.sql
set search_path = public;
```

```
insert into tbase values(1,2);
select count(1) from tbase;
```

```
[pgxz@VM_0_3_centos ~]$ psql -h 172.16.0.29 -p 15432 -U tbase -d postgres -f /data/tbase/tbase.sql
SET
INSERT 0 1
count
-----
10001
(1 row)
```

■ 在内部执行

```
[pgxz@VM_0_3_centos ~]$ psql -h 172.16.0.29 -p 15432 -U tbase -d postgres
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=# \i /data/tbase/tbase.sql
SET
INSERT 0 1
count
-----
10002
(1 row)
```

6.4.3.7、调用编辑器编写 sql 脚本

```
postgres=# \e
"/tmp/psql.edit.5532.sql" 2L, 35C written
```

编辑完成后保证退出执行

6.4.3.8、调用外部命令

```
postgres=# \! cat /data/tbase/tbase.sql
set search_path = public;
insert into tbase values(1,2);
select count(1) from tbase;
postgres=# \! ls
backup data install java pgbench pgxz pgxzconf shell tbase.sql
postgres=# \! ls -l
total 8
drwxrwxr-x 3 pgxz pgxz 20 Dec 20 09:53 backup
drwxrwxr-x 3 pgxz pgxz 18 Dec 10 17:50 data
drwxr-xr-x 8 pgxz pgxz 92 Mar 6 09:25 install
```

```
drwxrwxr-x 2 pgxz pgxz 204 Dec 29 16:00 java
drwxrwxr-x 2 pgxz pgxz 119 Mar 12 17:15 pgbench
drwxrwxr-x 6 pgxz pgxz 75 Jan 28 10:53 pgxz
drwxrwxr-x 2 pgxz pgxz 25 Feb 27 15:33 pgxzconf
drwxrwxr-x 3 pgxz pgxz 4096 Mar 16 19:33 shell
-rw-rw-r-- 1 pgxz pgxz 85 Mar 28 10:17 tbase.sql
postgres=#
```

6.4.3.9、将执行的结果保存到文件

```
postgres=# \o /data/tbase/log.txt
postgres=# select * from tbase where id=1;
postgres=# \! cat /data/tbase/log.txt
id | mc
----+----
 1 | 1
 1 | 2
 1 | 2
(3 rows)
```

```
postgres=# \o
postgres=# select * from tbase where id=1;
id | mc
----+----
 1 | 1
 1 | 2
 1 | 2
(3 rows)
```

6.4.3.10、改变当前的工作目录

```
postgres=# \! ls
backup data install java log.txt pgbench pgxz pgxzconf shell tbase.sql
postgres=# \cd /data
postgres=# \! ls
package pgsql pgxz pgxzConfdb
postgres=#
```

6.4.3.11、插件管理

- 查看当前库加载了那些插件

```
postgres=# \dx
```

List of installed extensions			
Name	Version	Schema	Description


```
template for new databases
```

```
| | | | | pgxz=CTc/pgxz | |
|
(3 rows)
```

■ 创建一个新库

```
postgres=# create database mydb;
CREATE DATABASE
postgres=#
```

6.4.3.13、模式相关操作

■ \dn 显示当前库所有模式

```
postgres=# \dn
List of schemas
  Name | Owner
-----+-----
 pgxc  | pgxz
 public | pgxz
(2 rows)
```

■ \dn+显示当前库所有模式（包含注释）

```
postgres=# \dn+
List of schemas
  Name | Owner | Access privileges | Description
-----+-----+-----+-----
 pgxc  | pgxz  |                    |
 public | pgxz  | pgxz=UC/pgxz      +| standard public schema
        |        | =UC/pgxz          |
(2 rows)
```

■ 创建一个新模式

```
postgres=# create schema mysche;
CREATE SCHEMA
```

6.4.3.14、用户相关操作

■ \du 显示当前集群中所有数据库用户

```
postgres=# \du
List of roles
```

Role name	Attributes	Member of
pgxz	Superuser, Create role, Create DB, Replication	{}
tbase		{}

- \dn+显示当前集群中所有数据库用户（包含注释）

```
postgres=# \du+
```

List of roles			
Role name	Attributes	Member of	Description
pgxz	Superuser, Create role, Create DB, Replication	{}	
tbase		{}	

- 创建一个新的用户

```
postgres=# create role pgxc with login ;
CREATE ROLE
```

- 配置用户密码

```
postgres=# \password pgxc
Enter new password:
Enter it again:
```

6.4.3.15、表相关操作

- 建立数据表

```
postgres=# create table tbase(id int,mc text) distribute by shard(id);
CREATE TABLE
```

- \d 查看表结构，包括使用的触发器

```
postgres=# \d tbase
        Table "public.tbase"
  Column |  Type  | Modifiers
-----+-----+-----
   id    | integer |
   mc    | text    |
```

- \d+查看表结构（包含注释），表类型，分布节点

```
postgres=# \d+ tbase
        Table "public.tbase"
  Column |  Type  | Modifiers | Storage | Stats target | Description
```

```
-----+-----+-----+-----+-----+-----+
id      | integer |          | plain   |          |
mc      | text    |          | extended|          |
```

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

■ \dt 查看表列表

```
postgres=# \dt
```

List of relations

```
Schema |      Name      | Type | Owner
-----+-----+-----+-----
public | t_time_range | table | pgxz
public | tbase         | table | pgxz
(2 rows)
```

■ \dt+ 查看表列表详细信息，包含表大小和注释

这里连接的节点如果是 cn 的话，表大小为所有 dn 节点大小之和，否则为只是该节点的表大小

```
postgres=# \dt+
```

List of relations

```
Schema |      Name      | Type | Owner | Size  | Description
-----+-----+-----+-----+-----+-----
public | t_time_range | table | pgxz  | 0 bytes | 这是一个日期分区表
public | tbase         | table | pgxz  | 576 kB  |
(2 rows)
```

■ \dt+ 显示某个模式下的所有表

```
postgres=# \dt+ pgxc.*
```

List of relations

```
Schema |      Name      | Type | Owner | Size  | Description
-----+-----+-----+-----+-----+-----
pgxc    | order_main | table | pgxz  | 0 bytes |
(1 row)
```

■ \dt+ 表名 显示某个表的详细信息

```
postgres=# \dt+ tbase
```

List of relations

```
Schema | Name  | Type | Owner | Size  | Description
-----+-----+-----+-----+-----+-----
public | tbase | table | pgxz  | 576 kB |
(1 row)
```

■ \dt+通配符列出适配的表

```
postgres=# \dt+ t*
```

List of relations

Schema	Name	Type	Owner	Size	Description
public	t_time_range	table	pgxz	0 bytes	这是一个日期分区表
public	tbase	table	pgxz	576 kB	

(2 rows)

```
postgres=# \dt+ t_*
```

List of relations

Schema	Name	Type	Owner	Size	Description
public	t_time_range	table	pgxz	0 bytes	这是一个日期分区表

(1 row)

6.4.3.16、视图相关操作

■ 建立视图

```
postgres=# create or replace view tbase_view as select * from tbase;
CREATE VIEW
```

■ \d 查视图结构

```
postgres=# \d tbase
```

Table "public.tbase"

Column	Type	Modifiers
id	integer	
mc	text	

■ \d+查看视图结构（包含注释），包含创建视图的 sql 语句

```
postgres=# \d+ tbase_view
```

View "public.tbase_view"

Column	Type	Modifiers	Storage	Description
id	integer		plain	
mc	text		extended	

View definition:

```
SELECT tbase.id,
       tbase.mc
FROM tbase;
```

■ \dv 查看视图列表

```
postgres=# \dv
                List of relations
 Schema |          Name          | Type | Owner |
-----+-----+-----+-----
 pgxc   | t_time_range_view | view | pgxz  |
 public | tbase_view         | view | pgxz  |
(2 rows)
```

■ \dv+查看视图列表详细信息（包含注释）

```
postgres=# \dv+
                List of relations
 Schema |          Name          | Type | Owner | Size  | Description
-----+-----+-----+-----+-----+-----
 pgxc   | t_time_range_view | view | pgxz  | 0 bytes |
 public | tbase_view         | view | pgxz  | 0 bytes | 我的视图
(2 rows)
```

■ \dv+显示某个模式下的所有视图

```
postgres=# \dv+ pgxc.*
                List of relations
 Schema |          Name          | Type | Owner | Size  | Description
-----+-----+-----+-----+-----+-----
 pgxc   | t_time_range_view | view | pgxz  | 0 bytes |
(1 row)
```

■ \dv+视图名 显示某个视图的详细信息

```
postgres=# \dv+ tbase_view
                List of relations
 Schema |          Name          | Type | Owner | Size  | Description
-----+-----+-----+-----+-----+-----
 public | tbase_view         | view | pgxz  | 0 bytes | 我的视图
(1 row)
```

■ \dv+通配符列出适配的视图

```
postgres=# \dv+ t*
                List of relations
 Schema |          Name          | Type | Owner | Size  | Description
-----+-----+-----+-----+-----+-----
 pgxc   | t_time_range_view | view | pgxz  | 0 bytes |
```

```
public | tbase_view          | view | pgxz  | 0 bytes | 我的视图
(2 rows)
```

```
postgres=# \dv+ tb*
```

List of relations

```
Schema |      Name      | Type | Owner | Size  | Description
-----+-----+-----+-----+-----+-----
public | tbase_view     | view | pgxz  | 0 bytes | 我的视图
(1 row)
```

6.4.3.17、物化视图相关操作

■ 建立物化视图

```
postgres=# create MATERIALIZED VIEW tbase_count as select count(1) as num from tbase;
SELECT 1
postgres=# select * from tbase_count;
   num
-----
 10000
(1 row)
```

■ \d 查物化视图结构

```
postgres=# \d tbase_count
Materialized view "public.tbase_count"
 Column | Type  | Modifiers
-----+-----+-----
 num    | bigint |
```

■ \d+查看物化视图结构（包含注释），包含创建物化视图的 sql 语句

```
postgres=# \d+ tbase_count
Materialized view "public.tbase_count"
 Column | Type  | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
 num    | bigint |           | plain   |              |
View definition:
SELECT count(1) AS num
FROM tbase;
```

■ \dm 查看视图列表

```
postgres=# \dm
List of relations
Schema |      Name      | Type | Owner
```

```

-----+-----+-----+-----
pgxc   | tbase_sum   | materialized view | pgxz
public | tbase_count | materialized view | pgxz
(2 rows)

```

■ \dm+查看物化视图列表详细信息（包含注释），占用空间大小

```
postgres=# \dm+
```

```

                        List of relations
 Schema |      Name      |      Type      | Owner |      Size      | Description
-----+-----+-----+-----+-----+-----
pgxc    | tbase_sum      | materialized view | pgxz  | 8192 bytes |
public  | tbase_count    | materialized view | pgxz  | 8192 bytes | tbase 总记录数
(2 rows)

```

■ \dm+显示某个模式下的所有物化视图

```
postgres=# \dm+ pgxc.*
```

```

                        List of relations
 Schema |      Name      |      Type      | Owner |      Size      | Description
-----+-----+-----+-----+-----+-----
pgxc    | tbase_sum      | materialized view | pgxz  | 8192 bytes |
(1 row)

```

■ \dm+视图名 显示某个物化视图的详细信息

```
postgres=# \dm+ tbase_count
```

```

                        List of relations
 Schema |      Name      |      Type      | Owner |      Size      | Description
-----+-----+-----+-----+-----+-----
public  | tbase_count    | materialized view | pgxz  | 8192 bytes | tbase 总记录数
(1 row)

```

■ \dm+通配符列出适配的物化视图

```
postgres=# \dm t*
```

```

                        List of relations
 Schema |      Name      |      Type      | Owner
-----+-----+-----+-----
pgxc    | tbase_sum      | materialized view | pgxz
public  | tbase_count    | materialized view | pgxz
(2 rows)

```

```
postgres=# \dm tbase_c*
```

```

                        List of relations
 Schema |      Name      |      Type      | Owner
-----+-----+-----+-----

```

```
-----+-----+-----+-----
public | tbase_count | materialized view | pgxz
(1 row)
```

6.4.3.18、序列相关操作

■ 建立序列

```
postgres=# create sequence tbase_seq;
CREATE SEQUENCE
postgres=# create sequence pgxc.pgxz_seq;
CREATE SEQUENCE
```

■ \d 查看序列定义和使用情况

```
postgres=# \d tbase_seq
          Sequence "public.tbase_seq"
  Column | Type   | Value
-----+-----+-----
last_value | bigint | 1
log_cnt   | bigint | 0
is_called | boolean | f
```

■ \ds 查看序列列表

```
postgres=# \ds
          List of relations
 Schema | Name      | Type   | Owner
-----+-----+-----+-----
 pgxc   | pgxz_seq  | sequence | pgxz
 public | tbase_seq | sequence | pgxz
(2 rows)
```

■ \ds+ 查看序列列表详细信息（包含注释），占用空间大小

```
postgres=# \ds+
          List of relations
 Schema | Name      | Type   | Owner | Size      | Description
-----+-----+-----+-----+-----+-----
 pgxc   | pgxz_seq  | sequence | pgxz  | 8192 bytes |
 public | tbase_seq | sequence | pgxz  | 8192 bytes | tbase 序列
(2 rows)
```

■ \ds+ 显示某个模式下的所有序列

```
postgres=# \ds+ pgxc.*
```

```

                        List of relations
 Schema |   Name   | Type   | Owner |   Size   | Description
-----+-----+-----+-----+-----+-----
 pgxc   | pgxz_seq | sequence | pgxz  | 8192 bytes |
(1 row)

```

■ \ds+序列名 显示某个序列的详细信息

```
postgres=# \ds+ tbase_seq
```

```

                        List of relations
 Schema |   Name   | Type   | Owner |   Size   | Description
-----+-----+-----+-----+-----+-----
 public | tbase_seq | sequence | pgxz  | 8192 bytes | tbase 序列
(1 row)

```

■ \ds+通配符列出适配的序列

```
postgres=# \ds *_seq
```

```

                        List of relations
 Schema |   Name   | Type   | Owner |
-----+-----+-----+-----+
 pgxc   | pgxz_seq | sequence | pgxz
 public | tbase_seq | sequence | pgxz
(2 rows)

```

```
postgres=# \ds t*_seq
```

```

                        List of relations
 Schema |   Name   | Type   | Owner |
-----+-----+-----+-----+
 public | tbase_seq | sequence | pgxz
(1 row)

```

6.4.3.19、索引相关操作

■ 建立索引

```

postgres=# create unique index tbase_id_idx on tbase(id);
CREATE INDEX
postgres=# create index tbase_mc_idx on tbase(mc);
CREATE INDEX
postgres=#

```

■ \di 查看索引列表

```
postgres=# \di
```

List of relations

Schema	Name	Type	Owner	Table
public	tbase_id_uidx	index	pgxz	tbase
public	tbase_mc_idx	index	pgxz	tbase

(2 rows)

■ \di+查看索引列表详细信息（包含注释），占用空间大小--只能在 dn 上面查看索引大小

```
postgres=# \di+
```

List of relations

Schema	Name	Type	Owner	Table	Size	Description
public	tbase_id_uidx	index	pgxz	tbase	8192 bytes	tbase 唯一索引
public	tbase_mc_idx	index	pgxz	tbase	8192 bytes	

(2 rows)

■ \di+显示某个模式下的所有索引--只能在 dn 上面查看索引大小

```
postgres=# \di+ public.*
```

List of relations

Schema	Name	Type	Owner	Table	Size	Description
public	tbase_id_uidx	index	pgxz	tbase	8192 bytes	tbase 唯一索引
public	tbase_mc_idx	index	pgxz	tbase	8192 bytes	

(2 rows)

■ \di+索引名 显示某个索引的详细信息--只能在 dn 上面查看索引大小

```
postgres=# \di+ public.tbase_id_uidx
```

List of relations

Schema	Name	Type	Owner	Table	Size	Description
public	tbase_id_uidx	index	pgxz	tbase	8192 bytes	tbase 唯一索引

(1 row)

■ \di+通配符列出适配的索引--只能在 dn 上面查看索引大小

```
postgres=# \di+ *idx
```

List of relations

Schema	Name	Type	Owner	Table	Size	Description
pgxc	order_main_id_idx	index	pgxz	order_main	8192 bytes	
public	tbase_id_uidx	index	pgxz	tbase	8192 bytes	tbase 唯一索引
public	tbase_mc_idx	index	pgxz	tbase	8192 bytes	

(3 rows)

postgres=# \di+ *uidx--只能在 dn 上面查看索引大小

```

                                List of relations
 Schema |      Name      | Type | Owner | Table |      Size      | Description
-----+-----+-----+-----+-----+-----+-----
 public | tbase_id_uidx | index | pgxz  | tbase | 8192 bytes | tbase 唯一索引
(1 row)

```

6.4.3.20、函数相关操作

■ 建立函数

```

postgres=# CREATE OR REPLACE FUNCTION tbase_f1(a_1 text) returns text as
postgres-# $$
postgres-# begin
postgres-#     return a_1;
postgres-# end;
postgres-# $$
postgres-# language plpgsql;
CREATE FUNCTION
postgres=# CREATE OR REPLACE FUNCTION pgxc.tbase_f2(a_2 text) returns text as
postgres-# $$
postgres-# begin
postgres-#     return a_2;
postgres-# end;
postgres-# $$
postgres-# language plpgsql;
CREATE FUNCTION
postgres=#

```

■ \df 查看函数列表

```

postgres=# \df

                                List of functions
 Schema |      Name      | Result data type | Argument data types | Type
-----+-----+-----+-----+-----
 pgxc   | tbase_f2 | text              | a_2 text            | normal
 public | tbase_f1 | text              | a_1 text            | normal
(2 rows)

```

■ \df+ 查看函数列表详细信息（包含注释），定义

```

postgres=# \x
Expanded display is on.
postgres=# \df+ tbase*
List of functions

```

```

-[ RECORD 1 ]-----+-----
Schema          | pgxc
Name             | tbase_f2
Result data type | text
Argument data types | a_2 text
Type             | normal
Volatility       | volatile
Parallel         | unsafe
Owner            | pgxz
Security         | invoker
Access privileges |
Language         | plpgsql
Source code      |
                  | begin
                  |      return a_2;
                  | end;
Description      |

-[ RECORD 2 ]-----+-----
Schema          | public
Name             | tbase_f1
Result data type | text
Argument data types | a_1 text
Type             | normal
Volatility       | volatile
Parallel         | unsafe
Owner            | pgxz
Security         | invoker
Access privileges |
Language         | plpgsql
Source code      |
                  | begin
                  |      return a_1;
                  | end;
Description      |

```

■ \df+显示某个模式下的所有索引

```

-[ RECORD 1 ]-----+-----
Schema          | public
Name             | tbase_f1
Result data type | text
Argument data types | a_1 text
Type             | normal
Volatility       | volatile

```

```

Parallel      | unsafe
Owner         | pgxz
Security      | invoker
Access privileges |
Language      | plpgsql
Source code   |
              |          +
              |      begin      +
              |          return a_1;+
              |      end;        +
              |
Description   |

```

■ \df+函数名 显示某个函数的详细信息

```

postgres=# \df+ tbase_f1
List of functions
-[ RECORD 1 ]-----+-----
Schema          | public
Name            | tbase_f1
Result data type | text
Argument data types | a_1 text
Type            | normal
Volatility       | volatile
Parallel        | unsafe
Owner           | pgxz
Security        | invoker
Access privileges |
Language        | plpgsql
Source code     |
                |          +
                |      begin      +
                |          return a_1;+
                |      end;        +
                |
Description     |

```

■ \df+通配符列出适配的函数

```

postgres=# \df+ tbase*
List of functions
-[ RECORD 1 ]-----+-----
Schema          | pgxc
Name            | tbase_f2
Result data type | text
Argument data types | a_2 text
Type            | normal
Volatility       | volatile

```

```

Parallel      | unsafe
Owner         | pgxz
Security      | invoker
Access privileges |
Language      | plpgsql
Source code   |                               +
              | begin                          +
              |         return a_2;            +
              | end;                            +
              |

```

```

Description   |
-[ RECORD 2 ]-----+-----

```

```

Schema        | public
Name          | tbase_f1
Result data type | text
Argument data types | a_1 text
Type          | normal
Volatility     | volatile
Parallel      | unsafe
Owner         | pgxz
Security      | invoker
Access privileges |
Language      | plpgsql
Source code   |                               +
              | begin                          +
              |         return a_1;            +
              | end;                            +
              |
Description   |

```

postgres=# \df+ *f1

List of functions

```

-[ RECORD 1 ]-----+-----
Schema        | public
Name          | tbase_f1
Result data type | text
Argument data types | a_1 text
Type          | normal
Volatility     | volatile
Parallel      | unsafe
Owner         | pgxz
Security      | invoker
Access privileges |
Language      | plpgsql
Source code   |                               +
              | begin                          +

```

```

        |         return a_1;+
        |         end;          +
        |
Description |

```

6.4.3.21、自定义数据类型相关操作

■ 建立数据类型

```

postgres=# CREATE TYPE bug_status AS ENUM ('new', 'open', 'closed');
CREATE TYPE

```

■ \dT 查看自定义数据类型列表

```

postgres=# \dT

          List of data types
 Schema | Name      | Description
-----+-----+-----
 pg_oracle | nvarchar2 | oracle nvarchar2(length)
 pg_oracle | varchar2  | oracle varchar2(length)
 public    | bug_status |

```

■ \dT+查看自定义数据类型列表详细信息（包含 enum 类型的值）

```

postgres=# \dT+

          List of data types
 Schema | Name      | Internal name | Size | Elements | Access privileges | Description
-----+-----+-----+-----+-----+-----+-----
 pg_oracle | nvarchar2 | nvarchar2    | var  |          |                   | oracle nvarchar2(length)
 pg_oracle | varchar2  | varchar2     | var  |          |                   | oracle varchar2(length)
 public    | bug_status | bug_status   | 4    | new      +|                   |
           |            |              |      | open    +|                   |
           |            |              |      | closed  |                   |

```

■ \dT+显示某个模式下的所有自定义类型

```

postgres=# \dT+ public.*

          List of data types
 Schema | Name      | Internal name | Size | Elements | Access privileges | Description
-----+-----+-----+-----+-----+-----+-----
 public | bug_status | bug_status   | 4    | new      +|                   |
           |            |              |      | open    +|                   |
           |            |              |      | closed  |                   |

```

■ \dT+自定义数据类型 显示某个数据类型的详细信息

```
postgres=# \dT+ bug_status
```

List of data types

Schema	Name	Internal name	Size	Elements	Access privileges	Description
public	bug_status	bug_status	4	new	+	
				open	+	
				closed		

```
(1 row)
```

■ \dT+通配符列出适配的数据类型

```
postgres=# \dT+ bug_*
```

List of data types

Schema	Name	Internal name	Size	Elements	Owner	Access privileges	Description
public	bug_status	bug_status	4	new	pgxz		
				open	+		
				closed			

6.4.3.22、存储过程语句相关操作

■ 加载某个存储过程语言

```
postgres=# create language plpgsql;
CREATE LANGUAGE
```

■ \dL 查看存储过程语言列表

```
postgres=# \dL
```

List of languages

Name	Owner	Trusted	Description
plpgsql	pgxz	t	PL/pgSQL procedural language

```
(1 row)
```

6.4.3.23、列出表、视图和序列和它们相关的访问权限

■ 显示所有对象的访问权限

```
postgres=# \dp
```

Access privileges

Schema	Name	Type	Access privileges	Column privileges	Policies
public	pg_stat_statements	view	pgxz=arwdDxt/pgxz+		
			=r/pgxz		

```

public | tbase          | table | pgxz=arwdDxt/pgxz+|
      |                |       | pgxc=r/pgxz      |
public | tbase_seq      | sequence | pgxz=rwU/pgxz   +|
      |                |       | pgxc=rwU/pgxz   |
(3 rows)

```

■ \dp+某个对象名，只显示匹配的对象

```
postgres=# \dp tbase
```

```

Access privileges
Schema | Name | Type | Access privileges | Column privileges | Policies
-----+-----+-----+-----+-----+-----
public | tbase | table | pgxz=arwdDxt/pgxz+|
      |      |      | pgxc=r/pgxz      |
(1 row)

```

```
postgres=# \dp public.*
```

```

Access privileges
Schema | Name | Type | Access privileges | Column privileges | Policies
-----+-----+-----+-----+-----+-----
public | pg_stat_statements | view | pgxz=arwdDxt/pgxz+|
      |                  |      | =r/pgxz          |
public | tbase | table | pgxz=arwdDxt/pgxz+|
      |      |      | pgxc=r/pgxz      |
public | tbase_seq | sequence | pgxz=rwU/pgxz   +|
      |          |          | pgxc=rwU/pgxz   |
(3 rows)

```

6.4.3.24、列出库或用户定义的配置

```
postgres=# \drds
```

```

List of settings
Role | Database | Settings
-----+-----+-----
pgxc |          | log_statement=none
      | postgres | search_path="$user", public, pgxc
(2 rows)

```

6.4.3.25、copy 命令的使用

这是一个针对客户端文件操作的 copy 应用，服务器端 copy 使用见开发工程师文档 4.9

■ \copy to 将数据复制到本地文件中

```
postgres=# \copy tbase to '/data/tbase/tbase.txt';
```

```
postgres=# \! cat /data/tbase/tbase.txt
1         tbase
```

■ \copy from 将本地文件复制到数据表中

```
postgres=# \copy tbase from '/data/tbase/tbase.txt';
```

6.4.3.26、copy FROM stdin 使用方法

```
postgres=# COPY tbase (id, mc) FROM stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
```

```
>> 1      tbase
>> 2      \N
>> 3      pgxc
>> \.
```

```
postgres=# select * from tbase;
```

```
 id | mc
----+-----
  1 | tbase
  2 |
  3 | pgxc
(3 rows)
```

6.4.3.27、打印当前查询缓冲区到标准输出

```
postgres=# select * from tbase;
```

```
 id | mc
----+-----
  1 | tbase
(1 row)
```

```
postgres=# \p
select * from tbase;
```

6.4.3.28、自定义显示格式

■ 配置 null 的显示代替字符串

```
postgres=# \pset null '(null)'
Null display is "(null)".
postgres=# insert into tbase values(2,null);
INSERT 0 1
postgres=# select * from tbase;
 id | mc
```

```

-----+-----
 1 | tbase
 2 | (null)
(2 rows)

```

■ 不显示边框

```

postgres=# \set border 0
Border style is 0.
postgres=# select * from tbase;
id    mc
--
 1 tbase
 2 (null)
(2 rows)

```

■ 只显示记录

```

postgres=# \set tuples_only
Showing only tuples.
postgres=# select * from tbase;
 1 tbase
 2 (null)

```

■ 列之间使用逗号分隔

```

postgres=# \set format unaligned
Output format is unaligned.
postgres=# \set fieldsep ,
Field separator is ",".
postgres=# select * from tbase;
1,tbase
2,(null)

```

6.4.3.29、sql 命令帮助查看

```

postgres=# \h
Available help:

```

```

ABORT
DROP INDEX
ALTER AGGREGATE
DROP LANGUAGE
ALTER COLLATION
DROP MATERIALIZED VIEW

```

```

ALTER USER
LOCK
ALTER USER MAPPING
MOVE
ALTER VIEW
NOTIFY

```

```

CREATE ROLE
CREATE RULE
CREATE SCHEMA

```

。 。 。



```
postgres=# \h begin;
Command:      BEGIN
Description:  start a transaction block
Syntax:
BEGIN [ WORK | TRANSACTION ] [ transaction_mode [, ...] ]
```

where transaction_mode is one of:

```
ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ
UNCOMMITTED }
READ WRITE | READ ONLY
[ NOT ] DEFERRABLE
```

```
postgres=#
```

7、运维脚本使用

Tbase 运维脚本是一套基于 shell 的管理脚本，用于 tbase 集群服务异常情况下快速高效恢复服务

■ 脚本运行依赖环境

运行脚本的机器到各个节点机器需要打信任关系
脚本依赖于 confdb 数据

■ 脚本使用对象

主要面向 Tbase DBA

7.1、配置运行环境 env.sh

7.1.1、在 confdb 中创建集群选择函数

```
[root@VM_0_37_centos ~]# su tbaseConfdb
[tbaseConfdb@VM_0_37_centos root]$ psql -h 127.0.0.1 -p 54321 -d postgres -U postgres
psql (9.6.2)
Type "help" for help.

postgres=# create or replace function pgxzm_conf.cluster_name_set(a_cluster_name text) returns text as
$$
declare
    v_status integer;
    v_sql text;
begin
    select status into v_status from pgxzm_oss.tbl_oss_project where cluster_name=a_cluster_name;
```

```

if not found then
    return '集群 ('||a_cluster_name||') 不存在.';
end if;
if v_status!=4 then
    return '集群 ('||a_cluster_name||') 未初始化.';
end if;
v_sql:='
create or replace view pgxzm_conf.tbl_cluster_postgres_info as
select
    node_id,
    master_nid,
    tt_create as tt,
    node_name as nodename,
    case when role in (1,3,4) then 1 else 0 end as isprimary,
    case when role in (4,5) then 0 when role in (3) then 1 else 2 end as nodetype,
    node_ip as ip,
    pg_port as port,
    pooler_port,
    bin_dir as bindir,
    data_dir datadir,
    t.parenthostip,
    t.parentport,
    dbname as database,
    dbuser as datauser,
    dbpwd as password
from
    pgxzm_oss.tbl_oss_node
    left outer join (select node_id as p_node_id,node_ip as parenthostip,pg_port as parentport from
pgxzm_oss.tbl_oss_node where master_nid=node_id and cluster_name = '||a_cluster_name||') as t on
t.p_node_id=tbl_oss_node.master_nid and t.p_node_id!=tbl_oss_node.node_id
where
    tbl_oss_node.cluster_name = '||a_cluster_name||'
;
execute v_sql;
return '设置当前操作集群完成';
end;
$$
language plpgsql;
CREATE FUNCTION
postgres=#

```

7.1.2、配置 tbase 用户环境变量

打开脚本包，编辑 env.sh 文件，让集群环境修改成相应的参数值

```
#!/bin/sh
```

```

OSUSER="tbase"

#所有脚本只准许 pgxz 用户运行
if [ `whoami` != $OSUSER ];then
    echo "only $OSUSER can run me!"
    exit 1
fi

#confdb 所在机器 ip
PGHOST=127.0.0.1
#confdb port
PGPORT=54321
#集群名称
PGDATABASE=postgres
#数据库用户
PGUSER="postgres"
#ssh port
SSH_PORT=22
#集群拓补结构表
TABLE="pgxzm_conf.tbl_cluster_"$PGDATABASE"_info"
#center 所在路径, ansible 安装的在/home/pgxz/pgxzm/center
CENTER_PATH="/data/tbase/install/tbase_oss"
#agent 所在路径, ansible 安装的在/home/pgxz/pgxzm/agent
AGENT_PATH="/data/tbase/install/tbase_oss"
#Tbase 应用程序所在路径
PGXZ_PATH="/usr/local/install/tbase_pgxz"
#center 所在机器
CENTER_TABLE="pgxzm_conf.tbl_center_info"
#配置 PATH 的扩展路径
EXTPATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin"

export PGHOST PGPORT PGDATABASE PGUSER OSUSER

export PATH=$PGXZ_PATH/bin:$PATH
export LD_LIBRARY_PATH=$PGXZ_PATH/lib:${LD_LIBRARY_PATH}

```

7.1.3、配置 root 用户环境变量

```

#!/bin/sh

#所有脚本只准许 root 用户运行
if [ `whoami` != "root" ];then
    echo "only root can run me!"
    exit 1

```

```
fi
```

```
#confdb 所在机器 ip
PGHOST=127.0.0.1
#confdb port
PGPORT=54321
#集群名称
PGDATABASE=postgres
#数据库用户
PGUSER="postgres"
#ssh port
SSH_PORT=22
#集群拓补结构表
TABLE="pgxzm_conf.tbl_cluster_"$PGDATABASE"_info"
#center 所在路径, ansible 安装的在/home/pgxz/pgxzm/center
CENTER_PATH="/data/tbase/install/tbase_oss"
#agent 所在路径, ansible 安装的在/home/pgxz/pgxzm/agent
AGENT_PATH="/data/tbase/install/tbase_oss"
#Tbase 应用程序所在路径
PGXZ_PATH="/usr/local/install/tbase_pgxz"
#center 所在机器
CENTER_TABLE="pgxzm_conf.tbl_center_info"

export PGHOST PGPORT PGDATABASE PGUSER OSUSER

export PATH=$PGXZ_PATH/bin:$PATH
export LD_LIBRARY_PATH=$PGXZ_PATH/lib:${LD_LIBRARY_PATH}
```

7.1.4、配置当前操作集群

```
[tbase@VM_0_37_centos shell]$ ./tbase_cluster_set.sh tbase_01_1
```

设置当前操作集群完成

提示：集群名称在 3.3.5 集群管理页面可以查到

7.2、center 服务管理

7.2.1、启动 center 服务

```
[tbase@VM_0_37_centos shell]$ ./center_start.sh
start center begin...
ssh -p 22 -T tbase@172.16.0.37 "/data/tbase/install/tbase_oss/tools/op/start.sh>/dev/null 2>&1 & "
ssh -p 22 -T tbase@172.16.0.42 "/data/tbase/install/tbase_oss/tools/op/start.sh>/dev/null 2>&1 & "
start center end
```

如果某个节点启动后服务没有拉起来，你可以直接运行下面的脚本(ip 替换为故障机器 ip 地址)

```
ssh -p 22 -T tbase@172.16.0.37 "/data/tbase/install/tbase_oss/tools/op/start.sh"
```

系统会显示整个启动过程，出错信息直接抛出来，方便定位问题

7.2.2、停止 center 服务

```
[tbase@VM_0_37_centos shell]$ ./center_stop.sh
```

```
stop center begin...
```

```
ssh -p 22 -T tbase@172.16.0.37 "/data/tbase/install/tbase_oss/tools/op/stop.sh">/dev/null 2>&1 & "
```

```
ssh -p 22 -T tbase@172.16.0.42 "/data/tbase/install/tbase_oss/tools/op/stop.sh">/dev/null 2>&1 & "
```

```
stop center end
```

注意，停止后请使用 `center_status.sh` 脚本确认 center 服务完成停止后才能进行其它操作

如果某个节点服务无法正常关闭，你可以直接运行下面的脚本(ip 替换为故障机器 ip 地址)

```
ssh -p 22 -T tbase@172.16.0.37 "/data/tbase/install/tbase_oss/tools/op/stop.sh"
```

系统会显示整个关闭过程，出错信息直接抛出来，方便定位问题

7.2.3、查看 center 状态

```
[tbase@VM_0_37_centos shell]$ ./center_status.sh
```

```
show center status begin...
```

```
ssh -p 22 -T tbase@172.16.0.37 "ps -ewf | grep config | grep -v color |grep -v 'grep' | grep -v 'root'|grep -v ./center_status.sh"
```

```
tbase      31984  31964   2 11:31 ?           00:00:00 /data/tbase/install/tbase_oss/bin/oss_server -conf /data/tbase/install/tbase_oss/config/tbase_oss_conf.ini -log /data/tbase/install/tbase_oss/logs/ -level 1
```

```
ssh -p 22 -T tbase@172.16.0.42 "ps -ewf | grep config | grep -v color |grep -v 'grep' | grep -v 'root'|grep -v ./center_status.sh"
```

```
tbase      23619  23600   1 11:31 ?           00:00:00 /data/tbase/install/tbase_oss/bin/oss_server -conf /data/tbase/install/tbase_oss/config/tbase_oss_conf.ini -log /data/tbase/install/tbase_oss/logs/ -level 1
```

```
show center status end
```

7.2.4、center 应用程序升级

```
[tbase@VM_0_37_centos shell]$ ./center_upgrade.sh
```

注意：升级前请把 center 服务关闭掉

7.3、agent 服务管理

7.3.1、启动当前集群 agent 服务

```
[tbase@VM_0_37_centos shell]$ ./agent_start.sh
start agent begin...
ssh -p 22 -T tbase@172.16.0.42 "/data/tbase/install/tbase_oss/tools/op/start.sh>/dev/null 2>&1 & "
ssh -p 22 -T tbase@172.16.0.37 "/data/tbase/install/tbase_oss/tools/op/start.sh>/dev/null 2>&1 & "
start agent end
```

如果某个节点启动后服务没有拉起来，你可以直接运行下面的脚本(ip 替换为故障机器 ip 地址)

```
ssh -p 22 -T tbase@172.16.0.42 "/data/tbase/install/tbase_oss/tools/op/start.sh"
```

系统会显示整个启动过程，出错信息直接抛出来，方便定位问题

7.3.2、停止当前集群 agent 服务

```
[tbase@VM_0_37_centos shell]$ ./agent_stop.sh
stop agent begin...
ssh -p 22 -T tbase@172.16.0.42 "/data/tbase/install/tbase_oss/tools/op/stop.sh>/dev/null 2>&1 & "
ssh -p 22 -T tbase@172.16.0.37 "/data/tbase/install/tbase_oss/tools/op/stop.sh>/dev/null 2>&1 & "
stop agent end
```

注意，停止后请使用 `agent_status.sh` 脚本确认 agent 服务完成停止后才能进行其它操作

如果某个节点服务无法正常关闭，你可以直接运行下面的脚本(ip 替换为故障机器 ip 地址)

```
ssh -p 22 -T tbase@172.16.0.42 "/data/tbase/install/tbase_oss/tools/op/stop.sh"
```

系统会显示整个关闭过程，出错信息直接抛出来，方便定位问题

7.3.3、查看当前集群 agent 状态

```
[tbase@VM_0_37_centos shell]$ ./agent_status.sh
show agent status begin...
ssh -p 22 -T tbase@172.16.0.42 "ps -ewf | grep config | grep -v color |grep -v 'grep' | grep -v 'root'|grep -v ./agent_status.sh"
tbase      12565      1  0 11:43 ?           00:00:00 /data/tbase/install/tbase_oss/bin/oss_server -conf
/data/tbase/install/tbase_oss/config//tbase_oss_conf.ini -log /data/tbase/install/tbase_oss/logs/ -level 1
ssh -p 22 -T tbase@172.16.0.37 "ps -ewf | grep config | grep -v color |grep -v 'grep' | grep -v 'root'|grep -v ./agent_status.sh"
tbase      25264      1  0 11:43 ?           00:00:00 /data/tbase/install/tbase_oss/bin/oss_server -conf
/data/tbase/install/tbase_oss/config//tbase_oss_conf.ini -log /data/tbase/install/tbase_oss/logs/ -level 1
show agent status end
```

7.3.4、启动所有机器 agent 服务

```
[tbase@VM_0_37_centos shell]$ ./agent_all_start.sh
```

7.3.5、停止所有机器 agent 服务

```
[tbase@VM_0_37_centos shell]$ ./agent_all_stop.sh
```

7.3.6、查看所有机器 agent 服务状态

```
[tbase@VM_0_37_centos shell]$ ./agent_all_status.sh
```

7.3.7、agent 应用程序升级

```
[tbase@VM_0_37_centos shell]$ ./agent_upgrade.sh
```

注意：升级前请把 agent 服务关闭掉

7.4、tbase 服务管理

7.3.1、启动 tbase 服务

手工操作 TBase 服务时务必把 agent 服务给关掉，并且确认是不是自动拉起。

```
[tbase@VM_0_37_centos shell]$ ./tbase_start.sh
```

start gtm begin...

```
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data4/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export LD_LIBRARY_PATH=/data/tbase/data4/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data4/tbase_01_1/2.0.6/install/tbase_pgxz/bin/gtm_ctl start -Z gtm -D /data/tbase/data4/tbase_01_1/data/gtm001>/dev/null 2>&1 & "
ssh -p 22 -T tbase@172.16.0.42 "export PATH=/data/tbase/data4/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export LD_LIBRARY_PATH=/data/tbase/data4/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data4/tbase_01_1/2.0.6/install/tbase_pgxz/bin/gtm_ctl start -Z gtm -D /data/tbase/data4/tbase_01_1/data/gtm002>/dev/null 2>&1 & "
start gtm end
```

start dn_master begin...

```
ssh -p 22 -T @172.16.0.37 "export PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export LD_LIBRARY_PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/bin/pg_ctl start -Z datanode -D /data/tbase/data1/tbase_01_1/data/dn001>/dev/null 2>&1 & "
ssh -p 22 -T @172.16.0.42 "export PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export LD_LIBRARY_PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz/bin/pg_ctl start -Z datanode -D /data/tbase/data2/tbase_01_1/data/dn002>/dev/null 2>&1 & "
```

& "

start dn_master end

start dn_slave begin...

```
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export
LD_LIBRARY_PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data2/tbase_01_1/2.
0.6/install/tbase_pgxz/bin/pg_ctl start -Z datanode -D /data/tbase/data2/tbase_01_1/data/dn002>/dev/null 2>&1
& "
```

```
ssh -p 22 -T tbase@172.16.0.42 "export PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export
LD_LIBRARY_PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data1/tbase_01_1/2.
0.6/install/tbase_pgxz/bin/pg_ctl start -Z datanode -D /data/tbase/data1/tbase_01_1/data/dn001>/dev/null 2>&1
& "
```

start dn_slave end

start cn begin...

```
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export
LD_LIBRARY_PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data3/tbase_01_1/2.
0.6/install/tbase_pgxz/bin/pg_ctl start -Z coordinator -D /data/tbase/data3/tbase_01_1/data/cn001>/dev/null
2>&1 & "
```

```
ssh -p 22 -T tbase@172.16.0.42 "export PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export
LD_LIBRARY_PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data3/tbase_01_1/2.
0.6/install/tbase_pgxz/bin/pg_ctl start -Z coordinator -D /data/tbase/data3/tbase_01_1/data/cn002>/dev/null
2>&1 & "
```

start cn end

如果某个节点启动后服务没有拉起来，你可以直接运行下面的脚本(ip 替换为故障机器 ip 地址，红色 data 换成故障节点的目录地址)

```
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export
LD_LIBRARY_PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data3/tbase_01_1/2.
0.6/install/tbase_pgxz/bin/pg_ctl start -Z coordinator -D /data/tbase/data3/tbase_01_1/data/cn001"
```

系统会显示整个启动过程，出错信息直接抛出来，方便定位问题

7.3.2、停止 tbase 服务

7.3.2.1、停止所有节点

```
[tbase@VM_0_37_centos shell]$ ./tbase_stop.sh
```

stop cn begin...

```
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export
LD_LIBRARY_PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data3/tbase_01_1/2.
0.6/install/tbase_pgxz/bin/pg_ctl stop -m i -Z coordinator -D /data/tbase/data3/tbase_01_1/data/cn001>/dev/null
2>&1 & "
```

```
ssh -p 22 -T tbase@172.16.0.42 "export PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export
LD_LIBRARY_PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data3/tbase_01_1/2.
```

```
0.6/install/tbase_pgxz/bin/pg_ctl stop -m i -Z coordinator -D /data/tbase/data3/tbase_01_1/data/cn002>/dev/null
2>&1 & "
stop cn end
```

stop dn_slave begin...

```
ssh -p 22 -T tbase@172.16.0.42 "export PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export
LD_LIBRARY_PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data1/tbase_01_1/2.
0.6/install/tbase_pgxz/bin/pg_ctl stop -m i -Z datanode -D /data/tbase/data1/tbase_01_1/data/dn001>/dev/null
2>&1 & "
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export
LD_LIBRARY_PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data2/tbase_01_1/2.
0.6/install/tbase_pgxz/bin/pg_ctl stop -m i -Z datanode -D /data/tbase/data2/tbase_01_1/data/dn002>/dev/null
2>&1 & "
stop dn_slave end
```

stop dn_master begin...

```
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export
LD_LIBRARY_PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data1/tbase_01_1/2.
0.6/install/tbase_pgxz/bin/pg_ctl stop -m i -Z datanode -D /data/tbase/data1/tbase_01_1/data/dn001>/dev/null
2>&1 & "
ssh -p 22 -T tbase@172.16.0.42 "export PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export
LD_LIBRARY_PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data2/tbase_01_1/2.
0.6/install/tbase_pgxz/bin/pg_ctl stop -m i -Z datanode -D /data/tbase/data2/tbase_01_1/data/dn002>/dev/null
2>&1 & "
stop dn_master end
```

stop gtm begin...

```
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data4/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export
LD_LIBRARY_PATH=/data/tbase/data4/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data4/tbase_01_1/2.
0.6/install/tbase_pgxz/bin/gtm_ctl stop -Z gtm -D /data/tbase/data4/tbase_01_1/data/gtm001>/dev/null 2>&1 & "
ssh -p 22 -T tbase@172.16.0.42 "export PATH=/data/tbase/data4/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export
LD_LIBRARY_PATH=/data/tbase/data4/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data4/tbase_01_1/2.
0.6/install/tbase_pgxz/bin/gtm_ctl stop -Z gtm -D /data/tbase/data4/tbase_01_1/data/gtm002>/dev/null 2>&1 & "
stop gtm end
```

注意，脚本会检查 agent 服务是否处于停止状态，如果不是会提示先关闭 agent 服务，因为 agent 服务会自动拉起 tbase 各个节点。停止后请使用 tbase_status.sh 脚本确认 tbase 服务完成停止后才能进行其它操作

如果某个节点服务无法正常关闭，你可以直接运行下面的脚本(ip 替换为故障机器 ip 地址，红色 data 换成故障节点的目录地址)

```
ssh -p 22 -T tbase@172.16.0.42 "export PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export
LD_LIBRARY_PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz/lib;/data/tbase/data2/tbase_01_1/2.
0.6/install/tbase_pgxz/bin/pg_ctl stop -m i -Z datanode -D /data/tbase/data2/tbase_01_1/data/dn002 "
```

系统会显示整个关闭过程，出错信息直接抛出来，方便定位问题

7.3.2.2、停止 cn 节点

```
[tbase@VM_0_37_centos shell]$ ./tbase_stop_cn.sh
stop cn begin...
ssh -p 22 -T tbase@172.16.0.37 "export
PATH=/data/tbase/data3/tbase_01_5/2.0.7/install/tbase_pgxz//bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/r
oot/bin;export
LD_LIBRARY_PATH=/data/tbase/data3/tbase_01_5/2.0.7/install/tbase_pgxz//lib;/data/tbase/data3/tbase_01_5/2.
0.7/install/tbase_pgxz//bin/pg_ctl stop -m i -Z coordinator -D /data/tbase/data3/tbase_01_5/data/cn001>/dev/null
2>&1 & "
ssh -p 22 -T tbase@172.16.0.42 "export
PATH=/data/tbase/data3/tbase_01_5/2.0.7/install/tbase_pgxz//bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/r
oot/bin;export
LD_LIBRARY_PATH=/data/tbase/data3/tbase_01_5/2.0.7/install/tbase_pgxz//lib;/data/tbase/data3/tbase_01_5/2.
0.7/install/tbase_pgxz//bin/pg_ctl stop -m i -Z coordinator -D /data/tbase/data3/tbase_01_5/data/cn002>/dev/null
2>&1 & "
stop cn end
```

7.3.2.3、停止 dn 备节点

```
[tbase@VM_0_37_centos shell]$ ./tbase_stop_dn_master.sh
stop dn_master begin...
ssh -p 22 -T tbase@172.16.0.37 "export
PATH=/data/tbase/data1/tbase_01_5/2.0.7/install/tbase_pgxz//bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/r
oot/bin;export
LD_LIBRARY_PATH=/data/tbase/data1/tbase_01_5/2.0.7/install/tbase_pgxz//lib;/data/tbase/data1/tbase_01_5/2.
0.7/install/tbase_pgxz//bin/pg_ctl stop -m i -Z datanode -D /data/tbase/data1/tbase_01_5/data/dn001>/dev/null
2>&1 & "
ssh -p 22 -T tbase@172.16.0.42 "export
PATH=/data/tbase/data2/tbase_01_5/2.0.7/install/tbase_pgxz//bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/r
oot/bin;export
LD_LIBRARY_PATH=/data/tbase/data2/tbase_01_5/2.0.7/install/tbase_pgxz//lib;/data/tbase/data2/tbase_01_5/2.
0.7/install/tbase_pgxz//bin/pg_ctl stop -m i -Z datanode -D /data/tbase/data2/tbase_01_5/data/dn002>/dev/null
2>&1 & "
stop dn_master end
```

7.3.2.4、停止 gtm 节点

```
[tbase@VM_0_37_centos shell]$ ./tbase_stop_gtm.sh
stop gtm begin...
ssh -p 22 -T tbase@172.16.0.37 "export
PATH=/data/tbase/data4/tbase_01_5/2.0.7/install/tbase_pgxz//bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/r
oot/bin;export
LD_LIBRARY_PATH=/data/tbase/data4/tbase_01_5/2.0.7/install/tbase_pgxz//lib;/data/tbase/data4/tbase_01_5/2.
0.7/install/tbase_pgxz//bin/gtm_ctl stop -Z gtm -D /data/tbase/data4/tbase_01_5/data/gtm001>/dev/null 2>&1 & "
```

```
ssh -p 22 -T tbase@172.16.0.42 "export
PATH=/data/tbase/data4/tbase_01_5/2.0.7/install/tbase_pgxz//bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/r
oot/bin;export
LD_LIBRARY_PATH=/data/tbase/data4/tbase_01_5/2.0.7/install/tbase_pgxz//lib:/data/tbase/data4/tbase_01_5/2.
0.7/install/tbase_pgxz//bin/gtm_ctl stop -Z gtm -D /data/tbase/data4/tbase_01_5/data/gtm002>/dev/null 2>&1 & "
stop gtm end
```

7.3.3、查看 tbase 状态

```
[tbase@VM_0_37_centos shell]$ ./tbase_status.sh
```

show gtm status begin...

```
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data4/tbase_01_1/2.0.6/install/tbase_pgxz//bin;export
LD_LIBRARY_PATH=/data/tbase/data4/tbase_01_1/2.0.6/install/tbase_pgxz//lib:/data/tbase/data4/tbase_01_1/2.
0.6/install/tbase_pgxz//bin/gtm_ctl status -Z gtm -D /data/tbase/data4/tbase_01_1/data/gtm001 -H 127.0.0.1 -P
11006 "
```

```
gtm_ctl: server is running (PID: 20870)
```

```
current node [primary] gts[658380396748] xlog [0/0]
```

```
[standby/sync] host[172.16.0.42:11006] gts [658679318412] xlog[0/0]
```

```
"-D" "/data/tbase/data4/tbase_01_1/data/gtm001"
```

```
ssh -p 22 -T tbase@172.16.0.42 "export PATH=/data/tbase/data4/tbase_01_1/2.0.6/install/tbase_pgxz//bin;export
LD_LIBRARY_PATH=/data/tbase/data4/tbase_01_1/2.0.6/install/tbase_pgxz//lib:/data/tbase/data4/tbase_01_1/2.
0.6/install/tbase_pgxz//bin/gtm_ctl status -Z gtm -D /data/tbase/data4/tbase_01_1/data/gtm002 -H 127.0.0.1 -P
11006 "
```

```
gtm_ctl: server is running (PID: 5169)
```

```
current node [standby] gts[658679444836] xlog [0/0]
```

```
"-D" "/data/tbase/data4/tbase_01_1/data/gtm002"
```

```
show gtm status end
```

show dn_master status begin...

```
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz//bin;export
LD_LIBRARY_PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz//lib:/data/tbase/data1/tbase_01_1/2.
0.6/install/tbase_pgxz//bin/pg_ctl status -Z datanode -D /data/tbase/data1/tbase_01_1/data/dn001"
```

```
pg_ctl: server is running (PID: 20906)
```

```
/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/bin/postgres
```

```
"--datanode"
```

```
"-D"
```

```
"/data/tbase/data1/tbase_01_1/data/dn001"
```

```
ssh -p 22 -T tbase@172.16.0.42 "export PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz//bin;export
LD_LIBRARY_PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz//lib:/data/tbase/data2/tbase_01_1/2.
0.6/install/tbase_pgxz//bin/pg_ctl status -Z datanode -D /data/tbase/data2/tbase_01_1/data/dn002"
```

```
pg_ctl: server is running (PID: 5185)
```

```
/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz/bin/postgres
```

```
"--datanode"
```

```
"-D"
```

```
"/data/tbase/data2/tbase_01_1/data/dn002"
```

```
show dn_master status end
```

show dn_slave status begin...

```
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz//bin;export
```

```
LD_LIBRARY_PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz/lib:/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz/bin/pg_ctl status -Z datanode -D /data/tbase/data2/tbase_01_1/data/dn002 "
pg_ctl: server is running (PID: 20930)
/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz/bin/postgres          "--datanode"          "-D"
"/data/tbase/data2/tbase_01_1/data/dn002"
ssh -p 22 -T tbase@172.16.0.42 "export PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export LD_LIBRARY_PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/lib:/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/bin/pg_ctl status -Z datanode -D /data/tbase/data1/tbase_01_1/data/dn001 "
pg_ctl: server is running (PID: 5202)
/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/bin/postgres          "--datanode"          "-D"
"/data/tbase/data1/tbase_01_1/data/dn001"
show dn_slave status end
```

show cn status begin...

```
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export LD_LIBRARY_PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/lib:/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/bin/pg_ctl status -Z coordinator -D /data/tbase/data3/tbase_01_1/data/cn001"
pg_ctl: server is running (PID: 20954)
/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/bin/postgres          "--coordinator"       "-D"
"/data/tbase/data3/tbase_01_1/data/cn001"
ssh -p 22 -T tbase@172.16.0.42 "export PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export LD_LIBRARY_PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/lib:/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/bin/pg_ctl status -Z coordinator -D /data/tbase/data3/tbase_01_1/data/cn002"
pg_ctl: server is running (PID: 5228)
/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/bin/postgres          "--coordinator"       "-D"
"/data/tbase/data3/tbase_01_1/data/cn002"
show cn status end
```

看到 server is running (PID: 31885)这样的信息表示服务正在运行, 看到 no server running 这样的信息表示服务处于停止状态

7.3.4、reload 服务

```
[tbase@VM_0_37_centos shell]$ ./tbase_reload.sh
```

reload cn begin...

```
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export LD_LIBRARY_PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/lib:/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/bin/pg_ctl reload -Z coordinator -D /data/tbase/data3/tbase_01_1/data/cn001>/dev/null 2>&1 & "
ssh -p 22 -T tbase@172.16.0.42 "export PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/bin;export LD_LIBRARY_PATH=/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/lib:/data/tbase/data3/tbase_01_1/2.0.6/install/tbase_pgxz/bin/pg_ctl reload -Z coordinator -D /data/tbase/data3/tbase_01_1/data/cn002>/dev/null 2>&1 & "
reload cn end
```

reload dn_slave begin...

```
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz//bin;export
LD_LIBRARY_PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz//lib;/data/tbase/data2/tbase_01_1/2.
0.6/install/tbase_pgxz//bin/pg_ctl reload -Z datanode -D /data/tbase/data2/tbase_01_1/data/dn002>/dev/null 2>&1
& "
```

```
ssh -p 22 -T tbase@172.16.0.42 "export PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz//bin;export
LD_LIBRARY_PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz//lib;/data/tbase/data1/tbase_01_1/2.
0.6/install/tbase_pgxz//bin/pg_ctl reload -Z datanode -D /data/tbase/data1/tbase_01_1/data/dn001>/dev/null 2>&1
& "
```

```
reload dn_slave end
```

reload dn_master begin...

```
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz//bin;export
LD_LIBRARY_PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz//lib;/data/tbase/data1/tbase_01_1/2.
0.6/install/tbase_pgxz//bin/pg_ctl reload -Z datanode -D /data/tbase/data1/tbase_01_1/data/dn001>/dev/null 2>&1
& "
```

```
ssh -p 22 -T tbase@172.16.0.42 "export PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz//bin;export
LD_LIBRARY_PATH=/data/tbase/data2/tbase_01_1/2.0.6/install/tbase_pgxz//lib;/data/tbase/data2/tbase_01_1/2.
0.6/install/tbase_pgxz//bin/pg_ctl reload -Z datanode -D /data/tbase/data2/tbase_01_1/data/dn002>/dev/null 2>&1
& "
```

```
reload dn_master end
```

如果某个节点服务 reload 后参数不生效, 你可以直接运行下面的脚本(ip 替换为故障机器 ip 地址, 红色 data 换成故障节点的目录地址)

```
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz//bin;export
LD_LIBRARY_PATH=/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz//lib;/data/tbase/data1/tbase_01_1/2.
0.6/install/tbase_pgxz//bin/pg_ctl reload -Z datanode -D /data/tbase/data1/tbase_01_1/data/dn001 "
```

系统会显示整个 reload 过程, 出错信息直接抛出来, 方便定位问题

7.3.5、备份 tbase 配置文件

```
[tbase@VM_0_37_centos shell]$ ./tbase_conf_backup.sh
```

backup gtm conf

```
ssh -p 22 -T tbase@172.16.0.37 "cp /data/tbase/data4/tbase_01_1/data/gtm001/gtm.conf
/data/tbase/data4/tbase_01_1/data/gtm001/gtm.conf.2018_08_14_11_55_05 "
```

```
ssh -p 22 -T tbase@172.16.0.42 "cp /data/tbase/data4/tbase_01_1/data/gtm002/gtm.conf
/data/tbase/data4/tbase_01_1/data/gtm002/gtm.conf.2018_08_14_11_55_05 "
```

backup dn_master conf

```
ssh -p 22 -T tbase@172.16.0.37 "cp /data/tbase/data1/tbase_01_1/data/dn001/postgresql.conf
/data/tbase/data1/tbase_01_1/data/dn001/postgresql.conf.2018_08_14_11_55_05 "
```

```
ssh -p 22 -T tbase@172.16.0.37 "cp /data/tbase/data1/tbase_01_1/data/dn001/pg_hba.conf
/data/tbase/data1/tbase_01_1/data/dn001/pg_hba.conf.2018_08_14_11_55_05 "
```

```
ssh -p 22 -T tbase@172.16.0.42 "cp /data/tbase/data2/tbase_01_1/data/dn002/postgresql.conf
```

```
/data/tbase/data2/tbase_01_1/data/dn002/postgresql.conf.2018_08_14_11_55_05 "
```

```
ssh -p 22 -T tbase@172.16.0.42 "cp /data/tbase/data2/tbase_01_1/data/dn002/pg_hba.conf
/data/tbase/data2/tbase_01_1/data/dn002/pg_hba.conf.2018_08_14_11_55_05 "
```

backup dn_slave conf

```
ssh -p 22 -T tbase@172.16.0.37 "cp /data/tbase/data2/tbase_01_1/data/dn002/postgresql.conf
/data/tbase/data2/tbase_01_1/data/dn002/postgresql.conf.2018_08_14_11_55_05 "
```

```
ssh -p 22 -T tbase@172.16.0.37 "cp /data/tbase/data2/tbase_01_1/data/dn002/pg_hba.conf
/data/tbase/data2/tbase_01_1/data/dn002/pg_hba.conf.2018_08_14_11_55_05 "
```

```
ssh -p 22 -T tbase@172.16.0.37 "cp /data/tbase/data2/tbase_01_1/data/dn002/recovery.conf
/data/tbase/data2/tbase_01_1/data/dn002/recovery.conf.2018_08_14_11_55_05 "
```

```
ssh -p 22 -T tbase@172.16.0.42 "cp /data/tbase/data1/tbase_01_1/data/dn001/postgresql.conf
/data/tbase/data1/tbase_01_1/data/dn001/postgresql.conf.2018_08_14_11_55_05 "
```

```
ssh -p 22 -T tbase@172.16.0.42 "cp /data/tbase/data1/tbase_01_1/data/dn001/pg_hba.conf
/data/tbase/data1/tbase_01_1/data/dn001/pg_hba.conf.2018_08_14_11_55_05 "
```

```
ssh -p 22 -T tbase@172.16.0.42 "cp /data/tbase/data1/tbase_01_1/data/dn001/recovery.conf
/data/tbase/data1/tbase_01_1/data/dn001/recovery.conf.2018_08_14_11_55_05 "
```

backup cn conf

```
ssh -p 22 -T tbase@172.16.0.37 "cp /data/tbase/data3/tbase_01_1/data/cn001/postgresql.conf
/data/tbase/data3/tbase_01_1/data/cn001/postgresql.conf.2018_08_14_11_55_05 "
```

```
ssh -p 22 -T tbase@172.16.0.37 "cp /data/tbase/data3/tbase_01_1/data/cn001/pg_hba.conf
/data/tbase/data3/tbase_01_1/data/cn001/pg_hba.conf.2018_08_14_11_55_05 "
```

```
ssh -p 22 -T tbase@172.16.0.42 "cp /data/tbase/data3/tbase_01_1/data/cn002/postgresql.conf
/data/tbase/data3/tbase_01_1/data/cn002/postgresql.conf.2018_08_14_11_55_05 "
```

```
ssh -p 22 -T tbase@172.16.0.42 "cp /data/tbase/data3/tbase_01_1/data/cn002/pg_hba.conf
/data/tbase/data3/tbase_01_1/data/cn002/pg_hba.conf.2018_08_14_11_55_05 "
```

```
[tbase@VM_0_37_centos shell]$
```

根据节点的类型，脚本备份不同的文件，备份文件存放在节点的目录下，文件为 postgresql.conf.2018_08_14_11_55_05 或者 pg_hba.conf.2018_08_14_11_55_05 或者 recovery.conf.2018_08_14_11_55_05 或者 gtm.conf.2018_08_14_11_55_05，数字为备份时间戳，xxx 分别为

7.3.6、tbase 应用程序升级

```
[tbase@VM_0_37_centos shell]$ ./tbase_upgrade.sh
```

```
gtm_ctl: could not open PID file "/data/tbase/data4/tbase_01_1/data/gtm001/gtm.pid": No such file or directory
```

```
gtm_ctl: could not open PID file "/data/tbase/data4/tbase_01_1/data/gtm002/gtm.pid": No such file or directory
```

```
tar xzf pgxz.tar.gz -C ./
```

```
tbase upgrade begin...
```

```
ssh -p 22 -T pgxz@172.16.0.13 "mkdir -p /data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/"
```

```
scp -P 22 -r ./pgxz/* tbase@172.16.0.13:/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/ > /dev/null
```

```
ssh -p 22 -T pgxz@172.16.0.13 "md5sum /data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/bin/postgres"
```

```
bfe745482729626395b748c43b370479 /data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/bin/postgres
```

```
ssh -p 22 -T pgxz@172.16.0.3 "mkdir -p /data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/"
```

```
scp -P 22 -r ./pgxz/* tbase@172.16.0.3:/data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/ > /dev/null
ssh -p 22 -T pgxz@172.16.0.3 "md5sum /data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/bin/postgres"
bfe745482729626395b748c43b370479 /data/tbase/data1/tbase_01_1/2.0.6/install/tbase_pgxz/bin/postgres
tbase upgrade end
```

TBase 应用程序升级执行步骤如下：

- 检查 pgxz.tar.gz 升级包是否存在
- 检查 tbase 服务是否处于运行状态，如果处于运行状态，升级程序尝试关闭服务，关闭成功有继续执行升级程序，如果关闭失败则提示失败并退出。
- 解压缩 pgxz.tar.gz 包
- 创建目录（目录如果存在则跳过）
- scp 程序到 tbase 安装目录
- 输出升级后的 postgres 应用程序 md5 值，所有节点升级后这个值要一致

升级应用程序时请退出所有 psql 应用连接，否则导致 psql 应用升级失败

7.3.7、tbase 证书更新

```
[tbase@VM_0_37_centos shell]$ mkdir -p /data/tbase/shell/pgxz_license/
上传 license.info 文件到 /data/tbase/shell/pgxz_license/目录下,然后执行更新脚本
[tbase@VM_0_37_centos shell]$ ./tbase_pg_hba_scp.sh pgxz_license
```

证书重新生效需要重新 cn 的节点服务

7.5、tbase 管理脚本

7.5.1、2pc 残留处理

cn 节点 2pc 残留 rollback

```
[tbase@VM_0_37_centos shell]$ ./tbase_cn_2pc_rollback.sh
```

dn 节点 2pc 残留 rollback

```
[tbase@VM_0_37_centos shell]$ ./tbase_dn_2pc_rollback.sh
```

7.5.2、向所有 cn 节点发出 sql 查询

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_cn.sh "select count(1) from pg_stat_activity"
cn001 --- psql -h 172.16.0.37 -p 11000 -d postgres -U tbase -c "select count(1) from pg_stat_activity"
count
-----
```

(1 row)

```
cn002 --- psql -h 172.16.0.42 -p 11000 -d postgres -U tbase -c "select count(1) from pg_stat_activity"
count
```

```
-----
      17
```

(1 row)

7.5.3、向所有 dn 主节点发出 sql 查询

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_dn_master.sh "select count(1) from pg_stat_activity"
```

```
dn001 --- psql -h 172.16.0.37 -p 11002 -d postgres -U tbase -c "select count(1) from pg_stat_activity"
count
```

```
-----
      14
```

(1 row)

```
dn002 --- psql -h 172.16.0.42 -p 11004 -d postgres -U tbase -c "select count(1) from pg_stat_activity"
count
```

```
-----
      14
```

(1 row)

7.5.4、向所有 dn 备节点发出 sql 查询

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_dn_slave.sh "select count(1) from pg_stat_activity"
```

```
psql -h 172.16.0.37 -p 11004 -d postgres -U tbase -c "select count(1) from pg_stat_activity"
count
```

```
-----
       6
```

(1 row)

```
psql -h 172.16.0.42 -p 11002 -d postgres -U tbase -c "select count(1) from pg_stat_activity"
count
```

```
-----
       6
```

(1 row)

7.5.5、向当前集群所有主机发出 shell 命令

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_command.sh "/usr/bin/du /data/tbase/ --max-depth=1 -h"
```

```
ssh -p 22 -T tbase@172.16.0.42 "/usr/bin/du /data/tbase/ --max-depth=1 -h"
```

```
16K      /data/tbase/.ssh
```

```

485M    /data/tbase/install
9.4G    /data/tbase/data1
6.9G    /data/tbase/data2
4.9G    /data/tbase/data3
245M    /data/tbase/data4
4.0K    /data/tbase/.cache
0       /data/tbase/.config
22G     /data/tbase/
ssh -p 22 -T tbase@172.16.0.37 "/usr/bin/du /data/tbase/ --max-depth=1 -h"
16K     /data/tbase/.ssh
519M    /data/tbase/install
7.0G    /data/tbase/data1
9.3G    /data/tbase/data2
4.9G    /data/tbase/data3
171M    /data/tbase/data4
4.0K    /data/tbase/.cache
0       /data/tbase/.config
79M     /data/tbase/shell
3.2G    /data/tbase/tpch
3.7M    /data/tbase/pgbench
26G     /data/tbase/

```

上面显示节点占用空间对比

7.5.6、复制某个文件或目录到**当前集群**所有主机

```

[tbase@VM_0_37_centos shell]$ ./tbase_run_scp.sh "/data/tbase/shell/sysctl.conf" "/tmp"
scp -P 22 -r /data/tbase/shell/sysctl.conf tbase@172.16.0.42:/tmp
sysctl.conf
100%    0      0.0KB/s   00:00
scp -P 22 -r /data/tbase/shell/sysctl.conf tbase@172.16.0.37:/tmp
sysctl.conf
100%    0      0.0KB/s   00:00
[tbase@VM_0_37_centos shell]$

```

7.5.7、复制 **pg_hba.conf/postgresql.conf.user** 到**当前集群**所有 **cn,dn** 节点

```

[pgxz@VM_0_3_centos ~]$ ./tbase_pg_hba_scp.sh "/data/pgxz/data/pgxz/cn01/pg_hba.conf"

backup dn_master conf
scp -P 22 -r /data/pgxz/data/pgxz/cn01/pg_hba.conf pngx@172.16.0.29:/data/pgxz/data/pgxz/dn01
pg_hba.conf
100%  444    0.4KB/s   00:00
scp -P 22 -r /data/pgxz/data/pgxz/cn01/pg_hba.conf pngx@172.16.0.29:/data/pgxz/data/pgxz/dn02
pg_hba.conf

```

。 。 。

使用这个功能可以方便实现节点参数统一

7.5.8、向所有 dn 主节点发出 sql 查询--后台执行

可以用于并发执行一些需要长时间运行的命令，如

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_back_sql_dn_master.sh "insert into t select * from t"
```

7.5.9、向当前集群所有主机发出 shell 命令--root 用户

该脚本跟 7.5.5 功能是一致的，只是用运行用户限制为 root 用户，所以该脚本可以运行管理员命令，如下面命令用于释放内存

```
[root@VM_0_37_centos shell]#./tbase_run_command_root.sh "sync;sync;sync;echo 3 > /proc/sys/vm/drop_caches &"
ssh -p 22 -T pgxz@172.16.0.42 "sync;sync;sync;echo 3 > /proc/sys/vm/drop_caches &"
ssh -p 22 -T pgxz@172.16.0.37 "sync;sync;sync;echo 3 > /proc/sys/vm/drop_caches &"
```

7.5.10、标准化 cn 的 postgresql.conf 配置

```
[tbase@VM_0_37_centos shell]$ ./tbase_postgresql_conf_make_cn.sh
```

标准化后，cn postgresql.conf 文件的内容简化如下

```
pgxc_node_name = 'cn001'
port = 11000
pooler_port = 11001
synchronous_commit = local
synchronous_standby_names = "
archive_status_control = 'continue'
include_if_exists='/data/tbase/data3/tbase_01_1/data/cn001/postgresql.conf.user'
```

其它公共参数统一在/postgresql.conf.user 文件中，**注意：配置前请使用 tbase_conf_backup.sh 把配置文件备份**

7.5.11、标准化 dn 的 postgresql.conf 配置

```
[tbase@VM_0_37_centos shell]$ ./tbase_postgresql_conf_make_dn.sh
```

标准化后，dn postgresql.conf 文件的内容简化如下

```
pgxc_node_name = 'dn001'
port = 11002
pooler_port = 11003
```

```
synchronous_commit = local
synchronous_standby_names = "
archive_status_control = 'continue'
include_if_exists='/data/tbase/data1/tbase_01_1/data/dn001/postgresql.conf.user'
```

其它公共参数统一在/postgresql.conf.user 文件中，注意：配置前请使用 **tbase_conf_backup.sh** 把配置文件备份

7.5.12、检查应用程序的 md5 值

```
[tbase@VM_0_37_centos shell]$ ./tbase_md5_check.sh postgres
tbase upgrade begin...
ssh -p 22 -T pgxz@172.16.0.37 "md5sum /data/tbase/data3/tbase_01_5/2.0.7/install/tbase_pgxz/bin/postgres"
d7167771621d018cad21d522971c1c91 /data/tbase/data3/tbase_01_5/2.0.7/install/tbase_pgxz/bin/postgres
ssh -p 22 -T pgxz@172.16.0.37 "md5sum /data/tbase/data1/tbase_01_5/2.0.7/install/tbase_pgxz/bin/postgres"
d7167771621d018cad21d522971c1c91 /data/tbase/data1/tbase_01_5/2.0.7/install/tbase_pgxz/bin/postgres
```

同一个集群，其应用程序的 md5 值必需是一致的，否则就是版本号不一致

7.5.13、返回有节点的 pg_controldata 信息

```
[tbase@VM_0_37_centos shell]$ ./tbase_controldata.sh
pg_controldata cn begin...
ssh -p 22 -T tbase@172.16.0.37 "export PATH=/data/tbase/data3/tbase_01_5/2.0.7/install/tbase_pgxz/bin;export
LD_LIBRARY_PATH=/data/tbase/data3/tbase_01_5/2.0.7/install/tbase_pgxz/lib;/data/tbase/data3/tbase_01_5/2.
0.7/install/tbase_pgxz/bin/pg_controldata -D /data/tbase/data3/tbase_01_5/data/cn001 "
pg_control version number:          1002
Catalog version number:             201707211
Database system identifier:          6612520829144386588
Database cluster state:              in production
pg_control last modified:            Wed 07 Nov 2018 10:54:51 AM CST
Latest checkpoint location:          4/340036B8
Prior checkpoint location:           4/340005F8
Latest checkpoint's REDO location:   4/34003648
Latest checkpoint's REDO WAL file:   0000000100000004000000034
Latest checkpoint's TimeLineID:      1
Latest checkpoint's PrevTimeLineID:  1
Latest checkpoint's full_page_writes: on
Latest checkpoint's NextXID:          0:1940119
Latest checkpoint's NextOID:          204800
Latest checkpoint's NextMultiXactId:  1
Latest checkpoint's NextMultiOffset:  0
Latest checkpoint's oldestXID:        3
Latest checkpoint's oldestXID's DB:   1
Latest checkpoint's oldestActiveXID:  1940115
```

```

Latest checkpoint's oldestMultiXid: 1
Latest checkpoint's oldestMulti's DB: 1
Latest checkpoint's oldestCommitTsXid:3
Latest checkpoint's newestCommitTsXid:3
Latest checkpoint's latestCommitTs:1916627707041
Time of latest checkpoint: Wed 07 Nov 2018 10:54:51 AM CST
Fake LSN counter for unlogged rels: 0/1
Minimum recovery ending location: 0/0
Min recovery ending loc's timeline: 0
Backup start location: 0/0
Backup end location: 0/0
End-of-backup record required: no
wal_level setting: logical
wal_log_hints setting: off
max_connections setting: 2000
max_worker_processes setting: 1024
max_prepared_xacts setting: 20000
max_locks_per_xact setting: 64
track_commit_timestamp setting: on
Maximum data alignment: 8
Database block size: 8192
Blocks per segment of large relation: 131072
WAL block size: 8192
Bytes per WAL segment: 16777216
Maximum length of identifiers: 64
Maximum columns in an index: 32
Maximum size of a TOAST chunk: 1972
Size of a large-object chunk: 2048
Date/time type storage: 64-bit integers
Float4 argument passing: by value
Float8 argument passing: by value
Data page checksum version: 0

```

Mock authentication nonce:

```
c973492121d5fcca02f622b2ab58b60c96850349fdd6b062b010dd8abd6c1fa4
```

```
Need mvcc if page is visible: 0
```

我们可以这样执行返回所有机器的 checkpoint times 时间点

```
[tbase@VM_0_37_centos shell]$ ./tbase_controldata.sh ./tbase_controldata.sh | grep "Time of latest checkpoint"
```

```

Time of latest checkpoint: Wed 07 Nov 2018 10:54:51 AM CST
Time of latest checkpoint: Wed 07 Nov 2018 10:55:23 AM CST
Time of latest checkpoint: Wed 07 Nov 2018 10:54:53 AM CST
Time of latest checkpoint: Wed 07 Nov 2018 10:54:52 AM CST
Time of latest checkpoint: Wed 07 Nov 2018 10:54:52 AM CST
Time of latest checkpoint: Wed 07 Nov 2018 10:54:53 AM CST

```

```
[tbase@VM_0_37_centos shell]$
```

7.5.14、返回所有节点 xlog 文件个数和容量

```
[tbase@VM_0_37_centos shell]$ ./tbase_wal_size_check.sh
```

```
dn master wal size check
```

```
ssh -p 22 -T tbase@172.16.0.37 "du /data/tbase/data1/tbase_01_5/data/dn001/pg_wal/ -h;ls /data/tbase/data1/tbase_01_5/data/dn001/pg_wal/ -l | wc -l"
```

```
4.0K /data/tbase/data1/tbase_01_5/data/dn001/pg_wal/archive_status
```

```
3.6G /data/tbase/data1/tbase_01_5/data/dn001/pg_wal/
```

```
230
```

```
ssh -p 22 -T tbase@172.16.0.42 "du /data/tbase/data2/tbase_01_5/data/dn002/pg_wal/ -h;ls /data/tbase/data2/tbase_01_5/data/dn002/pg_wal/ -l | wc -l"
```

```
4.0K /data/tbase/data2/tbase_01_5/data/dn002/pg_wal/archive_status
```

```
3.6G /data/tbase/data2/tbase_01_5/data/dn002/pg_wal/
```

```
230
```

8、运维技巧及故障处理

8.1、检查配置是否一致

8.1.1、检查 center 是否启用主备自动倒换配置

查询 confdb 的配置参数，修改该参数需要重启 center 服务才能生效

```
[root@VM_0_37_centos ~]# su tbaseConfdb
```

```
[tbaseConfdb@VM_0_37_centos root]$ psql -h 127.0.0.1 -U postgres -d postgres -p 54321
```

```
psql (9.6.2)
```

```
Type "help" for help.
```

```
postgres=# select * from pgxzm_oss.tbl_oss_config_param where conf_item='IS_AUTO_SWITCH';
```

conf_item	conf_val	conf_role	conf_type	tt_create	tt_update
IS_AUTO_SWITCH	false	Master		4 2018-08-06 21:52:10.125231	2018-08-06 21:52:10.125231

```
(1 row)
```

```
postgres=#
```

如果 `conf_val = true` 时手工维护服务器要特别小心，因为主节点一旦故障，center 就会发起主备切换程序，这可能不是你想要的效果。在如果 shell 模式下做运维，最好 center 和 agent 服务给先关掉，等最后修改确认 tbase 服务正常情况下再把 center,agent 拉起来。

8.1.2、检查 agent 服务运行一致性

■ 检查是否启用主备同步智能适用

```
[root@VM_0_37_centos ~]# su tbaseConfdb
[tbaseConfdb@VM_0_37_centos root]$ psql -h 127.0.0.1 -U postgres -d postgres -p 54321
psql (9.6.2)
Type "help" for help.
```

```
postgres=# select * from pgxzm_oss.tbl_oss_config_param where
conf_item='AGT_IS_AUTO_CHECK_TXMODE';
      conf_item      | conf_val | conf_role | conf_type |      tt_create      |      tt_update
-----+-----+-----+-----+-----+-----
 AGT_IS_AUTO_CHECK_TXMODE | false   | Agent     |           | 2018-08-06 21:52:10.131718 | 2018-08-06 21:52:10.131718
(1 row)
```

```
postgres=#
```

如果 `conf_val = true` 时，则表示 `dn` 主备同步会处于智能模式，当同步数据累计超过 64MB 的延迟时主备同步状态会切换为异步。如果你希望不管那种状态下主备同步模式必需一致，则需要把其值配置成 `false`，修改其值需要重启 agent 才能生效

8.1.3、检查机器的资源占用情况

8.1.3.1、硬盘占用情况

```
./tbase_run_command.sh "df -h"
ssh -p 22 -T pgxz@172.16.0.37 "df -h"
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        50G   12G   35G   26% /
devtmpfs         3.9G    0   3.9G    0% /dev
tmpfs            3.9G   60K   3.9G    1% /dev/shm
tmpfs            3.9G  804K   3.9G    1% /run
tmpfs            3.9G    0   3.9G    0% /sys/fs/cgroup
tmpfs            783M    0   783M    0% /run/user/0
/dev/vdb1       200G   12G  189G    6% /data
tmpfs            783M    0   783M    0% /run/user/1002
ssh -p 22 -T pgxz@172.16.0.42 "df -h"
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        50G   13G   35G   27% /
devtmpfs         3.9G    0   3.9G    0% /dev
tmpfs            3.9G   68K   3.9G    1% /dev/shm
tmpfs            3.9G  844K   3.9G    1% /run
tmpfs            3.9G    0   3.9G    0% /sys/fs/cgroup
```

```
tmpfs          783M      0  783M    0% /run/user/0
/dev/vdb1      200G    12G  189G    6% /data
tmpfs          783M      0  783M    0% /run/user/1002
```

8.1.3.2、内存占用情况

```
./tbase_run_command.sh "free -m"
```

```
ssh -p 22 -T pgxz@172.16.0.37 "free -m"
```

	total	used	free	shared	buff/cache	available
Mem:	7822	1666	1206		4376	1459
Swap:	4095	990	3105			

```
ssh -p 22 -T pgxz@172.16.0.42 "free -m"
```

	total	used	free	shared	buff/cache	available
Mem:	7822	811	1427		4487	2191
Swap:	4095	273	3822			

8.1.3.3、系统压力情况

```
./tbase_run_command.sh "w"
```

```
ssh -p 22 -T pgxz@172.16.0.37 "w"
```

```
21:36:29 up 111 days,  7:23,  0 users,  load average: 0.31, 0.69, 0.65
```

```
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
```

```
ssh -p 22 -T pgxz@172.16.0.42 "w"
```

```
21:36:29 up 111 days,  7:24,  1 user,  load average: 0.95, 2.16, 2.50
```

```
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
```

8.1.4、cn 运行参数一致性检查及管理

8.1.4.1、是否禁止删除 xlog

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_cn.sh "show archive_status_control"
```

```
cn001 --- psql -h 172.16.0.37 -p 11000 -d postgres -U tbase -c "show archive_status_control"
```

```
archive_status_control
```

```
-----
```

```
continue
```

```
(1 row)
```

```
cn002 --- psql -h 172.16.0.42 -p 11000 -d postgres -U tbase -c "show archive_status_control"
```

```
archive_status_control
```

```
-----
```

```
continue
```

```
(1 row)
```

如果该值配置为 break 时，则 xlog 会一直累积下去，不会删除，一不小心有可能会把磁盘空间给占满

8.1.4.2、查看动态加载模块

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_cn.sh "show shared_preload_libraries"
cn001 --- psql -h 172.16.0.37 -p 11000 -d postgres -U tbase -c "show shared_preload_libraries"
shared_preload_libraries
-----
pg_stat_statements
(1 row)

cn002 --- psql -h 172.16.0.42 -p 11000 -d postgres -U tbase -c "show shared_preload_libraries"
shared_preload_libraries
-----
pg_stat_statements
(1 row)
```

8.1.4.3、检查参数是否一致

■ 显示单一参数

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_cn.sh "show enable_data_mask"
cn001 --- psql -h 172.16.0.37 -p 11000 -d postgres -U tbase -c "show enable_data_mask"
enable_data_mask
-----
off
(1 row)

cn002 --- psql -h 172.16.0.42 -p 11000 -d postgres -U tbase -c "show enable_data_mask"
enable_data_mask
-----
off
(1 row)
```

■ 显示多个参数

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_cn.sh "select current_setting('max_connections') as
max_connections,current_setting('shared_buffers') as shared_buffers"
cn001 --- psql -h 172.16.0.37 -p 11000 -d postgres -U tbase -c "select current_setting('max_connections') as
max_connections,current_setting('shared_buffers') as shared_buffers"
max_connections | shared_buffers
-----+-----
2000            | 512MB
(1 row)

cn002 --- psql -h 172.16.0.42 -p 11000 -d postgres -U tbase -c "select current_setting('max_connections') as
```

```
max_connections,current_setting('shared_buffers') as shared_buffers"
max_connections | shared_buffers
-----+-----
2000           | 512MB
(1 row)
```

8.1.4.4、检查执行计划是否一致

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_cn.sh "explain select count(1) from t"
cn001 --- psql -h 172.16.0.37 -p 11000 -d postgres -U tbase -c "explain select count(1) from t"
QUERY PLAN
-----
Finalize Aggregate  (cost=26024.85..26024.86 rows=1 width=8)
-> Remote Subquery Scan on all (dn001,dn002,dn003,dn004) (cost=26024.52..26024.84 rows=3 width=8)
-> Gather  (cost=25924.52..25924.83 rows=3 width=8)
    Workers Planned: 3
-> Partial Aggregate  (cost=24924.52..24924.53 rows=1 width=8)
    -> Parallel Seq Scan on t  (cost=0.00..23311.61 rows=645161 width=0)
(6 rows)
```

```
cn002 --- psql -h 172.16.0.42 -p 11000 -d postgres -U tbase -c "explain select count(1) from t"
QUERY PLAN
-----
Finalize Aggregate  (cost=121.52..121.53 rows=1 width=8)
-> Remote Subquery Scan on all (dn001,dn002,dn003,dn004) (cost=121.50..121.52 rows=1 width=8)
-> Partial Aggregate  (cost=21.50..21.51 rows=1 width=8)
    -> Seq Scan on t  (cost=0.00..19.20 rows=920 width=0)
(4 rows)
```

上面显示 cn002 的统计信息不正确，需要执行 vacuum analyze。

8.1.4.5、检查表定义是否一致

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_cn.sh "\d+ public.t"
cn001 --- psql -h 172.16.0.37 -p 11000 -d postgres -U tbase -c "\d+ public.t"
Table "public.t"
Column | Type   | Collation | Nullable | Default | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----+-----+-----
id      | integer |           |          |         | plain   |              |
mc      | text    |           |          |         | extended |              |
Distribute By: SHARD(id)
Location Nodes: ALL DATANODES
```

```
cn002 --- psql -h 172.16.0.42 -p 11000 -d postgres -U tbase -c "\d+ public.t"
Table "public.t"
```

Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
id	integer				plain		
mc	text				extended		

Distribute By: SHARD(id)

Location Nodes: ALL DATANODES

[tbase@VM_0_37_centos shell]\$

8.1.4.6、检查 pgxc_node 配置是否一致

[tbase@VM_0_37_centos shell]\$./tbase_run_sql_cn.sh "select * from pgxc_node"

cn001 --- psql -h 172.16.0.37 -p 11000 -d postgres -U tbase -c "select * from pgxc_node"

node_name	node_type	node_port	node_host	nodeis_primary	nodeis_preferred	node_id	node_cluster_name
agtm_0	G	11006	172.16.0.37	t	f	475343005	tbase_cluster
cn001	C	11000	172.16.0.37	f	f	-264077367	tbase_cluster
cn002	C	11000	172.16.0.42	f	f	-674870440	tbase_cluster
dn002	D	11004	172.16.0.42	f	f	-17499968	tbase_cluster
dn001	D	11002	172.16.0.37	f	f	2142761564	tbase_cluster

(5 rows)

cn002 --- psql -h 172.16.0.42 -p 11000 -d postgres -U tbase -c "select * from pgxc_node"

node_name	node_type	node_port	node_host	nodeis_primary	nodeis_preferred	node_id	node_cluster_name
agtm_0	G	11006	172.16.0.37	t	f	475343005	tbase_cluster
cn001	C	11000	172.16.0.37	f	f	-264077367	tbase_cluster
cn002	C	11000	172.16.0.42	f	f	-674870440	tbase_cluster
dn002	D	11004	172.16.0.42	f	f	-17499968	tbase_cluster
dn001	D	11002	172.16.0.37	f	f	2142761564	tbase_cluster

(5 rows)

8.1.4.7、向所有 cn 发出 pgxc_pool_reload()

[tbase@VM_0_37_centos shell]\$./tbase_run_sql_cn.sh "select pgxc_pool_reload()"

cn001 --- psql -h 172.16.0.37 -p 11000 -d postgres -U tbase -c "select pgxc_pool_reload()"

pgxc_pool_reload

t

(1 row)

cn002 --- psql -h 172.16.0.42 -p 11000 -d postgres -U tbase -c "select pgxc_pool_reload()"

pgxc_pool_reload

t

(1 row)

如果你手动修改了多个 cn 的 pgxc_node 配置，则可以通过上面的语句实现多个 cn 同时配置生效

8.1.5、dn 运行参数检查及管理

8.1.5.1、检查执行计划是否一致

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_dn_master.sh "explain select count(1) from t"
dn001 --- psql -h 172.16.0.37 -p 11002 -d postgres -U tbase -c "explain select count(1) from t"
```

QUERY PLAN

```
-----
Finalize Aggregate  (cost=7824.83..7824.84 rows=1 width=8)
->  Gather  (cost=7824.61..7824.82 rows=2 width=8)
      Workers Planned: 2
      ->  Partial Aggregate  (cost=6824.61..6824.62 rows=1 width=8)
            ->  Parallel Seq Scan on t  (cost=0.00..6303.69 rows=208369 width=0)
```

(5 rows)

```
dn002 --- psql -h 172.16.0.42 -p 11004 -d postgres -U tbase -c "explain select count(1) from t"
```

QUERY PLAN

```
-----
Finalize Aggregate  (cost=7794.50..7794.51 rows=1 width=8)
->  Gather  (cost=7794.28..7794.49 rows=2 width=8)
      Workers Planned: 2
      ->  Partial Aggregate  (cost=6794.28..6794.29 rows=1 width=8)
            ->  Parallel Seq Scan on t  (cost=0.00..6275.43 rows=207542 width=0)
```

(5 rows)

这个很重要，执行计划如果不一致，则集群就会存在短板

8.1.5.2、检查是否有备机存在

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_dn_master.sh "select client_addr, sync_state from pg_stat_replication"
```

```
dn001 --- psql -h 172.16.0.37 -p 11002 -d postgres -U tbase -c "select client_addr, sync_state from pg_stat_replication"
```

client_addr | sync_state

-----+-----

172.16.0.42 | async

(1 row)

```
dn002 --- psql -h 172.16.0.42 -p 11004 -d postgres -U tbase -c "select client_addr, sync_state from pg_stat_replication"
```

```

client_addr | sync_state
-----+-----
172.16.0.37 | async
(1 row)

```

sync_state 显示了主备之前同步的级别，sync 值为同步，async 值为异步

8.1.5.3、检查表的大小

■ 容量大小

```

[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_dn_master.sh "\dt+"
dn001 --- psql -h 172.16.0.37 -p 11002 -d postgres -U tbase -c "\dt+"
                List of relations
 Schema | Name | Type  | Owner | Size  | Allocated Size | Description
-----+-----+-----+-----+-----+-----+-----
 public | t    | table | tbase | 33 MB | 33 MB           |
(1 row)

```

```

dn002 --- psql -h 172.16.0.42 -p 11004 -d postgres -U tbase -c "\dt+"
                List of relations
 Schema | Name | Type  | Owner | Size  | Allocated Size | Description
-----+-----+-----+-----+-----+-----+-----
 public | t    | table | tbase | 33 MB | 33 MB           |
(1 row)

```

■ 记录数大小

```

[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_dn_master.sh "select count(1) from t_time_range"
psql -h 172.16.0.29 -p 23001 -d postgres -U pgxz -c "select count(1) from t_time_range"
count
-----
11740603
(1 row)

```

```

psql -h 172.16.0.29 -p 23002 -d postgres -U pgxz -c "select count(1) from t_time_range"
count
-----
11864540
(1 row)

```

■ 通过系统表快速确认记录数大约量

```

[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_dn_master.sh "select schemaname,relname,n_live_tup from
pg_stat_all_tables where relname='t_time_range'"

```

```
dn001 --- psql -h 172.16.0.37 -p 11002 -d postgres -U tbase -c "select schemaname,relname,n_live_tup from
pg_stat_all_tables where relname='t'"
schemaname | relname | n_live_tup
-----+-----+-----
public      | t          |      500086
(1 row)
```

```
dn002 --- psql -h 172.16.0.42 -p 11004 -d postgres -U tbase -c "select schemaname,relname,n_live_tup from
pg_stat_all_tables where relname='t'"
schemaname | relname | n_live_tup
-----+-----+-----
public      | t          |      498102
(1 row)
```

通过表的大小可以确认数据是否有倾斜

8.1.5.4、比较各个 dn 执行相同语句的时间

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_dn_master.sh "explain analyze select * from t"
dn001 --- psql -h 172.16.0.37 -p 11002 -d postgres -U tbase -c "explain analyze select * from t"
QUERY PLAN
-----
Seq Scan on t (cost=0.00..9220.86 rows=500086 width=10) (actual time=0.067..204.963 rows=500086
loops=1)
Planning time: 0.434 ms
Execution time: 304.276 ms
(3 rows)
```

```
dn002 --- psql -h 172.16.0.42 -p 11004 -d postgres -U tbase -c "explain analyze select * from t"
QUERY PLAN
-----
Seq Scan on t (cost=0.00..9181.02 rows=498102 width=10) (actual time=0.058..213.873 rows=498102
loops=1)
Planning time: 0.376 ms
Execution time: 319.381 ms
(3 rows)
```

8.1.6、检查 dn 备运行参数

8.1.6.1、检查备机角色是否正确

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_dn_slave.sh "select pg_is_in_recovery()"
psql -h 172.16.0.37 -p 11004 -d postgres -U tbase -c "select pg_is_in_recovery()"
pg_is_in_recovery
-----
```

```
t
(1 row)
```

```
psql -h 172.16.0.42 -p 11002 -d postgres -U tbase -c "select pg_is_in_recovery()
pg_is_in_recovery
```

```
-----
t
(1 row)
```

select pg_is_in_recovery()返回的值如果为 true,表示该节点为备节点

8.2、故障处理

8.2.1、节点服务启动或者停止失败排查方法

从监控平台上或者脚本操作节点服务失败的排查方法

8.2.1.1、检查磁盘空间是否足够

```
[pgxz@VM_0_29_centos ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        50G   13G   35G   27% /
devtmpfs         3.9G     0  3.9G    0% /dev
tmpfs            3.9G   68K   3.9G    1% /dev/shm
tmpfs            3.9G  844K   3.9G    1% /run
tmpfs            3.9G     0  3.9G    0% /sys/fs/cgroup
tmpfs            783M     0  783M    0% /run/user/0
/dev/vdb1        200G   15G  186G    8% /data
tmpfs            783M     0  783M    0% /run/user/1002
```

上面的 Avail 表示还有多少空间可用，如果空间不够的话会影响 tbase 服务的运行

8.2.1.2、把 agent 和 center 服务关闭

执行/data/tbase/install/tbase_oss/tools/op/stop.sh 关闭 agent 服务，也可以执行 ./agent_stop.sh 把所有机器的 agent 服务先临时关掉，这是因为 agent 会不断的拉起 tbase 有故障的服务节点，会给你手工运维带来麻烦

center 如果配置了主备自动倒换，那么手工运维 tbase 服务时需要把 center 服务也关掉，防止自动主备切换。执行./center_stop.sh 把所有 center 服务先临时关掉，否则会给运维带来麻烦

8.2.1.3、查看故障节点目录中 pg_log 目录最后一个日志文件日志内容

输入 ll -rt | tail -n 10，最示在最后一个文件就是最新的日志文件，使用 vim 打开，按 shift + g 移到文件

内容最后面，按日志提示出错解决问题即可

各种错误处理方法请参考错误码文档

8.2.1.4、服务恢复后把 agent,center 服务启动

```
./center_statrt.sh
```

```
./agent_start.sh
```

8.2.2、取消掉某些查询方法

有时应用写错了 sql 语句，或者程序 bug 引发大量不需要的查询并发执行，占用了大量系统资源，这时需要 dba 使用工具取消掉这些正在执行的查询

8.2.2.1、查询要取消进程 pid

```
postgres=# select * from pg_stat_activity where datid is not null and pid!=pg_backend_pid() limit 1;
```

```
-[ RECORD 1 ]-----+-----
datid          | 13325
datname        | postgres
pid            | 3981
usesysid       | 10
username       | tbase
application_name | psql
client_addr    | 127.0.0.1
client_hostname |
client_port    | 56254
backend_start   | 2018-08-14 19:33:12.235117+08
xact_start     |
query_start    | 2018-08-14 19:33:26.458043+08
state_change    | 2018-08-14 19:33:26.60002+08
wait_event_type | Client
wait_event      | ClientRead
state          | idle
backend_xid     |
backend_xmin    |
query          | select count(1) from t;
backend_type    | client backend
```

8.2.2.2、取消某个节点上面的查询

```
postgres=# select pg_cancel_backend(3981)
```

8.2.2.3、取消所有 cn 节点上面的查询

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_cn.sh "select pg_cancel_backend(pid) from pg_stat_activity
where query like '%update t set mc%' and pid!=pg_backend_pid()"
```

你可以先执行

```
./tbase_run_sql_cn.sh "select pid,query from pg_stat_activity where query like '%update t set mc%' and
pid!=pg_backend_pid()"
```

确认要取消进程是否正确

8.2.2.4、取消所有 dn 节点上面的查询

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_dn_master.sh "select pg_cancel_backend(pid) from
pg_stat_activity where query like '%update t set mc%' and pid!=pg_backend_pid()"
```

你可以先执行

```
./tbase_run_sql_dn_master.sh "select pid,query from pg_stat_activity where query like '%update t set mc%' and
pid!=pg_backend_pid()"
```

确认要取消进程是否正确

使用执行 `pg_cancel_backend()` 有时进程不会响应,这时需要使用 `pg_terminate_backend()` 来杀死某个进程,见 8.2.3

8.2.3、kill 掉某些进程方法

8.2.3.1、查到要 kill 掉进程 pid

```
postgres=# select * from pg_stat_activity where datid is not null and pid!=pg_backend_pid() limit 1;
```

```
-[ RECORD 1 ]-----+-----
```

datid	13325
datname	postgres
pid	3981
usesysid	10
username	tbase
application_name	psql
client_addr	127.0.0.1
client_hostname	
client_port	56254
backend_start	2018-08-14 19:33:12.235117+08
xact_start	

```

query_start      | 2018-08-14 19:33:26.458043+08
state_change     | 2018-08-14 19:33:26.60002+08
wait_event_type  | Client
wait_event       | ClientRead
state            | idle
backend_xid      |
backend_xmin     |
query            | select count(1) from t;
backend_type     | client backend

```

8.2.3.2、kill 某个节点上面的查询

```
postgres=# select pg_terminate_backend(3981)
```

8.2.3.3、kill 所有 cn 节点上面的查询

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_cn.sh "select pg_terminate_backend(pid) from pg_stat_activity where query like '%update t set mc%' and pid!=pg_backend_pid()"
```

你可以先执行

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_cn.sh "select pid,query from pg_stat_activity where query like '%update t set mc%' and pid!=pg_backend_pid()"
```

确认要 kill 进程是否正确

8.2.3.3、kill 所有 dn 节点上面的查询

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_dn_master.sh "select pg_terminate_backend(pid) from pg_stat_activity where query like '%update t set mc%' and pid!=pg_backend_pid()"
```

你可以先执行

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_dn_master.sh "select pid,query from pg_stat_activity where query like '%update t set mc%' and pid!=pg_backend_pid()"
```

确认要 kill 进程是否正确

8.2.4、执行 ddl 或者更新数据卡住排查

8.2.4.1、检查 dn 主备节点的复制同步级别

使用下面语句检查主备同步模式

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_dn_master.sh "select current_setting('synchronous_commit') as
synchronous_commit,current_setting('synchronous_standby_names') as
synchronous_standby_names,array((select sync_state from pg_stat_replication )) as sync_state"
```

```

synchronous_commit | synchronous_standby_names | sync_state
-----+-----+-----
local              |                               | {async}
(1 row)
```

如果返回值为 `synchronous_commit = on`、`synchronous_standby_names` 不为空则表示主备为同步复制模式，这时 `sync_state` 值必需为 `sync`，否则无法执行 `ddl` 或者更新的 `dml` 操作

解决问题方法

如果 `sync_state` 值为空则表示备机服务没拉起来，把备机服务拉起来即可

或者

- 1、按 8.1.2 配置主备复制为某智能模式
- 2、把 `synchronous_commit` 值配置为 `local`
- 3、reload dn 主服务

8.2.4.2、检查 cn 或者 dn 节点是否有 2pc 残留

使用下面语句检查 cn 上是否有 2pc 存在

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_cn.sh "select gid from pg_catalog.pg_prepared_xacts"
psql -h 172.16.0.37 -p 15432 -d postgres -U pgxz -c "select gid from pg_catalog.pg_prepared_xacts"
gid
-----
tbase_2pc
(1 row)
```

```
psql -h 172.16.0.42 -p 15432 -d postgres -U pgxz -c "select gid from pg_catalog.pg_prepared_xacts"
gid
-----
(0 rows)
```

使用下面语句检查 dn 上是否有 2pc 存在

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_dn_master.sh "select gid from pg_catalog.pg_prepared_xacts"
psql -h 172.16.0.37 -p 23001 -d postgres -U pgxz -c "select gid from pg_catalog.pg_prepared_xacts"
gid
-----
(0 rows)
```

```
psql -h 172.16.0.42 -p 23002 -d postgres -U pgxz -c "select gid from pg_catalog.pg_prepared_xacts"
gid
-----
tbase_2pc
(1 row)
```

解决问题方法

commit 或者 rollback 影响的 2pc 残留，清理 cn 2pc 残留方法

```
rollback cn 2pc : ./tbase_cn_2pc_rollback.sh
commit cn 2pc : ./tbase_cn_2pc_commit.sh
```

```
rollback dn 2pc : ./tbase_dn_2pc_rollback.sh
commit dn 2pc : ./tbase_dn_2pc_commit.sh
```

8.2.4.3、排它锁影响

使用下面语句查询各个 cn 或者 dn 节点上面是否运行相关锁语句

检查 cn 的语句

```
[tbase@VM_0_37_centos shell]$. /tbase_run_sql_cn.sh "select * from pg_stat_activity where query ilike '%t_time_range%' and pid!=pg_backend_pid()"
```

检查 dn 的语句

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_dn_master.sh "select * from pg_stat_activity where query ilike '%t_time_range%' and pid!=pg_backend_pid()"
```

上面的 t_time_range 为资源关键字，可以是表名，索引名。。。

解决问题方法

使用 pg_cancel_backend 函数把查询取消掉

在 cn 上执行

```
[tbase@VM_0_37_centos shell]$. /tbase_run_sql_cn.sh "select pg_cancel_backend(pid) from pg_stat_activity where query like '%t_time_range%' and pid!=pg_backend_pid()"
```

在 dn 上面执行

```
[tbase@VM_0_37_centos shell]$. /tbase_run_sql_dn_master.sh "select pg_cancel_backend(pid) from pg_stat_activity where query like '%t_time_range%' and pid!=pg_backend_pid()"
```

使用 pg_terminate_backend 函数把进程 kill 掉

在 cn 上执行

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_cn.sh "select pg_terminate_backend(pid) from pg_stat_activity where query like '%t_time_range%' and pid!=pg_backend_pid()"
```

在 dn 上面执行

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_dn_master.sh "select pg_terminate_backend(pid) from pg_stat_activity where query like '%t_time_range%' and pid!=pg_backend_pid()"
```

8.3、高级运维技术

8.3.1、配置共享 postgresql.conf 运行参数

做为集群，我们希望每个 cn 或者 dn 节点中运行的参数保持一致，由于 postgresql.conf 中存在如 port,pgxc_node_name 这样与节点关联性强的参数，每次修改 postgresql.conf 中的参数后无法直接覆盖，下面介绍如何分离共享参数方法

8.3.1.1、建立一个 postgresql.conf.user 共享参数文件

```
[tbase@VM_0_37_centos shell]$ vim /data/tbase/shell/postgresql.conf.user
```

```
#参数值按实际环境填写
listen_addresses = '0.0.0.0'
max_connections = 2000
max_pool_size = 65535
superuser_reserved_connections = 10
tcp_keepalives_idle = 60
tcp_keepalives_interval = 10
tcp_keepalives_count = 10
shared_buffers = 512MB
max_prepared_transactions = 20000
work_mem = 4MB
maintenance_work_mem = 256MB
wal_level = replica
max_wal_senders = 64
hot_standby = on
max_standby_archive_delay = 1800s
max_standby_streaming_delay = 1800s
wal_keep_segments = 4096
full_page_writes = on
```

```
archive_mode = off
archive_timeout = 1800
archive_command = 'echo 0'
wal_buffers = 16MB
bgwriter_lru_maxpages = 500
bgwriter_lru_multiplier = 2.0
bgwriter_delay = 10ms
seq_page_cost = 1.0
random_page_cost = 2.0
checkpoint_timeout = 5min
max_wal_size = 1GB
min_wal_size = 256MB
checkpoint_completion_target = 0.93
checkpoint_warning = 30s
effective_cache_size = 6GB
log_destination = 'csvlog'
logging_collector = on
log_directory = 'log'
log_rotation_age = 120
log_rotation_size = 1024MB
track_activity_query_size = 4096
log_min_duration_statement = 10000
log_checkpoints = off
log_connections = off
log_disconnections = off
log_duration = off
log_line_prefix = '%m %a %r %d %u %p %x'
log_statement = 'none'
log_timezone = 'PRC'
log_filename = 'postgresql-%A-%H.log'
track_counts = on
autovacuum = on
autovacuum_max_workers = 1
autovacuum_vacuum_cost_delay = 20ms
autovacuum_vacuum_scale_factor = 0.0002
autovacuum_analyze_scale_factor = 0.0001
vacuum_cost_delay = 0
vacuum_cost_limit = 10000
log_lock_waits = on
deadlock_timeout = 1s
datestyle = 'iso, ymd'
timezone = 'PRC'
#enable_mergejoin = off
#enable_nestloop = off
#enable_indexonlyscan = off
```

#下面同几个参数请查看一键安装后的参数一般为 zh_CN.UTF-8 或 en_US.utf8 或 C

```

lc_messages = 'en_US.utf8'
lc_monetary = 'en_US.utf8'
lc_numeric = 'en_US.utf8'
lc_time = 'en_US.utf8'
default_text_search_config = 'pg_catalog.simple'
#parallel_user = 'gdt_new'
shared_preload_libraries = 'pg_stat_statements'
pg_stat_statements.max = 1000
pg_stat_statements.track = all
#enable_key_value = off
#base_backup_limit=1000
#manual_hot_date = '2017-12-01'
enable_oracle_compatible = off
enable_pooler_stuck_exit = off
#启用密码复杂度检查，默认是关闭
#密码规则：必须同时包含 大写字母、小写字母、特殊字符、数字。且长度>=8 位
#enable_check_password = off
#开启/关闭帐号锁定功能，默认关闭，该功能最好配合 pg_hba.conf 一起使用
#enable_lock_account = off
#开启/关闭打印，默认关闭
#lock_account_print = off
#帐号锁定时间，单位 s，默认 600s
#account_lock_time = 600
#连续失败 10 次，锁定帐号，默认 10 次
#account_lock_threshold = 10
#最多支持锁定 1000 个帐号，默认 1000，此参数修改后需要重启生效
#account_lock_track_count = 1000
enable_data_mask = off
enable_cls = off
enable_transparent_crypt = off
enable_audit = off
enable_fga = off
log_statement = 'none'
consumer_connect_timeout = 10
debug_data_pump = off
vacuum_delta = 1000
enable_distri_debug_print = off
max_replication_slots=64
max_wal_senders=64
max_logical_replication_workers=64
max_sync_workers_per_subscription=4
max_worker_processes = 1024
max_parallel_workers = 1024
enable_statistic = off
replication_level = 0
prefer_olap = off

```

```
olap_optimizer = on
synchronous_commit = local
synchronous_standby_names = "
```

8.3.1.2、分发 postgresql.conf.user 文件

```
[tbase@VM_0_37_centos shell]$ ./tbase_pg_hba_scp.sh /data/tbase/shell/postgresql.conf.user
```

8.3.1.2、cn 引用 postgresql.conf.user 配置

```
[tbase@VM_0_37_centos shell]$ ./tbase_postgresql_conf_make_cn.sh
```

8.3.1.3、dn 引用 postgresql.conf.user 配置

```
[tbase@VM_0_37_centos shell]$ ./tbase_postgresql_conf_make_dn.sh
```

8.3.2、同步配置文件方法

8.3.2.1、同步 postgresql.conf.user 文件

```
[tbase@VM_0_37_centos shell]$ ./tbase_pg_hba_scp.sh /data/tbase/shell/postgresql.conf.user
```

8.3.2.2、同步 pg_hba.conf 文件

```
[tbase@VM_0_37_centos shell]$ ./tbase_pg_hba_scp.sh /data/tbase/shell/pg_hba.conf
```

8.4、多种级别运行参数配置

8.4.1、配置数据库级别运行参数

8.4.1.1、配置连接某个库后默认搜索路径

默认情况下，一个库的搜索路径为

```
postgres=# show search_path ;
search_path
-----
"$user",public
(1 row)
```

没有在搜索路径下的资源访问时需要写全路径，否则会出错

```
postgres=# select * from order_main;
ERROR:  relation "order_main" does not exist
LINE 1: select * from order_main;
                        ^
```

```
postgres=# select * from pgxc.order_main;
 id | dh
----+----
(0 rows)
```

配置搜索路径后即可实现非全路径访问

```
postgres=# alter database postgres set search_path="$user", public, pgxc,pg_catalog;
ALTER DATABASE
postgres=# \q
[pgxz@VM_0_3_centos root]$ psql
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=# select * from order_main;
 id | dh
----+----
(0 rows)
```

```
postgres=# show search_path;
      search_path
-----
 "$user", public, pgxc, pg_catalog
(1 row)
```

还原默认配置方法

```
postgres=# alter database postgres set search_path to DEFAULT ;
ALTER DATABASE
```

8.4.1.2、配置连接某个库使用的内存

■ 可使用的排序内存

```
postgres=# alter database postgres set work_mem = '8MB';
ALTER DATABASE
```

■ 可使用的维护内存

```
postgres=# alter database postgres set maintenance_work_mem TO '256MB';
ALTER DATABASE
```

8.4.1.3、配置连接某个库后使用的时区

```
postgres=# alter database postgres set TimeZone to cet;  
ALTER DATABASE
```

timestamp with time zone 类型对于本地化显示非常有帮助

8.4.1.4、配置连接某个库后使用的日期格式

```
postgres=# select current_date;  
date
```

```
-----  
2018-03-29  
(1 row)
```

```
postgres=# alter database postgres set DateStyle to SQL, DMY;  
ALTER DATABASE
```

```
postgres=# \q  
[pgxz@VM_0_3_centos root]$ psql  
psql (PostgreSQL 10 (TBase 2.01))  
Type "help" for help.
```

```
postgres=# select current_date;  
date
```

```
-----  
29/03/2018  
(1 row)
```

8.4.1.5、配置连接某个库后执行语句最多时长

```
postgres=# alter database postgres set statement_timeout =1000;  
ALTER DATABASE  
postgres=#
```

执行 1 秒超时

8.4.1.6、配置连接某个库后默认的客户端编码

```
postgres=# alter database postgres set client_encoding to gbk;  
ALTER DATABASE
```

配置客户端编码为 gbk，适用于数据库编码为 utf8，应用程序编码为 gbk 的应用

8.4.1.7、配置某个数据库最大连接数

```
postgres=# alter database postgres CONNECTION LIMIT 1;
ALTER DATABASE
```

该限制只对普通用户有影响

```
[pgxz@VM_0_3_centos root]$ psql -U pgxc
psql: FATAL:  too many connections for database "postgres"
```

取消限制

```
postgres=# alter database postgres CONNECTION LIMIT -1;
ALTER DATABASE
```

注意：该参数配置没使用\drds 无法查到，限制最大连接数只对普通用户起作用

下面语句可以查询数据库的连接数限制

```
select datname,datconlimit from pg_database;
```

8.4.1.8、配置某个库使用日志记录级别

```
postgres=# alter database postgres set log_statement=none;
ALTER DATABASE
```

这样对这个数据库的访问就不记录日志了

8.4.1.9、配置连接某个库后的 wal 日志写盘级别

```
postgres=# alter database postgres set synchronous_commit to local;
ALTER DATABASE
```

该库的更新操作只要求本地提交

8.4.1.10、配置连接某个库后禁用某个规划器

```
postgres=# alter database postgres set enable_indexonlyscan to off;
ALTER DATABASE
```

禁用 indexonlyscan 扫描

8.4.1.11、配置连接某个库后事务默认为只读事务

```
postgres=# alter database postgres set default_transaction_read_only to on;
ALTER DATABASE
```

不过 session 中可以通过这样来实现可写

```
postgres=> begin read write;
BEGIN
postgres=> create table t2(id int);
CREATE TABLE
postgres=> commit;
COMMIT
或者
postgres=> set default_transaction_read_only to off;
```

取消方法

```
postgres=# set default_transaction_read_only to off;
SET
postgres=# alter database postgres set default_transaction_read_only to off;
ALTER DATABASE
postgres=#
```

8.4.1.12、配置某库建表是否需要 oid 列

```
postgres=# alter database postgres set default_with_oids to on;
ALTER DATABASE
postgres=# \q
[pgxz@VM_0_3_centos root]$ psql
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=# create table t5(id int);
CREATE TABLE
postgres=# select oid,* from t5;
 oid | id
-----+----
(0 rows)
```

默认加了 oid 列，建立表可以加 without oids 指定该表不需要 oid 列

8.4.1.13、配置连接某个库后反斜线是否当作普通字符使用

```
postgres=# alter database postgres set standard_conforming_strings to on;
ALTER DATABASE
postgres=# \q
[pgxz@VM_0_3_centos root]$ psql
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=# select '\\';
?column?
-----
\\
(1 row)
```

如不当普通字符使用，则为转义符，而且使用时会警告

```
postgres=# set standard_conforming_strings to off;
SET
postgres=# select '\\';
WARNING:  nonstandard use of \\ in a string literal
LINE 1: select '\\';
           ^
HINT:  Use the escape string syntax for backslashes, e.g., E'\\'.
?column?
```

```
-----
\
(1 row)
```

```
postgres=# select E'\\';
?column?
-----
\
(1 row)
```

8.4.1.14、配置连接某个库后字符串存在反斜线时是否发出警告

```
postgres=# alter database postgres set escape_string_warning to on;
ALTER DATABASE
```

该参数要在 `standard_conforming_strings` 为 off 时才有效

8.4.1.15、配置连接某个库后 `expr = NULL` 当成 `expr is null` 处理

```
postgres=# alter database postgres set transform_null_equals to on;
ALTER DATABASE
postgres=# \q
[pgxz@VM_0_3_centos root]$ psql
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=# select * from tbase where mc=null;
 id | mc
----+----
  2 |
```

(1 row)

```
postgres=# set transform_null_equals to off;
SET
postgres=# select * from tbase where mc=null;
 id | mc
----+----
(0 rows)
```

如果打开, 那么表达式 `expr = NULL` 或者 `NULL = expr` 将被当做 `expr IS NULL` 处理, 也就是说, 如果 `expr` 得出 `NULL` 值则返回真, 否则返回假。正确的 SQL 标准兼容的 `expr = NULL` 行为总是返回 `NULL`(未知)

8.4.1.16、配置连接某个库后出错时中断连接

```
postgres=# alter database postgres set exit_on_error to on;
ALTER DATABASE
postgres=# \q
[pgxz@VM_0_3_centos ~]$ psql
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=# select pg_backend_pid();
 pg_backend_pid
-----
          29226
(1 row)
```

```
postgres=# select * from t1;
FATAL:  relation "t1" does not exist
LINE 1: select * from t1;
```

```
^
FATAL:  relation "t1" does not exist
LINE 1: select * from t1;
```

The connection to the server was lost. Attempting reset: Succeeded.

```
postgres=# select pg_backend_pid();
 pg_backend_pid
-----
          29253
(1 row)
```

如果为真, 那么任何错误都将终止当前连接。

8.4.1.17、查询数据库的个性化配置

```
postgres=# \drds
```

List of settings

Role	Database	Settings
pgxz		default_with_oids=on
		standard_conforming_strings=on
		transform_null_equals=on
postgres		log_statement=none

(2 rows)

```
postgres=# select * from pg_db_role_setting where setdatabase!=0;
```

setdatabase	setrole	setconfig
65537	0	{log_statement=none}

(1 row)

8.4.1.18、取消某个数据库的个性化配置

```
postgres=# alter database postgres reset all;
ALTER DATABASE
```

8.4.2、配置用户级别运行参数

8.4.2.1、配置某个用户连接后默认搜索路径

```
postgres=# alter role pgxz set search_path = "$user", public, pgxc;
```

```
ALTER ROLE
```

```
[pgxz@VM_0_3_centos ~]$ psql
```

```
psql (PostgreSQL 10 (TBase 2.01))
```

```
Type "help" for help.
```

```
postgres=# show search_path ;
```

```
search_path
```

```
-----
"$user", public, pgxc
```

(1 row)

还原默认配置方法

```
postgres=# alter role pgxz set search_path to default;
```

```
ALTER ROLE
```

注意：用户的配置参数会覆盖掉数据库的配置参数，如下所示

```
postgres=# alter role pgxz set search_path = "$user", public, pgxc;
ALTER ROLE
postgres=# alter database postgres set search_path to "$user", public, pgxc,pg_catalog;
ALTER DATABASE
postgres=# \q
[pgxz@VM_0_3_centos ~]$ psql
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.

postgres=# show search_path ;
      search_path
-----
"$user", public, pgxc
(1 row)
```

8.4.2.2、配置某个用户连接后使用的内存

■ 可使用的排序内存

```
postgres=# alter role pgxz set work_mem = '8MB';
ALTER ROLE
```

■ 可使用的维护内存

```
postgres=# alter role pgxz set maintenance_work_mem TO '256MB';
ALTER ROLE
```

8.4.2.3、配置某个用户连接后使用的时区

```
postgres=# alter role pgxz set TimeZone to cet;
ALTER ROLE
```

timestamp with time zone 类型对于本地化显示非常有帮助

8.4.2.4、配置某个用户连接后使用的日期格式

```
postgres=# select current_date;
      date
-----
2018-03-29
(1 row)
```

```
postgres=# alter role pgxz set DateStyle to SQL, DMY;
ALTER ROLE
postgres=# \q
[pgxz@VM_0_3_centos root]$ psql
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=# select current_date;
      date
-----
 29/03/2018
(1 row)
```

8.4.2.5、配置某个用户连接后执行语句最多时长

```
postgres=# alter role pgxz set statement_timeout = 1000;
ALTER ROLE
postgres=#
```

执行 1 秒超时

8.4.2.6、配置某个用户连接后默认的客户端编码

```
postgres=# alter role pgxz set client_encoding to gbk;
ALTER ROLE
```

配置客户端编码为 gbk，适用于数据库编码为 utf8，应用程序编码为 gbk 的应用

8.4.2.7、配置某个用户最大连接数

```
postgres=# alter role pgxc CONNECTION LIMIT 1;
ALTER ROLE
```

取消限制

```
postgres=# alter role pgxc CONNECTION LIMIT -1;
ALTER ROLE
```

注意：该参数配置没使用 \drds 无法查到，限制连接数只对普通用户起作用

下面语句可以查询用户的连接数限制

```
select rolname,rolconnlimit from pg_roles ;
```

8.4.2.8、配置某个用户的日志记录级别

```
postgres=# alter role pgxc set log_statement=none;
```

ALTER ROLE

postgres=# \q

[pgxz@VM_0_3_centos ~]\$ psql -U pgxc

psql (PostgreSQL 10 (TBase 2.01))

Type "help" for help.

postgres=> show log_statement;

log_statement

none

(1 row)

postgres=> \q

[pgxz@VM_0_3_centos ~]\$ psql

psql (PostgreSQL 10 (TBase 2.01))

Type "help" for help.

postgres=# show log_statement;

log_statement

all

(1 row)

这样对这个用户的访问就不记录日志了，普通用户无法通过 session 改变日志级别

8.4.2.9、配置某个用户连接后默认 wal 日志写盘级别

postgres=# alter role pgxc set synchronous_commit to local;

ALTER ROLE

该用户更新操作只要求本地提交

8.4.2.10、配置某个用户连接后禁用某个规划器

postgres=# alter role pgxz set enable_indexonlyscan to off;

ALTER ROLE

禁用 indexonlyscan 扫描

8.4.2.11、配置某个用户连接后事务默认为只读事务

postgres=# alter role pgxz set default_transaction_read_only to on;

ALTER ROLE

不过 session 中可以通过这样来实现可写

```
postgres=> begin read write;
BEGIN
postgres=> create table t2(id int);
CREATE TABLE
postgres=> commit;
COMMIT
或者
postgres=> set default_transaction_read_only to off;
```

取消方法

```
postgres=# set default_transaction_read_only to off;
SET
postgres=# alter role pgxz set default_transaction_read_only to off;
ALTER ROLE
postgres=#
```

8.4.2.12、配置某用户建表是否需要 oid 列

```
postgres=# alter role pgxz set default_with_oids to on;
ALTER ROLE
postgres=# \q
[pgxz@VM_0_3_centos root]$ psql
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=# create table t5(id int);
CREATE TABLE
postgres=# select oid,* from t5;
 oid | id
-----+----
(0 rows)
```

默认加了 oid 列

8.4.2.13、配置某个用户连接后反斜线是否当作普通字符使用

```
postgres=# alter role pgxz set standard_conforming_strings to on;
ALTER ROLE
postgres=# \q
[pgxz@VM_0_3_centos root]$ psql
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=# select '\\';
```

```
?column?
```

```
-----
\\
(1 row)
```

如不当普通字符使用，则为转义符，而且使用时会警告

```
postgres=# set standard_conforming_strings to off;
```

```
SET
```

```
postgres=# select '\\';
```

```
WARNING:  nonstandard use of \\ in a string literal
```

```
LINE 1: select '\\';
```

```
^
```

```
HINT:  Use the escape string syntax for backslashes, e.g., E'\\'.
```

```
?column?
```

```
-----
\
(1 row)
```

```
postgres=# select E'\\';
```

```
?column?
```

```
-----
\
(1 row)
```

8.4.2.14、配置某个用户连接后字符串存在反斜线时是否发出警告

```
postgres=# alter role pgxz set escape_string_warning to on;
```

```
ALTER ROLE
```

该参数要在 standard_conforming_strings 为 off 时才有效

8.4.2.15、配置某个用户连接后 expr = NULL 当成 expr is null 处理

```
postgres=# alter role pgxz set transform_null_equals to on;
```

```
ALTER ROLE
```

```
postgres=# \q
```

```
[pgxz@VM_0_3_centos root]$ psql
```

```
psql (PostgreSQL 10 (TBase 2.01))
```

```
Type "help" for help.
```

```
postgres=# select * from tbase where mc=null;
```

```
id | mc
```

```
----+----
 2 |
(1 row)
```

```
postgres=# set transform_null_equals to off;
SET
postgres=# select * from tbase where mc=null;
 id | mc
----+----
(0 rows)
```

如果打开, 那么表达式 `expr = NULL` 或者 `NULL = expr` 将被当做 `expr IS NULL` 处理, 也就是说, 如果 `expr` 得出 `NULL` 值则返回真, 否则返回假。正确的 SQL 标准兼容的 `expr = NULL` 行为总是返回 `NULL`(未知)

8.4.2.16、配置某个用户执行查询出错时中断连接

```
postgres=# alter role pgxz set exit_on_error to on;
ALTER ROLE
postgres=# \q
[pgxz@VM_0_3_centos ~]$ psql
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=# select pg_backend_pid();
 pg_backend_pid
-----
          2226
(1 row)
```

```
postgres=# select * from t1;
FATAL:  relation "t1" does not exist
LINE 1: select * from t1;
                ^

FATAL:  relation "t1" does not exist
LINE 1: select * from t1;
                ^
```

The connection to the server was lost. Attempting reset: Succeeded.

```
postgres=# select pg_backend_pid();
 pg_backend_pid
-----
          2259
(1 row)
```

如果为真, 那么任何错误都将终止当前连接。

8.4.2.17、查询用户的个性化配置

```
postgres=# \drds
```

```

List of settings
Role | Database | Settings
-----+-----+-----
pgxz |          | default_with_oids=on          +
      |          | standard_conforming_strings=on+
      |          | transform_null_equals=on
      | postgres | log_statement=none
(2 rows)

postgres=# select * from pg_db_role_setting where setrole!=0;
 setdatabase | setrole | setconfig
-----+-----+-----
           0 |       10 | {default_with_oids=on,standard_conforming_strings=on,transform_null_equals=on}
(1 row)

```

8.4.2.18、取消某个用户的个性化配置

```

postgres=# alter role pgxz reset all;
ALTER ROLE

```

8.4.3、配置表级别运行参数

8.4.3.1、配置配置个表触发 analyze 的条件

```

postgres=# alter table tbase set (autovacuum_analyze_scale_factor = 0.02);
ALTER TABLE
postgres=# alter table tbase set (autovacuum_analyze_threshold = 200);
ALTER TABLE

```

8.4.3.2、配置配置个表触发 vacuum 的条件

```

postgres=# alter table tbase set (autovacuum_vacuum_scale_factor = 0.02);
ALTER TABLE
postgres=# alter table tbase set (autovacuum_vacuum_threshold = 200);
ALTER TABLE
postgres=#

```

8.4.4、配置会话级别运行参数

8.4.4.1、配置会话的搜索路径

```

[pgxz@VM_0_3_centos ~]$ psql
psql (PostgreSQL 10 (TBase 2.01))

```

Type "help" for help.

```
postgres=# select * from order_main;
ERROR:  relation "order_main" does not exist
LINE 1: select * from order_main;
```

```
postgres=# show search_path ;
search_path
```

```
-----
"$user",public
(1 row)
```

```
postgres=# set search_path to "$user", public, pgxc;
```

SET

```
postgres=# select * from order_main;
```

```
id | dh
```

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

```
(0 rows)
```

注意：会话级别的配置参数会覆盖掉数据库和用户级别的配置参数，如下所示

```
postgres=# alter role pgxz set search_path = "$user", public, pgxc;
```

ALTER ROLE

```
postgres=# alter database postgres set search_path to "$user", public, pgxc,pg_catalog;
```

ALTER DATABASE

```
postgres=# \q
```

```
[pgxz@VM_0_3_centos ~]$ psql
```

```
psql (PostgreSQL 10 (TBase 2.01))
```

Type "help" for help.

```
postgres=# set search_path = "$user", public;
```

SET

```
postgres=# select * from order_main;
```

```
ERROR:  relation "order_main" does not exist
```

```
LINE 1: select * from order_main;
```

```
postgres=# show search_path ;
```

```
search_path
```

```
-----
"$user", public
```

```
(1 row)
```

8.4.4.2、配置会话使用的内存

■ 可使用的排序内存

```
postgres=# alter role pgxz set work_mem = '8MB';
ALTER ROLE
```

■ 可使用的维护内存

```
postgres=# alter role pgxz set maintenance_work_mem TO '256MB';
ALTER ROLE
```

8.4.4.3、配置会话使用的时区

```
postgres=# select now();
              now
-----
2018-03-30 17:00:35.374808+08
(1 row)
```

```
postgres=# set TimeZone to cet;
SET
postgres=# select now();
              now
-----
2018-03-30 11:00:47.198263+02
(1 row)
```

timestamp with time zone 类型对于本地化显示非常有帮助

8.4.4.4、配置会话使用的日期格式

```
postgres=# select current_date;
      date
-----
2018-03-30
(1 row)
```

```
postgres=# set DateStyle to SQL, DMY;
SET
postgres=# select current_date;
      date
-----
30/03/2018
(1 row)
```

```
postgres=# select '2010-01-01'::date;
      date
-----
```

01/01/2010

(1 row)

8.4.4.5、配置会话执行语句最多时长

```
postgres=# set statement_timeout = 1;
```

执行 1 毫秒超时

SET

```
postgres=# select * from tbase;
```

ERROR: canceling statement due to statement timeout

```
postgres=# set statement_timeout = 1000;
```

执行 1 秒超时

SET

```
postgres=# select * from tbase;
```

```
id | mc
```

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

```
1 | tbase
```

```
2 |
```

```
3 | pgxc
```

(3 rows)

8.4.4.6、配置会话的客户端编码

```
postgres=# set client_encoding to gbk;
```

SET

配置客户端编码为 gbk，适用于数据库编码为 utf8，应用程序编码为 gbk 的应用

8.4.4.7、配置会话的日志记录级别

```
postgres=# set log_statement=none;
```

SET

普通用户无法通过 session 改变日志级别

8.4.4.8、配置会话的 wal 日志写盘级别

```
postgres=# set synchronous_commit to local;
```

SET

该用户更新操作只要求本地提交

8.4.4.9、配置会话禁用某个规划器

```
postgres=# set enable_indexonlyscan to off;
SET
```

禁用 indexonlyscan 扫描

8.4.4.10、配置会话事务默认为只读事务

```
postgres=#set default_transaction_read_only to on;
SET
```

取消方法

```
postgres=#set default_transaction_read_only to off;
```

8.4.4.11、配置会话中建表是否需要 oid 列

```
postgres=# set default_with_oids to on;
SET
```

```
postgres=# create table t1(id int);
```

```
CREATE TABLE
```

```
postgres=# select oid,* from t1;
```

```
oid | id
```

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

```
(0 rows)
```

默认加了 oid 列

8.4.4.12、配置会话中反斜线是否当作普通字符使用

```
postgres=# set standard_conforming_strings to on;
SET
```

```
postgres=# select '\\';
```

```
?column?
```

```
-----
```

```
\\
```

```
(1 row)
```

如不当普通字符使用，则为转义符，而且使用时会警告

```
postgres=# set standard_conforming_strings to off;
SET
```

```
postgres=# select '\\';
```

WARNING: nonstandard use of \\ in a string literal

LINE 1: select '\\';

^

HINT: Use the escape string syntax for backslashes, e.g., E'\\'.

?column?

\

(1 row)

8.4.4.13、配置某个会话查询中字符串存在反斜线时是否发出警告

postgres=# set escape_string_warning to on;

SET

postgres=# select '\\';

WARNING: nonstandard use of \\ in a string literal

LINE 1: select '\\';

^

HINT: Use the escape string syntax for backslashes, e.g., E'\\'.

?column?

\

(1 row)

postgres=# set escape_string_warning to off;

SET

postgres=# select '\\';

?column?

\

(1 row)

该参数要在 standard_conforming_strings 为 off 时才有效

8.4.4.14、配置会话中 expr = NULL 当成 expr is null 处理

postgres=# set transform_null_equals to on;

SET

postgres=# select * from tbase where mc=null;

id | mc

----+----

2 |

(1 row)

postgres=# set transform_null_equals to off;

SET

```
postgres=# select * from tbase where mc=null;
 id | mc
----+----
(0 rows)
```

如果打开, 那么表达式 `expr = NULL` 或者 `NULL = expr` 将被当做 `expr IS NULL` 处理, 也就是说, 如果 `expr` 得出 `NULL` 值则返回真, 否则返回假。正确的 SQL 标准兼容的 `expr = NULL` 行为总是返回 `NULL`(未知)

8.4.4.15、配置会话查询出错时中断连接

```
postgres=# select pg_backend_pid();
 pg_backend_pid
-----
          22692
(1 row)

postgres=# set exit_on_error to on;
SET
postgres=# select * from tt;
FATAL:  relation "tt" does not exist
LINE 1: select * from tt;
              ^

FATAL:  relation "tt" does not exist
LINE 1: select * from tt;
              ^

The connection to the server was lost. Attempting reset: Succeeded.
postgres=# select pg_backend_pid();
 pg_backend_pid
-----
          23843
(1 row)

postgres=# set exit_on_error to off;
SET
postgres=# select * from tt;
ERROR:  relation "tt" does not exist
LINE 1: select * from tt;
              ^

postgres=# select pg_backend_pid();
 pg_backend_pid
-----
          23843
(1 row)
```

如果为真, 那么任何错误都将终止当前连接。

8.4.4.16、查询当前会话配置

```
postgres=# show log_statement;
```

```
log_statement
```

```
-----
```

```
all
```

```
(1 row)
```

```
postgres=# select current_setting('log_statement') as log_statement;
```

```
log_statement
```

```
-----
```

```
all
```

```
(1 row)
```

```
postgres=# select current_setting('log_statement') as log_statement,current_setting('statement_timeout') as statement_timeout;
```

```
log_statement | statement_timeout
```

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

```
all          | 0
```

```
(1 row)
```

```
postgres=# show all;
```

8.4.4.17、还原会话配置

```
postgres=# reset all;
```

```
RESET
```

```
postgres=#
```

9、审计

9.1、审计相关配置

```
#审计总开关，默认是关闭
```

```
enable_audit = on
```

```
#日期循环周期，默认是一周
```

```
alog_filename = 'audit-%A-%H.log'
```

```
#日志文件轮转检测周期，默认一天检测一次
```

```
alog_rotation_age = 24h
```

```
#日志文件轮转阈值大小，在每次轮转检测时，如果当前日志文件大小超过阈值，则进行日志文件轮转
```

```
alog_rotation_size = 10MB
```

#覆盖旧日志文件时是否 truncate 掉,默认是不 truncate 掉

```
alog_truncate_on_rotation = off
```

注：审计可以针对 cn 和 dn 节点，建议只 cn 进行配置

9.2、审计管理员

tbase 审计管理员的名称默认为“audit_admin”,审计管理员连接 Tbase 方法

```
psql -h 127.0.0.1 -p 15432 -d postgres -U audit_admin
```

psql (PostgreSQL 10.0 TBase V2)

Type "help" for help.

```
postgres=>
```

9.3、语句审计

语句审计用于审计系统中执行的 sql 语句，跟用户和对象无关，其覆盖面最广。

9.3.1、查询当前库配置了那些语句审计规则

```
select * from pg_audit_stmt_conf_detail ;
```

9.3.2、审计所有语句

```
postgres=> audit all;
```

AUDIT

9.3.3、审计指定语句

配置单个 select table 语句

```
postgres=> audit select table;
```

AUDIT

配置多个语句 create table,create view

```
postgres=> audit create table,create view;
```

AUDIT

使用短语

```
postgres=> audit Table;
```

AUDIT

上面一共审计了 Create Table / Drop Table / Truncate Table

9.3.4、清除所有语句审计

```
postgres=> clean statement audit;
CLEAN AUDIT
```

9.3.5、禁用指定语句

#禁用单个语句

```
postgres=> noaudit create table;
NO AUDIT
```

```
postgres=> select * from pg_audit_stmt_conf_detail ;
   auditor   | action_name | action_mode | action_ison
-----+-----+-----+-----
 audit_admin | Create View | Audit Always | t
 audit_admin | Create Table | Audit Always | f
(2 rows)
```

#禁用多个语句

```
postgres=> noaudit create table,create view;
NO AUDIT
```

#使用短语

```
postgres=> noaudit Table ;
NO AUDIT
```

9.3.6、执行成功时审计

```
postgres=> audit Select Table WHENEVER SUCCESSFUL;
AUDIT
postgres=>
```

那么这样的语句执行不记录

```
postgres=> select * from t limit 1;
ERROR:  permission denied for relation t
```

将 action_mode=Audit Always 的语句修改为 Audit When Success

```
postgres=> select * from pg_audit_stmt_conf_detail ;
   auditor   | action_name | action_mode | action_ison
-----+-----+-----+-----
 audit_admin | Select Table | Audit Always | t
(1 row)
```

```
postgres=> noaudit select table WHENEVER NOT SUCCESSFUL;
NO AUDIT
postgres=> select * from pg_audit_stmt_conf_detail ;
   auditor   | action_name |   action_mode   | action_ison
-----+-----+-----+-----
audit_admin | Select Table | Audit When Success | t
(1 row)
```

9.3.7、执行失败时审计

```
postgres=> audit Select Table WHENEVER NOT SUCCESSFUL;
AUDIT
postgres=>
```

那么这样的语句执行会被记录

```
postgres=> select * from t limit 1;
ERROR:  permission denied for relation t
```

将 action_mode=Audit Always 的语句修改为 Audit When Not Success

```
postgres=> select * from pg_audit_stmt_conf_detail ;
   auditor   | action_name | action_mode | action_ison
-----+-----+-----+-----
audit_admin | Select Table | Audit Always | t
(1 row)
```

```
postgres=> noaudit select table WHENEVER  SUCCESSFUL;
NO AUDIT
postgres=> select * from pg_audit_stmt_conf_detail ;
   auditor   | action_name |   action_mode   | action_ison
-----+-----+-----+-----
audit_admin | Select Table | Audit When Not Success | t
(1 row)
```

9.3.8、语句审计列表

■ 全名列表

Add Partitions
Alter Database
Alter Extension
Alter Function
Alter Index
Alter Materialized View

Create Node
Create Node Group
Create Role
Create Schema
Create Sequence
Create Sharding Group

Drop Sequence
Drop Sharding in Group
Drop Table
Drop Tablespace
Drop Trigger
Drop Type

Alter Node	Create Table	Drop User
Alter Role	Create Tablespace	Drop View
Alter Schema	Create Trigger	Enable Trigger
Alter Sequence	Create Type	Grant Function
Alter System	Create User	Grant Sequence
Alter Table	Create View	Grant Table
Alter Tablespace	Delete Table	Grant Type
Alter Trigger	Disable Trigger	Insert Table
Alter Type	Drop Database	Lock Table
Alter User	Drop Extension	Select Sequence
Alter View	Drop Function	Select Table
Comment Table	Drop Index	Set Role
Create Database	Drop Materialized View	System Audit
Create Extension	Drop Node	System Grant
Create Function	Drop Node Group	Truncate Table
Create Index	Drop Role	Update Table
Create Materialized View	Drop Schema	

■ 短语

短语	对应列表
Database	Alter Database Create Database Drop Database
Extension	Alter Extension Create Extension Drop Extension
Function	Alter Function Create Function Drop Function
Group	Create Node Group Create Sharding Group Drop Node Group Drop Sharding in Group
Index	Alter Index Create Index Drop Index
Materialized view	Alter Materialized View Create Materialized View Drop Materialized View
Node	Alter Node Create Node Drop Node
Partition	Add Partition
Role	Alter Role Create Role

	Drop Role Set Role
Schema	Alter Schema Create Schema Drop Schema
Sequence	Create Sequence Drop Sequence
Table	Create Table Drop Table Truncate Table
Tablespace	Alter Tablespace Create Tablespace Drop Tablespace
Trigger	Alter Trigger Create Trigger Drop Trigger Disable Trigger Enable Trigger
Type	Alter Type Create Type Drop Type
User	Alter User Create User Drop User
View	Alter View Create View Drop View

■ 定向语句

语句名	说明
Alter Sequence	审计 Alter Sequence 操作
Alter System	审计 Alter System 操作
Alter Table	审计 Alter Table 操作
Comment Table	审计在 Table、View、Materialized View、Column 上的 Comment 操作
Delete Table	审计在 Table、View 上的 Delete 操作
Grant Function	审计在 Function 上的 Grant、Revoke 操作
Grant Sequence	审计在 Sequence 上的 Grant、Revoke 操作
Grant Table	审计在 Table、View、Materialized View、Column 上的 Grant、Revoke 操作
Grant Type	审计在 Type 上的 Grant、Revoke 操作
Insert Table	审计在 Table、View 上的 Insert 操作
Lock Table	审计在 Table 上的 Lock 操作
Select Sequence	审计在 Sequence 上的 Select 操作

Select Table	审计在 Table、View、Materialized View 上的 Select 操作
System Audit	审计 Audit、NoAudit 操作
System Grant	审计 Grant、Revoke 操作
Update Table	审计在 Table、View 上的 Update 操作

9.4、用户审计

语句审计用于审计系统中某个用户执行的 sql 语句，跟用户强相关。

9.4.1、查询当前库配置了那些用户审计规则

```
select * from pg_audit_user_conf_detail ;
```

9.4.2、审计用户所有语句

```
postgres=> audit all by postgres;
AUDIT
```

9.4.3、审计用户执行的某些语句

配置单个 Select Table 语句

```
postgres=> audit Select Table by postgres;
AUDIT
```

配置多个语句

```
postgres=> audit create table,alter table by postgres;
AUDIT
```

使用短语

```
postgres=> audit Table by postgres;
AUDIT
```

上面一共审计了 Create Table / Drop Table / Truncate Table

```
postgres=> select * from pg_audit_user_conf_detail ;
 auditor | user_name | action_name | action_mode | action_ison
-----+-----+-----+-----+-----
 audit_admin | postgres | Create Table | Audit Always | t
 audit_admin | postgres | Drop Table   | Audit Always | t
 audit_admin | postgres | Truncate Table | Audit Always | t
(3 rows)
```

9.4.4、清除用户审计规则

清除所有用户审计规则

```
postgres=> clean user audit;
CLEAN AUDIT
```

清除某个用户审计规则

```
postgres=> clean user audit by postgres;
CLEAN AUDIT
```

9.4.5、禁用某个用户执行语句审计规则

#禁用单个语句

```
postgres=> noaudit create table by postgres;
NO AUDIT
```

```
postgres=> select * from pg_audit_user_conf_detail ;
  auditor   | user_name | action_name   | action_mode | action_ison
-----+-----+-----+-----+-----
audit_admin | postgres | Drop Table    | Audit Always | t
audit_admin | postgres | Truncate Table | Audit Always | t
audit_admin | postgres | Create Table  | Audit Always | f
(3 rows)
```

#禁用多个语句

```
postgres=> noaudit Truncate Table,Create Table by postgres;
NO AUDIT
```

#使用短语

```
postgres=> noaudit Table by postgres;
NO AUDIT
```

9.4.6、用户执行语句成功时审计

```
postgres=> audit Select Table by postgres WHENEVER SUCCESSFUL;
AUDIT
```

```
postgres=> select * from pg_audit_user_conf_detail ;
  auditor   | user_name | action_name   | action_mode | action_ison
-----+-----+-----+-----+-----
audit_admin | postgres | Select Table | Audit When Success | t
(1 row)
```

那么这样的语句执行不记录

```
postgres=> select * from t limit 1;
ERROR:  permission denied for relation t
```

将 action_mode=Audit Always 的语句修改为 Audit When Success

```
postgres=> select * from pg_audit_user_conf_detail ;
  auditor   | user_name | action_name | action_mode | action_ison
-----+-----+-----+-----+-----
  audit_admin | postgres | Select Table | Audit Always | t
(1 row)
```

```
postgres=> noaudit select table by postgres WHENEVER NOT SUCCESSFUL;
NO AUDIT
```

```
postgres=> select * from pg_audit_user_conf_detail ;
  auditor   | user_name | action_name | action_mode | action_ison
-----+-----+-----+-----+-----
  audit_admin | postgres | Select Table | Audit When Success | t
(1 row)
```

9.4.7、用户执行语句失败时审计

```
postgres=> audit Select Table by postgres WHENEVER NOT SUCCESSFUL;
AUDIT
postgres=> select * from pg_audit_user_conf_detail ;
  auditor   | user_name | action_name | action_mode | action_ison
-----+-----+-----+-----+-----
  audit_admin | postgres | Select Table | Audit When Not Success | t
(1 row)
```

那么这样的语句执行记录

```
postgres=> select * from t limit 1;
ERROR:  permission denied for relation t
```

将 action_mode=Audit Always 的语句修改为 Audit When Not Success

```
postgres=> select * from pg_audit_user_conf_detail ;
  auditor   | user_name | action_name | action_mode | action_ison
-----+-----+-----+-----+-----
  audit_admin | postgres | Select Table | Audit Always | t
(1 row)
```

```
postgres=> noaudit select table by postgres WHENEVER SUCCESSFUL;
NO AUDIT
```

```
postgres=> select * from pg_audit_user_conf_detail ;
  auditor   | user_name | action_name | action_mode | action_ison
-----+-----+-----+-----+-----
  audit_admin | postgres | Select Table | Audit When Not Success | t
```

(1 row)

9.4.8、用户审计语句列表

与语句审计 9.3.8 一致

9.5、对象审计

语句审计用于审计系统中对某个对象（table/view/seq...）的操作，跟某个对象强有关。

9.5.1、查询当前库配置了那些对象审计规则

```
select * from pg_audit_obj_conf_detail ;
```

9.5.2、审计某个对象

对一个表访问进行审计

```
audit all on table public.t ;
```

对一个视图访问进行审计

```
audit all on view public.t_view;
```

对一个序列访问进行审计

```
audit all on sequence t_seq;
```

对一个函数访问进行审计

```
audit all on sequence t_seq;
```

对一个物化视图进行审计

```
audit all on MATERIALIZED VIEW t_mv ;
```

9.5.3、审计对象某些操作语句

配置单个语句

```
postgres=> audit select on public.t ;
```

```
AUDIT
```

配置多个语句

```
postgres=> audit select,insert on public.t ;
```

```
AUDIT
```

9.5.4、清除对象审计规则

清除所有对象审计规则

```
postgres=> clean object audit;
CLEAN AUDIT
```

清除某个对象审计规则

```
postgres=> clean object audit on view public.t_view;
CLEAN AUDIT
```

9.5.5、禁用对象审计语句

#禁用某个对象的所有审计语句

```
postgres=> noaudit all on table public.t;
NO AUDIT
```

#禁用单个语句

```
postgres=> noaudit insert on table public.t;
NO AUDIT
```

#禁用多个语句

```
postgres=> noaudit update,delete on table public.t;
NO AUDIT
```

9.5.6、访问成功才审计

```
postgres=> audit all on table public.t WHENEVER SUCCESSFUL;
AUDIT
```

那么这样的语句执行不记录

```
postgres=> select * from t limit 1;
ERROR: permission denied for relation t
```

将 action_mode=Audit Always 的语句修改为 Audit When Success

```
postgres=> select * from pg_audit_obj_conf_detail where object_desc='table t' and action_name='Select' ;
 auditor   | object_class | object_desc | action_name | action_mode | action_ison
-----+-----+-----+-----+-----+-----
 audit_admin | pg_class    | table t    | Select     | Audit Always | t
(1 row)
```

```
postgres=> noaudit select on table public.t WHENEVER NOT SUCCESSFUL;
NO AUDIT
```

```
postgres=> select * from pg_audit_obj_conf_detail where object_desc='table t' and action_name='Select' ;
```

```

auditor | object_class | object_desc | action_name | action_mode | action_ison
-----+-----+-----+-----+-----+-----
audit_admin | pg_class | table t | Select | Audit When Success | t
(1 row)

```

9.5.7、访问不成功才审计

```

postgres=> audit all on table public.t WHENEVER NOT SUCCESSFUL;
AUDIT

```

那么这样的语句执行将被记录

```

postgres=> select * from t limit 1;
ERROR: permission denied for relation t

```

将 action_mode=Audit Always 的语句修改为 Audit When Not Success

```

ppostgres=> select * from pg_audit_obj_conf_detail where object_desc='table t' and action_name='Select' ;
auditor | object_class | object_desc | action_name | action_mode | action_ison
-----+-----+-----+-----+-----+-----
audit_admin | pg_class | table t | Select | Audit Always | t
(1 row)

```

```

postgres=> noaudit select on table public.t WHENEVER SUCCESSFUL;
NO AUDIT

```

```

postgres=> select * from pg_audit_obj_conf_detail where object_desc='table t' and action_name='Select' ;
auditor | object_class | object_desc | action_name | action_mode | action_ison
-----+-----+-----+-----+-----+-----
audit_admin | pg_class | table t | Select | Audit When Not Success | t
(1 row)

```

9.5.8、审计对象对应语句列表

对象	对应语句
Table	Alter Audit Comment Delete Grant Insert Lock Rename Select Update
View	Alter

	Audit Comment Delete Grant Insert Rename Select Update
Sequence	Alter Audit Grant Rename Select
Materialized View	Alter Audit Comment Grant Rename Select Update
Function	Audit Comment Grant
Type	Comment Rename

9.6、对象默认审计

对象默认审计用于审计所有存在或者将来才存在的对象的操作，这是一个非常有用的配置功能，如配置所有表（包括后面要建立的表）的 `select` 访问审计

9.6.1、查询当前库配置了那些默认对象审计规则

```
select * from pg_audit_obj_def_opts_detail ;
```

9.6.2、配置对象默认审计规则

#配置所有默认规则生效

```
postgres=> audit all on default;
```

AUDIT

#配置单条

```
postgres=> audit select on default;
```

AUDIT

#配置多条

```
postgres=> audit select,insert on default;
AUDIT
```

9.6.3、清除默认规则

```
postgres=> clean object audit on default;
CLEAN AUDIT
```

9.6.4、禁用默认的对象审计规则

```
#禁用所有配置的对象审计语句
postgres=> noaudit all on default;
NO AUDIT
```

```
#禁用某个默认规则语句
postgres=> noaudit select on default;
NO AUDIT
```

```
#禁用多个默认规则语句
postgres=> noaudit insert,delete on default;
NO AUDIT
```

9.6.5、访问成功才审计

```
postgres=> audit all on default WHENEVER SUCCESSFUL;
AUDIT
```

那么这样的语句执行不记录

```
postgres=> select * from t limit 1;
ERROR:  permission denied for relation t
```

将 action_mode=Audit Always 的语句修改为 Audit When Success

```
postgres=> noaudit all on default WHENEVER NOT SUCCESSFUL;
NO AUDIT
```

```
postgres=> select * from pg_audit_obj_def_opts_detail ;
   auditor   | action_name |   action_mode   | action_ison
-----+-----+-----+-----
audit_admin | Select     | Audit When Success | t
audit_admin | Insert     | Audit When Success | t
```

9.6.6、访问不成功才审计

```
postgres=> audit all on default WHENEVER NOT SUCCESSFUL;
AUDIT
```

那么这样的语句执行将被记录

```
postgres=> select * from t limit 1;
ERROR:  permission denied for relation t
```

将 action_mode=Audit Always 的语句修改为 Audit When Not Success

```
postgres=> noaudit all on default WHENEVER SUCCESSFUL;
NO AUDIT
postgres=> select * from pg_audit_obj_def_opts_detail ;
   auditor   | action_name |   action_mode   | action_ison
-----+-----+-----+-----
 audit_admin | Alter      | Audit When Not Success | t
 audit_admin | Audit      | Audit When Not Success | t
```

9.6.7、默认规则列表

Alter	Insert
Audit	Lock
Comment	Rename
Delete	Select
Grant	Update

9.7、审计日志

9.7.1、日志记录地方

pg_log/audit

9.7.2、日志格式

对象审计日志

AuditOutMessage: AuditType: "Object Audit", QueryString: "select * from t limit 1;", TopCommandTag: "SELECT", DatabaseID: 13319, DatabaseName: "postgres", DatabaseUserID: 10, DatabaseUserName: "postgres", NodeOid: 11838, NodeName: "cn01", NodeType: "Coordinator", NodeHost: "172.16.0.3", NodePort: 15432, NodeOSUser: "postgres", PostgresPID: 15078, PostmasterPID: 13747, BackendStartTime: "2018-05-29 19:26:45 CST", QueryBeginTime: "2018-05-29 21:37:10 CST", QueryEndTime: "2018-05-29 21:37:10 CST", QueryIsSuccess: 1, ClientHost: "127.0.0.1", ClientHostname: "unknown", ClientPort: "56768", AppName: "psql", ObjectClassID: 1259, ObjectId: 17920, ObjectSubId: 0, ObjectType: "Table", ObjectName: "t", ActionID: 60010, ActionName: "Select", ExecStatus: "Exec Successful"

对象默认审计日志

AuditOutMessage: AuditType: "Object Default Audit", QueryString: "select * from t limit 1;", TopCommandTag: "SELECT", DatabaseID: 13319, DatabaseName: "postgres", DatabaseUserID: 10, DatabaseUserName: "postgres", NodeOid: 11838, NodeName: "cn01", NodeType: "Coordinator", NodeHost: "172.16.0.3", NodePort: 15432, NodeOSUser: "postgres", PostgresPID: 15078, PostmasterPID: 13747, BackendStartTime: "2018-05-29 19:26:45 CST", QueryBeginTime: "2018-05-29 21:35:40 CST", QueryEndTime: "2018-05-29 21:35:40 CST", QueryIsSuccess: 1, ClientHost: "127.0.0.1", ClientHostname: "unknown", ClientPort: "56768", AppName: "psql", ObjectClassID: 1259, ObjectId: 17920, ObjectSubId: 0, ObjectType: "Table", ObjectName: "t", ActionID: 60010, ActionName: "Select", ExecStatus: "Exec Successful"

用户审计日志

AuditOutMessage: AuditType: "User Audit", QueryString: "select * from t limit 1;", TopCommandTag: "SELECT", DatabaseID: 13319, DatabaseName: "postgres", DatabaseUserID: 10, DatabaseUserName: "postgres", NodeOid: 11838, NodeName: "cn01", NodeType: "Coordinator", NodeHost: "172.16.0.3", NodePort: 15432, NodeOSUser: "postgres", PostgresPID: 15078, PostmasterPID: 13747, BackendStartTime: "2018-05-29 19:26:45 CST", QueryBeginTime: "2018-05-29 21:39:32 CST", QueryEndTime: "2018-05-29 21:39:32 CST", QueryIsSuccess: 1, ClientHost: "127.0.0.1", ClientHostname: "unknown", ClientPort: "56768", AppName: "psql", ObjectClassID: 1259, ObjectId: 17920, ObjectSubId: 0, ObjectType: "Table", ObjectName: "t", ActionID: 30013, ActionName: "Select Table", ExecStatus: "Exec Successful"

语句审计日志

AuditOutMessage: AuditType: "Statement Audit", QueryString: "select * from t limit 1;", TopCommandTag: "SELECT", DatabaseID: 13319, DatabaseName: "postgres", DatabaseUserID: 10, DatabaseUserName: "postgres", NodeOid: 11838, NodeName: "cn01", NodeType: "Coordinator", NodeHost: "172.16.0.3", NodePort: 15432, NodeOSUser: "postgres", PostgresPID: 15078, PostmasterPID: 13747, BackendStartTime: "2018-05-29 19:26:45 CST", QueryBeginTime: "2018-05-29 21:41:40 CST", QueryEndTime: "2018-05-29 21:41:40 CST", QueryIsSuccess: 1, ClientHost: "127.0.0.1", ClientHostname: "unknown", ClientPort: "56768", AppName: "psql", ObjectClassID: 1259, ObjectId: 17920, ObjectSubId: 0, ObjectType: "Table", ObjectName: "t", ActionID: 30013, ActionName: "Select Table", ExecStatus: "Exec Successful"

9.7.3、审计日志记录顺序

优先级：对象审计 >> 默认对象审计 >> 用户审计 >> 语句审计

匹配规则：命中不同类型的审计，按照上述审计顺序匹配最高优先级；命中同类型审计，输出所有匹配。

9.8、其它审计功能

9.8.1、清除所有审计配置

```
clean all audit;
```

9.8.2、清除系统已经失效的审计配置

对像，用户审计配置后，如果删除表或者用户，系统会残留审计配置，这时需要使用下面命令清除失效的审计配置

```
clean unknown audit;
```

9.9、细粒度审计（fag）

细粒度审计用于配置审计对表的 insert/delete/update/select，并且他可以配置命令的条件和命中列

9.9.1、与细粒度审计相关的配置

#审计总开关，默认是关闭，修改该配置只需要 reload

```
enable_fga = on
```

9.9.2、细粒度审计管理员

细粒度审计规则的配置需要使用 tbase 审计管理员的名称默认为“audit_admin”，审计管理员连接 Tbase 方法

```
psql -h 127.0.0.1 -p 15432 -d postgres -U audit_admin
```

```
psql (PostgreSQL 10.0 TBase V2)
```

```
Type "help" for help.
```

```
postgres=>
```

9.9.3、环境准备

```
[postgres@VM_0_3_centos data]$ psql -h 127.0.0.1 -U postgres -d postgres -p 15432
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

--确认 enable_fga 开关是打开的

```
postgres=# show enable_fga;
enable_fga
```

```
-----
on
```

```
(1 row)
```

--创建测试表

```
postgres=# create table fga_test(id int, f1 int,f2 text);
CREATE TABLE
postgres=#
```

9.9.4、添加审计规则

9.9.4.1、添加规则 add_policy 函数说明

■ 函数名

add_policy

■ 参数说明

参数名称	参数类型	参数说明
object_schema	text	审计对象的 schema，若函数中未定义，则取当前 schema
object_name	text	审计对象的名称，审计对象可以是表和视图
policy_name	text	策略名字，全局唯一
audit_condition	text	命中条件，字符串描述的布尔表达式，默认为 NULL，代表永真
audit_columns	text	访问列，可以指明多列，默认值为 null
handler_schema	text	内置告警函数的 schema，若函数中未定义，则取当前 schema,audit_enable，默认值为 null
handler_module	text	内置告警函数的名称，当前不支持参数传入，默认值为 null
audit_enable	BOOLEAN	是否启用策略，1 代表启用，0 代表禁用，

		默认为 1
statement_types	text	支持 SELECT, INSERT, DELETE, UPDATE 中的 1 种或者多种, 默认为 SELECT
audit_column_opts	BOOLEAN	协同 audit_columns, 如果是 1 代表命中所有列则审计输出, 0 代表只要命中任意 1 个列则审计匹配, 默认为 false

■ 使用方法

select add_policy(.....), 比如

```
select add_policy(object_schema:='public', object_name:='fga_test', policy_name:='poli_1');
```

9.9.4.2、添加规则实例

■ 增/删/改/查

```
[postgres@VM_0_3_centos root]$ psql -h 127.0.0.1 -p 15432 -U audit_admin -d postgres
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=>          select          add_policy(object_schema:='public',          object_name:='fga_test',
audit_columns:='f1',policy_name:='fga_test_1', audit_condition:='id=1', statement_types:='insert');
add_policy
-----
t
(1 row)
```

```
postgres=>          select          add_policy(object_schema:='public',          object_name:='fga_test',
audit_columns:='f1',policy_name:='fga_test_2', audit_condition:='f1<10', statement_types:='delete');
add_policy
-----
t
(1 row)
```

```
postgres=>          select          add_policy(object_schema:='public',          object_name:='fga_test',
audit_columns:='f1',policy_name:='fga_test_3', audit_condition:='f1>10', statement_types:='update');
add_policy
-----
t
(1 row)
```

```
postgres=>          select          add_policy(object_schema:='public',          object_name:='fga_test',
audit_columns:='f1',policy_name:='fga_test_4', audit_condition:='f1=10', statement_types:='select');
```

```
add_policy
```

```
-----
t
(1 row)
```

语句验证

```
[postgres@VM_0_3_centos data]$ psql -h 127.0.0.1 -U postgres -d postgres -p 15432
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=# insert into fga_test values(1,1,'a');
INSERT 0 1
postgres=# insert into fga_test values(2,5,'a');
INSERT 0 1
postgres=# insert into fga_test values(12,12,'a');
INSERT 0 1
postgres=# insert into fga_test values(12,10,'a');
INSERT 0 1
postgres=#
```

--上面的 insert 语句只有 insert into fga_test values(1,1,'a'); 会命中

--返回 f1=10, 所以命中 select 规则

```
postgres=# select * from fga_test where id=12;
 id | f1 | f2
----+----+----
 12 | 12 | a
 12 | 10 | a
(2 rows)
```

--没有返回 f1 列, 不命中 select

```
postgres=# select id from fga_test where id=12;
 id
----
 12
 12
(2 rows)
```

--反加的 id 不等于 10, 不命中 select

```
postgres=# select * from fga_test where id=2;
 id | f1 | f2
----+----+----
  2 |  5 | a
(1 row)
```

--更新涉及记录中 f1 的值不大于 10，所以不命中

```
postgres=# update fga_test set f2='b' where id=1;
UPDATE 1
```

--更新涉及记录中 f1 的值大于 10，所以命中

```
postgres=# update fga_test set f2='b' where f1=12;
UPDATE 1
```

--返回 f1=10 的记录，所以命中

```
postgres=# select * from fga_test where id=12;
 id | f1 | f2
----+----+----
 12 | 10 | b
 12 | 12 | b
(2 rows)
```

--没有返回 f1=10 的记录，所以不命中

```
postgres=# select * from fga_test where f1=12;
 id | f1 | f2
----+----+----
 12 | 12 | b
(1 row)
```

--没有返回 f1 列，所以不命中

```
postgres=# select id,f2 from fga_test where id=12;
 id | f2
----+----
 12 | b
 12 | b
(2 rows)
```

■ 复合命中条件

```
[postgres@VM_0_3_centos root]$ psql -h 127.0.0.1 -p 15432 -U audit_admin -d postgres
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=>          select          add_policy(object_schema:='public',          object_name:='fga_test',
audit_columns:='f1',policy_name:='fga_test_5', audit_condition:='id=2 and f1=100', statement_types:='insert');
 add_policy
-----
 t
(1 row)
```

语句验证

```
[postgres@VM_0_3_centos data]$ psql -h 127.0.0.1 -U postgres -d postgres -p 15432
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=# insert into fga_test values(10,1,'a');
INSERT 0 1
postgres=# insert into fga_test values(2,99,'a');
INSERT 0 1
--只有 id=2 and fl=100 中才能命中
postgres=# insert into fga_test values(2,100,'a');
INSERT 0 1
postgres=#
```

■ 多个操作组合 statment_types

```
[postgres@VM_0_3_centos root]$ psql -h 127.0.0.1 -p 15432 -U audit_admin -d postgres
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=>          select          add_policy(object_schema:='public',          object_name:='fga_test',
audit_columns:='fl',policy_name:='fga_test_6', audit_condition:='id=101', statement_types:='insert,select');
  add_policy
-----
t
(1 row)

postgres=>
```

语句验证

```
[postgres@VM_0_3_centos data]$ psql -h 127.0.0.1 -U postgres -d postgres -p 15432
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
--id=101 的 insert 命中
postgres=# insert into fga_test values(101,20,'a');
INSERT 0 1
```

```
--返回记录中包含 id=101 的记录，所以命中
postgres=# select * from fga_test where fl=20;
 id | fl | f2
----+----+----
 101 | 20 | a
(1 row)
```

```
postgres=#
```

■ 多个访问列 audit_column 组合

```
[postgres@VM_0_3_centos root]$ psql -h 127.0.0.1 -p 15432 -U audit_admin -d postgres
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

--需要命中所有列

```
postgres=>          select          add_policy(object_schema:='public',          object_name:='fga_test',
audit_columns:='id,f2',policy_name:='fga_test_7',audit_condition:='f1=100',
statement_types:='select',audit_column_opts:=true);
 add_policy
-----
 t
(1 row)(1 row)
```

--只需要命中其中一列

```
postgres=>          select          add_policy(object_schema:='public',          object_name:='fga_test',
audit_columns:='id,f2',policy_name:='fga_test_8',audit_condition:='f1=1',
statement_types:='select',audit_column_opts:=false);
 add_policy
-----
 t
(1 row)
```

语句验证

--f1=100,返回 id,f2 列，所以命中

```
postgres=# select id ,f2 from   fga_test where f1=100;
 id | f2
----+----
  2 | a
  2 | a
(2 rows)
```

--少返回了 f2 列，所以不命中

```
postgres=# select id ,f1 from   fga_test where f1=100;
 id | f1
----+----
  2 | 100
  2 | 100
(2 rows)
```

--只返回 f1,所以不命中

```
postgres=# select f1 from fga_test where f1=1;
 f1
----
  1
  1
  1
(3 rows)
```

--返回了 id 列，所以命中

```
postgres=# select id from fga_test where f1=1;
 id
----
  1
  1
 10
(3 rows)
```

■ 命中规则时触发执行函数

```
[postgres@VM_0_3_centos data]$ psql -h 127.0.0.1 -U postgres -d postgres -p 15432
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=# create or replace function fga_handle_fun() returns void as
postgres-# $$
postgres-# begin
postgres-#     insert into fga_test values((random()*10000)::int,(random()*10000)::int,'fag 触发函数');
postgres-# end
postgres-# $$
postgres-# language plpgsql;
CREATE FUNCTION
postgres=#
```

```
[postgres@VM_0_3_centos root]$ psql -h 127.0.0.1 -p 15432 -U audit_admin -d postgres
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=>          select          add_policy(object_schema:='public',          object_name:='fga_test',
audit_columns:='id',policy_name:='fga_test_9',audit_condition:='f1=1',
statement_types:='select',handler_schema:='public', handler_module:='fga_handle_fun');
 add_policy
-----
 t
(1 row)
```

```
postgres=>
```

语句验证

```
postgres=# select * from fga_test;
```

```

 id | f1 | f2
-----+-----+-----
   2 |   5 | a
   1 |   1 | b
 101 |  20 | a
   10 |   1 | a
8155 | 7382 | fag 触发函数
(5 rows)
```

9.9.5、删除审计规则

9.9.5.1、删除规则函数 drop_policy 说明

■ 函数名

drop_policy

■ 参数说明

参数名称	参数类型	参数说明
object_schema	text	审计对象的 schema，若函数中未定义，则取当前 schema
object_name	text	审计对象的名称，审计对象可以是表和视图
policy_name	text	策略名字，全局唯一

■ 使用方法

```
select drop_policy(object_schema:='public', object_name:='fga_test', policy_name:='fga_test_7');
```

9.9.5.2、删除规则实例

```
[postgres@VM_0_3_centos root]$ psql -h 127.0.0.1 -p 15432 -U audit_admin -d postgres
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=> select drop_policy(object_schema:='public', object_name:='fga_test', policy_name:='fga_test_8');
drop_policy
```

```
-----
t
(1 row)
```

```
postgres=>
```

9.9.6、禁用审计规则

9.9.6.1、禁用规则函数 disable_policy 说明

■ 函数名

disable_policy

■ 参数说明

参数名称	参数类型	参数说明
object_schema	text	审计对象的 schema，若函数中未定义，则取当前 schema
object_name	text	审计对象的名称，审计对象可以是表和视图
policy_name	text	策略名字，全局唯一

■ 使用方法

```
select disable_policy(object_schema:='public', object_name:='fga_test', policy_name:='fga_test_5');
```

9.9.6.2、禁用规则实例

```
[postgres@VM_0_3_centos root]$ psql -h 127.0.0.1 -p 15432 -U audit_admin -d postgres
```

```
psql (PostgreSQL 10.0 TBase V2)
```

```
Type "help" for help.
```

```
postgres=> select disable_policy(object_schema:='public', object_name:='fga_test', policy_name:='fga_test_5');
drop_policy
```

```
-----
t
(1 row)
```

```
postgres=>
```

9.9.7、启用 ENABLE_POLICY 审计规则

9.9.7.1、启用规则函数 enable_policy 说明

■ 函数名

enable_policy

■ 参数说明

参数名称	参数类型	参数说明
object_schema	text	审计对象的 schema，若函数中未定义，则取当前 schema
object_name	text	审计对象的名称，审计对象可以是表和视图
policy_name	text	策略名字，全局唯一

■ 使用方法

```
select enable_policy(object_schema:='public', object_name:='fga_test', policy_name:='fga_test_5');
```

9.9.7.2、启用规则实例

```
[postgres@VM_0_3_centos root]$ psql -h 127.0.0.1 -p 15432 -U audit_admin -d postgres
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=> select enable_policy(object_schema:='public', object_name:='fga_test', policy_name:='fga_test_5');
drop_policy
```

```
-----
```

```
 t
(1 row)
```

```
postgres=>
```

9.9.8、系统表及视图说明

■ 系统表 pg_audit_fga_conf

存储细粒度规则元数据

■ 系统视图 pg_audit_fga_conf_detail

用于查询细粒度规则的视图

■ 系统视图 pg_audit_fga_policy_columns_detail

用于查询那些字段的配置了细粒度规则

9.9.9、细粒度审计日志及说明

9.9.9.1、日志字段说明

字段	字段说明
TIMESTAMP	审计记录生成时的时间标记
DB_USER	发出查询的数据库用户
DATABASE_NAME	数据库名称
PID	进程 pid 号
USERHOST	用户连接的机器的主机名 (IP)
USERPORT	用户连接的机器的端口
Datanode_name	dn 的 pgxc_node_name
SESSION_ID	连接 id
SESSION START TIME	连接开始时间
OBJECT_SCHEMA	对该表的访问触发了审计的表所有者
OBJECT_NAME	对该表的 SELECT 操作触发了审计的表名称
POLICY_NAME	触发审计的策略名称 (如果对表定义了多个策略, 则每个策略将插入一条记录。在此情况下, 该列显示哪些行是由哪个策略插入的。)
SQL_TEXT	由用户提交的 SQL 语句
SQL_BIND	由 SQL 语句使用的绑定变量 (如果存在)

9.9.9.2、日志内容

细粒度审计的日志存放于各个 dn 的 log/audit/*.fga 文件中, 内容格式如下所示

```
2018-06-26 15:03:28.332 CST,"postgres","postgres",8745,"172.16.0.3:44786","dn01",5b31e525.2229,2018-06-26
15:03:01 CST,"public","fga_test","fga_test_8","SELECT f1 FROM fga_test WHERE (f1 = 1)","SELECT"
```

10、安全与脱敏

10.1、安全相关配置

#脱敏开发

```
enable_data_mask = on
```

#加密开关

```
enable_transparent_crypt = on
```

10.2、安全管理员

tbase 安全管理员的名称默认为“mls_admin”,审计管理员连接 Tbase 方法

```
psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
```

```
psql (PostgreSQL 10.0 TBase V2)
```

```
Type "help" for help.
```

```
postgres=>
```

10.3、加载安全插件

所有库使用加密和脱敏特性时需要先加载安全插件

```
psql -h 127.0.0.1 -p 15432 -d postgres -U pgxz
```

```
psql (PostgreSQL 10.0 TBase V2)
```

```
Type "help" for help.
```

```
postgres=# create extension tbase_mls ;
```

```
CREATE EXTENSION
```

```
postgres=#
```

10.4、创建和删除加密算法

10.4.1、使用 mls_admin 用户创建加密算法

```
psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
```

```
psql (PostgreSQL 10.0 TBase V2)
```

```
Type "help" for help.
```

```
postgres=> select MLS_TRANSPARENT_CRYPT_CREATE_ALGORITHM('AES128', '1949');
```

```
mls_transparent_crypt_create_algorithm
```

```
-----
1
(1 row)
```

```
postgres=> select MLS_TRANSPARENT_CRYPT_CREATE_ALGORITHM('AES192', '2018');
mls_transparent_crypt_create_algorithm
```

```
-----
2
(1 row)
```

```
postgres=> select MLS_TRANSPARENT_CRYPT_CREATE_ALGORITHM('AES256', 'abcd');
mls_transparent_crypt_create_algorithm
```

```
-----
3
(1 row)
```

```
postgres=> select MLS_TRANSPARENT_CRYPT_CREATE_ALGORITHM('AES256', '1688');
mls_transparent_crypt_create_algorithm
```

```
-----
4
(1 row)
```

```
postgres=> select MLS_TRANSPARENT_CRYPT_CREATE_ALGORITHM('AES256', '1010');
mls_transparent_crypt_create_algorithm
```

```
-----
5
(1 row)
```

```
postgres=> select MLS_TRANSPARENT_CRYPT_CREATE_ALGORITHM('SM4', '123456789-123456');
mls_transparent_crypt_create_algorithm
```

```
-----
6
(1 row)
```

当前支持的加密算法有 AES128, AES192, AES256, SM4, 其中 SM4 是国密, 密文 token 一定要 16 位字符, 返回的 1, 2, 3, 4, 5, 6 就是加密算法 id, 后面表加密或者行加密时需要用到。

函数 `MLS_TRANSPARENT_CRYPT_CREATE_ALGORITHM(algo_name name, password text)` 说明, 创建加密算法, 并指定该加密算法的密钥, 返回加密算法 id。

10.4.2、使用 `mls_admin` 删除加密算法

```
psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
```

```
postgres=> select MLS_TRANSPARENT_CRYPT_DROP_ALGORITHM(4);
 mls_transparent_crypt_drop_algorithm
-----
 t
(1 row)
```

```
postgres=> select MLS_TRANSPARENT_CRYPT_DROP_ALGORITHM(5);
 mls_transparent_crypt_drop_algorithm
-----
 t
(1 row)
```

参数值 4 和 5 为加密算法 id

函数说明 `MLS_TRANSPARENT_CRYPT_DROP_ALGORITHM(algo_id integer)`，删除指定 ID 的加密算法。如果该加密算法已经被使用，则不允许删除。

10.4.3、查看已经配置的加密算法列表

```
psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=> select * from pg_transparent_crypt_policy_algorithm order by algorithm_id ;
```

algorithm_id	option	algorithm_name	encrypt_oid	decrypt_oid	password	pubkey	privatekey	option_args
1	1	AES128		0	0 1949			cipher-algo = AES128, compress-algo=1, compress-level=5
2	1	AES192		0	0 2018			cipher-algo = AES192, compress-algo=1, compress-level=5
3	1	AES256		0	0 abcd			cipher-algo = AES256, compress-algo=1, compress-level=5
6	4	SM4		0	0 123456789-123456			

(4 rows)

10.5、表（行）加密

10.5.1、使用非 `mls_admin` 用户创建要加密的数据表

```
psql -h 127.0.0.1 -p 15432 -d postgres -U pgxz
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=# create table security_tb (test_a int2, test_b int4, test_c int8, test_d varchar, test_e text) distribute by
shard(test_a);
CREATE TABLE
postgres=#
```

10.5.2、使用 mls_admin 用户配置表加密规则

```
[postgres@VM_0_3_centos pgxzconf]$ psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=> select MLS_TRANSPARENT_CRYPT_ALGORITHM_BIND_TABLE('public', 'security_tb', 1);
 mls_transparent_crypt_algorithm_bind_table
```

```
-----
 t
(1 row)
```

```
postgres=>
```

函数说明

```
MLS_TRANSPARENT_CRYPT_ALGORITHM_BIND_TABLE(schema_name name,
table_name name, algo_id integer)
```

针对一个表（需要携带 schema），绑定特定的加密算法，相当于表文件加密，配置加密规则的表不能存在记录，否则提示出错如下：

```
ERROR: table_name:public.security_tb is not empty
```

```
CONTEXT: PL/pgSQL function mls_transparent_crypt_algorithm_bind_table(name,name,integer) line 36 at
RAISE
```

10.5.3、使用 mls_admin 用户删除表加密规则

```
psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=> select MLS_TRANSPARENT_CRYPT_ALGORITHM_UNBIND_TABLE('public', 'security_tb');
 mls_transparent_crypt_algorithm_unbind_table
```

```
-----
 t
(1 row)
```

函数说明

```
MLS_TRANSPARENT_CRYPT_ALGORITHM_UNBIND_TABLE(schema_name name,
table_name name)
```

针对一个表（需要携带 schema）解绑定特定的加密算法，相当于解除文件加密，注：解绑前表数据要清掉，

否则提示如下错误:

ERROR: table_name:public.security_tb is not empty, please TRUNCATE table and try again

CONTEXT: PL/pgSQL function mls_transparent_crypt_algorithm_unbind_table(name,name) line 28 at RAISE

10.5.4、查看那些表已经配置了加密规则

```
psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
```

```
psql (PostgreSQL 10.0 TBase V2)
```

```
Type "help" for help.
```

```
postgres=> select * from pg_transparent_crypt_policy_map;
```

relid	attnum	algorithm_id	spcoid	schemaid	spcname	nspname	tblname
50343	-32767		1	1663	2200	pg_default	public security_tb

(1 row)

```
postgres=>
```

10.5.5、数据插入演示

```
postgres=# insert into security_tb values(111,1111,1234567890,'this is CV','This is TX');
INSERT 0 1
```

```
postgres=# insert into security_tb values(222,2222,1234567890,'this is CV','This is TX');
INSERT 0 1
```

```
postgres=# select * from security_tb;
```

test_a	test_b	test_c	test_d	test_e
222	2222	1234567890	this is CV	This is TX
111	1111	1234567890	this is CV	This is TX

(2 rows)

```
postgres=# update security_tb set test_d='TBase';
```

```
UPDATE 2
```

```
postgres=# select * from security_tb_col;
```

test_a	test_b	test_c	test_d	test_e
222	2222	1234567890	TBase	This is TX
111	1111	1234567890	TBase	This is TX

(2 rows)

10.6、列加密

10.6.1、使用非 mls_admin 用户创建要加密的数据表

```
psql -h 127.0.0.1 -p 15432 -d postgres -U pgxz
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=# create table security_tb_col (test_a int2, test_b int4, test_c int8, test_d varchar, test_e text) distribute by
shard(test_a);
CREATE TABLE
postgres=#
```

10.6.2、使用 mls_admin 用户配置表列加密规则

```
[postgres@VM_0_3_centos pgxzconf]$ psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=> select MLS_TRANSPARENT_CRYPT_ALGORITHM_BIND_TABLE('public', 'security_tb_col',
'test_b', 1);
 mls_transparent_crypt_algorithm_bind_table
-----
 t
(1 row)
```

```
postgres=> select MLS_TRANSPARENT_CRYPT_ALGORITHM_BIND_TABLE('public', 'security_tb_col',
'test_c', 1);
 mls_transparent_crypt_algorithm_bind_table
-----
 t
(1 row)
```

```
postgres=> select MLS_TRANSPARENT_CRYPT_ALGORITHM_BIND_TABLE('public', 'security_tb_col',
'test_d', 2);
 mls_transparent_crypt_algorithm_bind_table
-----
 t
(1 row)
```

函数说明

```
MLS_TRANSPARENT_CRYPT_ALGORITHM_BIND_TABLE(schema_name name,
                                             table_name name,
                                             attr_name name,
```

针对一个表（需要携带 schema）的一个字段，绑定特定的加密算法。

不支持分布列加密：

```
ERROR:  distribute key could not bind crypt, public.security_tb_col attrname:test_a
CONTEXT:  PL/pgSQL function mls_transparent_crypt_algorithm_bind_table(name,name,name,integer) line 36
at RAISE
```

配置加密规则的表不能存在记录，否则提示出错如下：

```
ERROR:  table_name:public.security_tb_col is not empty
CONTEXT:  PL/pgSQL function mls_transparent_crypt_algorithm_bind_table(name,name,name,integer) line 43
at RAISE
postgres=>
```

10.6.3、使用 mls_admin 用户删除表列加密规则

```
psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=> select MLS_TRANSPARENT_CRYPT_ALGORITHM_UNBIND_TABLE('public', 'security_tb_col',
'test_b');
 mls_transparent_crypt_algorithm_unbind_table
-----
 t
(1 row)
```

```
postgres=> select MLS_TRANSPARENT_CRYPT_ALGORITHM_UNBIND_TABLE('public', 'security_tb_col',
'test_c');
 mls_transparent_crypt_algorithm_unbind_table
-----
 t
(1 row)
```

函数说明

```
MLS_TRANSPARENT_CRYPT_ALGORITHM_UNBIND_TABLE(
schema_name name,
table_name name,
attr_name name)
```

针对一个表（需要携带 schema）的一个字段，解绑定特定的加密算法。**注：解绑前表数据要清掉，否则提示如下错误。**

```
ERROR:  table_name:public.security_tb_col is not empty, please TRUNCATE table and try again
```

CONTEXT: PL/pgSQL function mls_transparent_crypt_algorithm_unbind_table(name,name,name) line 28 at RAISE

10.6.4、查看那些表已经配置了加密规则

```
psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=> select * from pg_transparent_crypt_policy_map;
 relid | attnum | algorithm_id | spcoid | schemaoid | spcname | nspname | tblname
-----+-----+-----+-----+-----+-----+-----+-----
 50355 |      3 |             1 |   1663 |      2200 | pg_default | public | security_tb_col
 50355 |      2 |             1 |   1663 |      2200 | pg_default | public | security_tb_col
(2 rows)
```

10.6.5、数据插入演示

```
postgres=# insert into security_tb_col values(111,1111,1234567890,'this is CV','This is TX');
INSERT 0 1
postgres=# insert into security_tb_col values(222,2222,1234567890,'this is CV','This is TX');
INSERT 0 1
postgres=# select * from security_tb_col;
 test_a | test_b | test_c | test_d | test_e
-----+-----+-----+-----+-----
    222 |    2222 | 1234567890 | this is CV | This is TX
    111 |    1111 | 1234567890 | this is CV | This is TX
(2 rows)
```

```
postgres=# update security_tb_col set test_d='TBase';
UPDATE 2
postgres=# select * from security_tb_col;
 test_a | test_b | test_c | test_d | test_e
-----+-----+-----+-----+-----
    222 |    2222 | 1234567890 | TBase | This is TX
    111 |    1111 | 1234567890 | TBase | This is TX
(2 rows)
```

10.7、加密前后物理文件对比

```

00000000h: 00 00 00 00 E8 99 02 D3 00 00 00 00 00 10 28 00 ; ... 翎.?.....(.
00000010h: 10 1F 00 20 04 20 01 00 00 00 00 00 00 00 00 00 ; ... . . . . .
00000020h: 00 00 00 00 10 9F E0 01 64 05 00 00 C3 0D 04 07 ; .... 焄.d...?..
00000030h: 03 02 D9 48 25 35 F8 42 18 5A 7A D2 C0 83 01 50 ; .. 貶%5鳥.Zz依?P
00000040h: C0 D3 9D 03 41 86 42 4A F7 37 96 2B 38 E1 09 AD ; 烙?A味J??8??
00000050h: BB 9A 5E 7C F7 E6 2F FA 45 92 5C 8E 4B 0F 6D 70 ; 糴^|塵/鷄扑嶸.mp
00000060h: 78 8D 37 A8 03 D9 FE 88 A4 36 CF 10 57 03 19 A9 ; x??洗迨6?W..?
00000070h: 32 5F 61 32 08 25 93 8F D3 A4 88 26 CE 87 42 A2 ; 2_a2.%搨嬰?螞B?
00000080h: F9 04 AD 08 65 E2 97 15 8D 47 AD 84 9E 28 B9 AA ; ??e鉞.嶠瓌?躬
00000090h: CC BF 1B E1 39 1C 49 90 87 93 96 50 08 35 BD 97 ; 炭.?.I恪摻P.5緩

00000000h: 00 00 00 00 F8 15 05 D3 00 00 00 00 00 10 28 00 ; ....?.?.....(.
00000010h: A8 1F 00 20 04 20 00 00 00 00 00 00 00 00 00 00 ; ?. . . . .
00000020h: 00 00 00 00 A8 9F AC 00 00 00 00 00 00 00 00 00 00 ; ... ?.....
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00000060h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00000070h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....

```

加密过的	 17555
未加密的	 17564

10.8、加密管理函数

10.8.1、MLS_TRANSPARENT_CRYPT_SHOW

查看元数据信息，并标注有效和无效。同时对所有节点上的元数据做校验，并给出校验结果

```
postgres=> select MLS_TRANSPARENT_CRYPT_SHOW();
```

```
NOTICE: local node name:cn01
```

```
NOTICE: -----SHOW pg_transparent_crypt_policy_map-----
```

```
NOTICE: VALID RECORD: relid:50355, attnum:2, algorithm_id:1, spcname:pg_default, nspname:public, tblname:security_tb_col
```

```
NOTICE: VALID RECORD: relid:50355, attnum:3, algorithm_id:1, spcname:pg_default, nspname:public, tblname:security_tb_col
```

```
NOTICE: VALID RECORD: relid:50355, attnum:4, algorithm_id:2, spcname:pg_default, nspname:public, tblname:security_tb_col
```

```
NOTICE: -----COMPARE WITH OTHER NODES-----
```

```
mls_transparent_crypt_show
```

```
-----
t
(1 row)
```

10.8.2、MLS_TRANSPARENT_CRYPT_CLEAN

清理无效的元数据信息

```
postgres=> select MLS_TRANSPARENT_CRYPT_CLEAN();
NOTICE:  CLEAN INVALID MLS META DATA ON (cn01)
NOTICE:  CLEAN INVALID MLS META DATA ON (cn02)
NOTICE:  CLEAN INVALID MLS META DATA ON (dn01)
NOTICE:  CLEAN INVALID MLS META DATA ON (dn02)
NOTICE:  CLEAN INVALID MLS META DATA ON (dn03)
NOTICE:  CLEAN INVALID MLS META DATA ON (dn04)
```

```
mls_transparent_crypt_clean
-----
t
(1 row)
```

10.9、schema 加密配置

schema 加密配置是指对某个 schema 配置加密规则，然后只有在这个 schema 下创建的数据表，这个数据表只能配置与 schema 一样的加密规则

10.9.1、给一个 schema 配置加密规则

```
[tbase@VM_0_37_centos shell]$ psql -U tbase
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

--建立一个 schema

```
postgres=# create schema sch1;
CREATE SCHEMA
```

--分配权限让 mls_admin 安全管理员可访问

```
postgres=# grant USAGE on SCHEMA sch1 to mls_admin;
GRANT
```

--使用 mls_admin 安全管理员连接

```
postgres=# \c - mls_admin
```

You are now connected to database "postgres" as user "mls_admin".

--为模式 sch1 配置加密规则 1, 1 是前面已经创建好的加密规则, 创建加密规则见 10.4.1

```
postgres=> select MLS_TRANSPARENT_CRYPT_ALGORITHM_BIND_SCHEMA('sch1', 1);
 mls_transparent_crypt_algorithm_bind_schema
```

```
-----
t
(1 row)
```

10.9.2、查询那些 schema 配置加密规则

```
[tbase@VM_0_37_centos shell]$ psql -U mls_admin
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=>
--直接查询配置表
```

```
postgres=> select * from pg_transparent_crypt_policy_schema;
 schemaoid | algorithm_id | option
```

```
-----+-----+-----
114850 | 1 |
(1 row)
```

--关联显示 schema 名称

```
postgres=> select pg_transparent_crypt_policy_schema.*,pg_namespace.nspname from
pg_transparent_crypt_policy_schema,pg_namespace where
pg_transparent_crypt_policy_schema.schemaoid=pg_namespace.oid;
 schemaoid | algorithm_id | option | nspname
```

```
-----+-----+-----+-----
114850 | 1 | | sch1
(1 row)
```

10.9.3、测试规则是否生效

```
[tbase@VM_0_37_centos shell]$ psql -U tbase
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

--在模式 sch1 下建立数据表 t

```
postgres=# create table sch1.t(id int,mc text);
```

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

--使用 mls_admin 用户连接查看

```
postgres=# \c - mls_admin
```

You are now connected to database "postgres" as user "mls_admin".

```
postgres=> select * from pg_transparent_crypt_policy_map where spcname ='sch1';
 relid  | attnum | algorithm_id | spcoid | schemaoid | spcname | nspname | tblname
-----+-----+-----+-----+-----+-----+-----+-----
 114851 |   -32767 |           1 |   1663 |    114850 | sch1    | pg_default | t
(1 row)
```

```
postgres=>
```

10.10、数据脱敏

10.10.1、使用非 mls_admin 用户创建数据表

```
psql -h 127.0.0.1 -p 15432 -d postgres -U pgxz
```

psql (PostgreSQL 10.0 TBase V2)

Type "help" for help.

```
postgres=# create table test_table_mask_all(id int, f1 int2, f2 int4, f3 int8, f4 varchar, f5 text, f6 char(20), f7
varchar2, f8 numeric, f9 timestamp, f10 float4, f11 float8);
```

CREATE TABLE

```
postgres=#
```

插入数据

```
insert into test_table_mask_all(id,f1,f2,f3,f4,f5,f6,f7,f8, f9,f10,f11) values(1, 111, 222, 333, 'Tbase', 'Tbase',
'Tbase', 'Tbase', '123.123', 'now()';, '444.444', '888.888');
```

```
insert into test_table_mask_all(id,f1,f2,f3,f4,f5,f6,f7,f8, f9,f10,f11) values(2, 1111, 2222, 3333, 'TbaseTbase',
'TbaseTbase', 'TbaseTbase', 'TbaseTbase', '123.123', 'now()';, '444.444', '888.888');
```

```
insert into test_table_mask_all(id,f1,f2,f3,f4,f5,f6 ,f7,f8, f9,f10,f11) values(3, 11111, 22222, 33333,
'TbaseTbase~.~', 'TbaseTbase~.~', 'TbaseTbase~.~', 'TbaseTbase~.~', '123.123', 'now()';, '444.444', '888.888');
```

10.10.2、使用 mls_admin 用户配置脱敏规则

```
psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
```

psql (PostgreSQL 10.0 TBase V2)

Type "help" for help.

```
select MLS_DATAMASK_CREATE('public', 'test_table_mask_all', 'f1', 1);
```

```
select MLS_DATAMASK_CREATE('public', 'test_table_mask_all', 'f2', 1);
```

```

select MLS_DATAMASK_CREATE('public', 'test_table_mask_all', 'f3', 1);
select MLS_DATAMASK_CREATE('public', 'test_table_mask_all', 'f4', 2);
select MLS_DATAMASK_CREATE('public', 'test_table_mask_all', 'f5', 2);
select MLS_DATAMASK_CREATE('public', 'test_table_mask_all', 'f6', 2);
select MLS_DATAMASK_CREATE('public', 'test_table_mask_all', 'f7', 2);
select MLS_DATAMASK_CREATE('public', 'test_table_mask_all', 'f8', 3);
select MLS_DATAMASK_CREATE('public', 'test_table_mask_all', 'f9', 3, '2018-05-30 15:16:57');
select MLS_DATAMASK_CREATE('public', 'test_table_mask_all', 'f10', 3);
select MLS_DATAMASK_CREATE('public', 'test_table_mask_all', 'f11', 3);

```

函数说明：

针对一个表（需要携带 schema）的一个字段，面向具体用户需要呈现的脱敏形式，定义该脱敏规则。

当 optionval 值为 1 时，表示需要脱敏的数据类型为整数（int2、int4、int8），对应的 defaultval 为脱敏后的目标值。

当 optionval 值为 2 时，表示需要托名的数据类型为字符类型（varchar、text、char、varchar2），defaultval 表示的时脱敏显示时，从字符值的第一字符开始，使用 'X'（大写字母）进行脱敏显示。如果 defaultval 超过实际的数值长度，则显示为全 'X' 字符。

同理，当 optionval 值为 4 时，面向的字段类型和操作方式与 2 一致，唯一的区别是，从字符的末尾向前脱敏显示。

当 optionval 值为 3 时，表示需要脱敏显示的为指定的值，面向（numeric、timestamp、float4、float8）类型

10.10.3、使用非 mls_admin 用户查询脱敏后的数据

```

postgres=# select * from test_table_mask_all;

```

id	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11
1	9999	9999	9999	XXXXe	XXXXe	XXXXe	XXXXe	9999	2018-05-30 15:16:57	9999	9999
2	9999	9999	9999	XXXXeTbase	XXXXeTbase	XXXXeTbase	XXXXeTbase	9999	2018-05-30 15:16:57	9999	9999
3	9999	9999	9999	XXXXeTbase~	XXXXeTbase~	XXXXeTbase~	XXXXeTbase~	9999	2018-05-30 15:16:57	9999	9999

(3 rows)

10.10.4、配置脱敏白名单用户

```
psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
```

psql (PostgreSQL 10.0 TBase V2)

Type "help" for help.

```

select MLS_DATAMASK_CREATE_USER_POLICY('public', 'test_table_mask_all', 'f1', 'pgxz');
select MLS_DATAMASK_CREATE_USER_POLICY('public', 'test_table_mask_all', 'f2', 'pgxz');
select MLS_DATAMASK_CREATE_USER_POLICY('public', 'test_table_mask_all', 'f3', 'pgxz');
select MLS_DATAMASK_CREATE_USER_POLICY('public', 'test_table_mask_all', 'f4', 'pgxz');
select MLS_DATAMASK_CREATE_USER_POLICY('public', 'test_table_mask_all', 'f5', 'pgxz');
select MLS_DATAMASK_CREATE_USER_POLICY('public', 'test_table_mask_all', 'f6', 'pgxz');
select MLS_DATAMASK_CREATE_USER_POLICY('public', 'test_table_mask_all', 'f7', 'pgxz');
select MLS_DATAMASK_CREATE_USER_POLICY('public', 'test_table_mask_all', 'f8', 'pgxz');
select MLS_DATAMASK_CREATE_USER_POLICY('public', 'test_table_mask_all', 'f9', 'pgxz');
select MLS_DATAMASK_CREATE_USER_POLICY('public', 'test_table_mask_all', 'f10', 'pgxz');

```

```
select MLS_DATAMASK_CREATE_USER_POLICY('public', 'test_table_mask_all', 'f11', 'pgxz');
```

验证配置效果，用户 pgxz 看到的是没脱敏的，postgres 用户看到的是脱敏过的数据

```
[postgres@VM_0_29_centos data]$ psql -U pgxz
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.

postgres=# select * from test_table_mask_all;
 id | f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
  1 | 111 | 222 | 333 | Tbase | Tbase | Tbase | Tbase | 123.123 | 2018-06-20 21:21:48.298549 | 444.444 | 888.888
  3 | 11111 | 22222 | 33333 | TbaseTbase~.~ | TbaseTbase~.~ | TbaseTbase~.~ | TbaseTbase~.~ | 123.123 | 2018-06-20 21:21:49.006636 | 444.444 | 888.888
  2 | 1111 | 2222 | 3333 | TbaseTbase | TbaseTbase | TbaseTbase | TbaseTbase | 123.123 | 2018-06-20 21:21:48.449501 | 444.444 | 888.888
(3 rows)

postgres=# \q
[postgres@VM_0_29_centos data]$ psql -U postgres
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.

postgres=# select * from test_table_mask_all;
 id | f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
  3 | 9999 | 9999 | 9999 | XXXXeTbase~.~ | XXXXeTbase~.~ | XXXXeTbase~.~ | XXXXeTbase~.~ | 9999 | 2018-05-30 15:16:57 | 9999 | 9999
  1 | 9999 | 9999 | 9999 | XXXXe | XXXXe | XXXXe | XXXXe | 9999 | 2018-05-30 15:16:57 | 9999 | 9999
  2 | 9999 | 9999 | 9999 | XXXXeTbase | XXXXeTbase | XXXXeTbase | XXXXeTbase | 9999 | 2018-05-30 15:16:57 | 9999 | 9999
(3 rows)

postgres=#
```

10.10.5、删除脱敏白名单用户

```
psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
select MLS_DATAMASK_DROP_USER_POLICY('public', 'test_table_mask_all', 'f1', 'pgxz');
select MLS_DATAMASK_DROP_USER_POLICY('public', 'test_table_mask_all', 'f2', 'pgxz');
select MLS_DATAMASK_DROP_USER_POLICY('public', 'test_table_mask_all', 'f3', 'pgxz');
```

验证配置效果，f1，f2，f3 是脱敏过的数据

```
[postgres@VM_0_29_centos data]$ psql -U pgxz
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.

postgres=# select * from test_table_mask_all;
 id | f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
  3 | 9999 | 9999 | 9999 | TbaseTbase~.~ | TbaseTbase~.~ | TbaseTbase~.~ | TbaseTbase~.~ | 123.123 | 2018-06-20 21:21:49.006636 | 444.444 | 888.888
  1 | 9999 | 9999 | 9999 | Tbase | Tbase | Tbase | Tbase | 123.123 | 2018-06-20 21:21:48.298549 | 444.444 | 888.888
  2 | 9999 | 9999 | 9999 | TbaseTbase | TbaseTbase | TbaseTbase | TbaseTbase | 123.123 | 2018-06-20 21:21:48.449501 | 444.444 | 888.888
(3 rows)
```

10.10.6、禁用用户白名单

```
psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=> select MLS_DATAMASK_DISABLE_USER_POLICY('public', 'test_table_mask_all', 'f4', 'pgxz');
 mls_datamask_disable_user_policy
```

```
-----
 t
(1 row)
```

验证配置效果

```
[postgres@VM_0_29_centos pgxzconf]$ psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.

postgres=> select MLS_DATAMASK_DISABLE_USER_POLICY('public', 'test_table_mask_all', 'f4','pgxz');
 mls_datamask_disable_user_policy
-----
 t
(1 row)

postgres=> \c - pgxz
You are now connected to database "postgres" as user "pgxz".
postgres=# select * from test_table_mask_all;
 id | f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11
----+---+---+---+---+---+---+---+---+---+---+
  1 | 111 | 222 | 333 | XXXXe | Tbase | Tbase | Tbase | 123.123 | 2018-06-20 21:21:48.298549 | 444.444 | 888.888
  2 | 1111 | 2222 | 3333 | XXXXeTbase | TbaseTbase | TbaseTbase | TbaseTbase | 123.123 | 2018-06-20 21:21:48.449501 | 444.444 | 888.888
  3 | 11111 | 22222 | 33333 | XXXXeTbase~ | TbaseTbase~ | TbaseTbase~ | TbaseTbase~ | 123.123 | 2018-06-20 21:21:49.006636 | 444.444 | 888.888
(3 rows)
```

10.10.7、启用用户白名单

```
psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
```

```
psql (PostgreSQL 10.0 TBase V2)
```

```
Type "help" for help.
```

```
postgres=> select MLS_DATAMASK_ENABLE_USER_POLICY('public', 'test_table_mask_all', 'f4','pgxz');
 mls_datamask_enable_user_policy
-----
```

```
 t
(1 row)
```

验证配置效果

```
[postgres@VM_0_29_centos pgxzconf]$ psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.

postgres=> select MLS_DATAMASK_ENABLE_USER_POLICY('public', 'test_table_mask_all', 'f4','pgxz');
 mls_datamask_enable_user_policy
-----
 t
(1 row)

postgres=> \c - pgxz
You are now connected to database "postgres" as user "pgxz".
postgres=# select * from test_table_mask_all;
 id | f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11
----+---+---+---+---+---+---+---+---+---+---+
  3 | 11111 | 22222 | 33333 | TbaseTbase~ | TbaseTbase~ | TbaseTbase~ | TbaseTbase~ | 123.123 | 2018-06-20 21:21:49.006636 | 444.444 | 888.888
  1 | 111 | 222 | 333 | Tbase | Tbase | Tbase | Tbase | 123.123 | 2018-06-20 21:21:48.298549 | 444.444 | 888.888
  2 | 1111 | 2222 | 3333 | TbaseTbase | TbaseTbase | TbaseTbase | TbaseTbase | 123.123 | 2018-06-20 21:21:48.449501 | 444.444 | 888.888
(3 rows)
```

10.10.8、更新脱敏规则

```
[postgres@VM_0_3_centos root]$ psql -d postgres -U mls_admin
```

```
psql (PostgreSQL 10.0 TBase V2)
```

```
Type "help" for help.
```

```
postgres=> select MLS_DATAMASK_UPDATE_TABLE_POLICY('public', 'test_table_mask_all', 'f1', 666);
```

```
mls_datamask_update_table_policy
```

```
-----
t
(1 row)
```

```
postgres=> select MLS_DATAMASK_UPDATE_TABLE_POLICY('public', 'test_table_mask_all', 'f2', 777);
select MLS_DATAMASK_UPDATE_TABLE_POLICY('public', 'test_table_mask_all', 'f3', 888);
select    MLS_DATAMASK_UPDATE_TABLE_POLICY('public',    'test_table_mask_all',    'f4',    6);
mls_datamask_update_table_policy
```

```
-----
t
(1 row)
```

```
postgres=> select MLS_DATAMASK_UPDATE_TABLE_POLICY('public', 'test_table_mask_all', 'f3', 888);
mls_datamask_update_table_policy
```

```
-----
t
(1 row)
```

```
postgres=> select MLS_DATAMASK_UPDATE_TABLE_POLICY('public', 'test_table_mask_all', 'f4', 6);
mls_datamask_update_table_policy
```

```
-----
t
(1 row)
```

```
postgres=>
```

验证查询结果

```
[postgres@VM_0_29_centos root]$ psql -U postgres
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.

postgres=# select * from test_table_mask_all;
 id | f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
  3 | 666 | 777 | 888 | XXXXXbase~.~ | XXXXeTbase~.~ | XXXXeTbase~.~ | XXXXeTbase~.~ | 9999 | 2018-05-30 15:16:57 | 9999 | 9999
  1 | 666 | 777 | 888 | XXXXX | XXXXe | XXXXe | XXXXe | 9999 | 2018-05-30 15:16:57 | 9999 | 9999
  2 | 666 | 777 | 888 | XXXXXbase | XXXXeTbase | XXXXeTbase | XXXXeTbase | 9999 | 2018-05-30 15:16:57 | 9999 | 9999
(3 rows)
```

10.10.9、删除脱敏规则

```
[postgres@VM_0_3_centos root]$ psql -d postgres -U mls_admin
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=> select * from pg_data_mask_map order by attnum;
 relid | attnum | enable | option | datamask | nspname | tblname | maskfunc | defaultval
-----+-----+-----+-----+-----+-----+-----+-----+-----
 50377 |      2 | t      |      1 | 666 | public | test_table_mask_all | 0 | 9999
```

```

50377 |      3 | t      |      1 |      777 | public | test_table_mask_all |      0 | 9999
50377 |      4 | t      |      1 |      888 | public | test_table_mask_all |      0 | 9999
50377 |      5 | t      |      2 |         6 | public | test_table_mask_all |      0 | 9999
50377 |      6 | t      |      2 |         4 | public | test_table_mask_all |      0 | 9999
50377 |      7 | t      |      2 |         4 | public | test_table_mask_all |      0 | 9999
50377 |      8 | t      |      2 |         4 | public | test_table_mask_all |      0 | 9999
50377 |      9 | t      |      3 |      9999 | public | test_table_mask_all |      0 | 9999
50377 |     10 | t      |      3 |         0 | public | test_table_mask_all |      0 | 2018-05-30 15:16:57
50377 |     11 | t      |      3 |      9999 | public | test_table_mask_all |      0 | 9999
50377 |     12 | t      |      3 |      9999 | public | test_table_mask_all |      0 | 9999
(11 rows)

```

```

postgres=> select MLS_DATAMASK_DROP_TABLE_POLICY('public', 'test_table_mask_all', 'f1');
 mls_datamask_drop_table_policy

```

```

-----
t
(1 row)

```

```

postgres=> select * from pg_data_mask_map order by attnum;

```

relid	attnum	enable	option	datamask	nspname	tblname	maskfunc	defaultval
50377	3	t		1	777 public	test_table_mask_all	0 9999	
50377	4	t		1	888 public	test_table_mask_all	0 9999	
50377	5	t		2	6 public	test_table_mask_all	0 9999	
50377	6	t		2	4 public	test_table_mask_all	0 9999	
50377	7	t		2	4 public	test_table_mask_all	0 9999	
50377	8	t		2	4 public	test_table_mask_all	0 9999	
50377	9	t		3	9999 public	test_table_mask_all	0 9999	
50377	10	t		3	0 public	test_table_mask_all	0 2018-05-30 15:16:57	
50377	11	t		3	9999 public	test_table_mask_all	0 9999	
50377	12	t		3	9999 public	test_table_mask_all	0 9999	

(10 rows)

```

postgres=>

```

10.10.10、查看已经配置那些脱敏规则

```

psql -h 127.0.0.1 -p 15432 -d postgres -U mls_admin
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.

```

```

postgres=> select * from pg_data_mask_map;

```

relid	attnum	enable	option	datamask	nspname	tblname	maskfunc	defaultval
50377	3	t		1	9999 public	test_table_mask_all	0 9999	
50377	4	t		1	9999 public	test_table_mask_all	0 9999	

```

50377 |      5 | t      |      2 |      4 | public | test_table_mask_all |      0 | 9999
50377 |      6 | t      |      2 |      4 | public | test_table_mask_all |      0 | 9999
50377 |      7 | t      |      2 |      4 | public | test_table_mask_all |      0 | 9999
50377 |      8 | t      |      2 |      4 | public | test_table_mask_all |      0 | 9999
50377 |      9 | t      |      3 |     9999 | public | test_table_mask_all |      0 | 9999
50377 |     10 | t      |      3 |      0 | public | test_table_mask_all |      0 | 2018-05-30 15:16:57
50377 |     11 | t      |      3 |     9999 | public | test_table_mask_all |      0 | 9999
50377 |     12 | t      |      3 |     9999 | public | test_table_mask_all |      0 | 9999

```

(10 rows)

postgres=>

10.10.11、查看已经配置了那些用户白名单

```

psql -d postgres -U mls_admin
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.

```

```

postgres=> select * from pg_data_mask_user;
 relid | userid | attnum | enable | username | nspname |      tblname
-----+-----+-----+-----+-----+-----+-----
50377 | 16967 |      9 | t      | pgxz     | public  | test_table_mask_all
50377 | 16967 |     10 | t      | pgxz     | public  | test_table_mask_all
50377 | 16967 |     11 | t      | pgxz     | public  | test_table_mask_all
50377 | 16967 |     12 | t      | pgxz     | public  | test_table_mask_all
50377 | 16967 |      5 | t      | pgxz     | public  | test_table_mask_all
50377 | 16967 |      6 | t      | pgxz     | public  | test_table_mask_all
50377 | 16967 |      7 | t      | pgxz     | public  | test_table_mask_all
50377 | 16967 |      8 | t      | pgxz     | public  | test_table_mask_all

```

(8 rows)

postgres=>

10.10.12、数据脱敏管理函数

10.10.12.1、MLS_DATAMASK_CLEAN

清理无效的元数据信息

```

psql -d postgres -U mls_admin
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.

```

```

postgres=> select MLS_DATAMASK_CLEAN();

```

```
NOTICE: CLEAN INVALID MLS META DATA ON (cn01)
NOTICE: CLEAN INVALID MLS META DATA ON (cn02)
NOTICE: CLEAN INVALID MLS META DATA ON (dn01)
NOTICE: CLEAN INVALID MLS META DATA ON (dn02)
NOTICE: CLEAN INVALID MLS META DATA ON (dn03)
NOTICE: CLEAN INVALID MLS META DATA ON (dn04)
```

```
mls_datamask_clean
```

```
-----
t
(1 row)
```

10.10.12.2、MLS_DATAMASK_SHOW

展示 datamask 的元数据信息，并在节点之间校验

```
psql -d postgres -U mls_admin
```

```
psql (PostgreSQL 10.0 TBase V2)
```

```
Type "help" for help.
```

```
postgres=> select * from MLS_DATAMASK_SHOW();
```

```
NOTICE: local node name:cn01
```

```
NOTICE: -----SHOW pg_data_mask_map-----
```

```
NOTICE: VALID RECORD: relid:50377, attnum:3, enable:t, option:1, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, attnum:4, enable:t, option:1, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, attnum:5, enable:t, option:2, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, attnum:6, enable:t, option:2, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, attnum:7, enable:t, option:2, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, attnum:8, enable:t, option:2, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, attnum:9, enable:t, option:3, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, attnum:10, enable:t, option:3, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, attnum:11, enable:t, option:3, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, attnum:12, enable:t, option:3, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: -----SHOW pg_data_mask_user-----
```

```
NOTICE: VALID RECORD: relid:50377, userid:16967, attnum:5, enable:t, username:pgxz, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, userid:16967, attnum:6, enable:t, username:pgxz, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, userid:16967, attnum:7, enable:t, username:pgxz, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, userid:16967, attnum:8, enable:t, username:pgxz, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, userid:16967, attnum:9, enable:t, username:pgxz, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, userid:16967, attnum:10, enable:t, username:pgxz, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, userid:16967, attnum:11, enable:t, username:pgxz, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: VALID RECORD: relid:50377, userid:16967, attnum:12, enable:t, username:pgxz, nspname:public, tblname:test_table_mask_all
```

```
NOTICE: -----COMPARE WITH OTHER NODES-----
```

```
mls_datamask_show
```

```
-----
t
(1 row)
```

11、用户及权限管理

11.1、用户管理

11.1.1、添加一个普通用户

连接到某个 cn 节点

```
[tbase@VM_0_29_centos pgxz]$ psql -h 172.16.0.29 -U tbase -d postgres -p 15432
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=# create role user1 with login password 'user1@123';
```

11.1.2、给用户添加一个 schema

连接到某个 cn 节点

```
[tbase@VM_0_29_centos pgxz]$ psql -h 172.16.0.29 -U tbase -d postgres -p 15432
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=# create schema user1 AUTHORIZATION user1;
```

11.1.3、添加一个管理员用户

连接到某个 cn 节点

```
[tbase@VM_0_29_centos pgxz]$ psql -h 172.16.0.29 -U tbase -d postgres -p 15432
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=# create role super_user1 with login password 'super_user1@123' superuser;
CREATE ROLE
postgres=#
```

11.2、权限管理

11.2.1、给用户分配模式使用权限

连接到某个 cn 节点

```
[tbase@VM_0_29_centos pgxz]$ psql -h 172.16.0.29 -U tbase -d postgres -p 15432
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=# grant usage on SCHEMA mysch to user1;
GRANT
postgres=#
```

默认情况下，普通用户是无法访问没授权的 `schema`，所以要授权用户访问某个表的访问权限，则需要将表所在的 `schema` 使用权分配给用户先。

11.2.2、给用户分配表使用权限

连接到某个 cn 节点

```
[tbase@VM_0_29_centos pgxz]$ psql -h 172.16.0.29 -U tbase -d postgres -p 15432
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=# grant SELECT on mysch.t1 to user1;
GRANT
postgres=#
```

增，删，改，查分别对应 INSERT/DELETE/UPDATE/SELECT，如果需要全部权限，则可以这样写成 `all`

```
postgres=# grant ALL on mysch.t1 to user1;
GRANT
postgres=#
```

11.2.3、将某个模式下所有表权限分配给某个用户

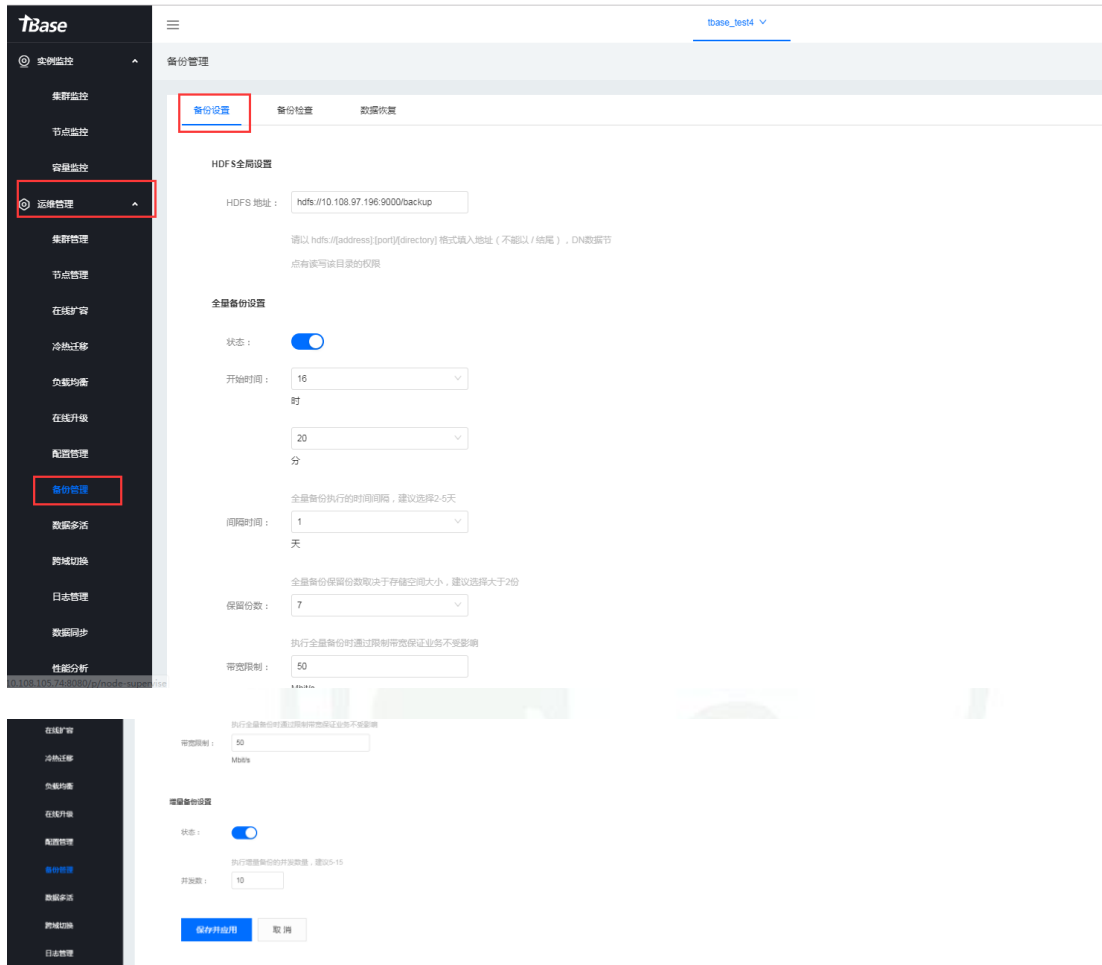
```
[tbase@VM_0_29_centos pgxz]$ psql -h 172.16.0.29 -U tbase -d postgres -p 15432
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=# grant SELECT ON ALL TABLES IN SCHEMA mysch TO user1;
GRANT
```

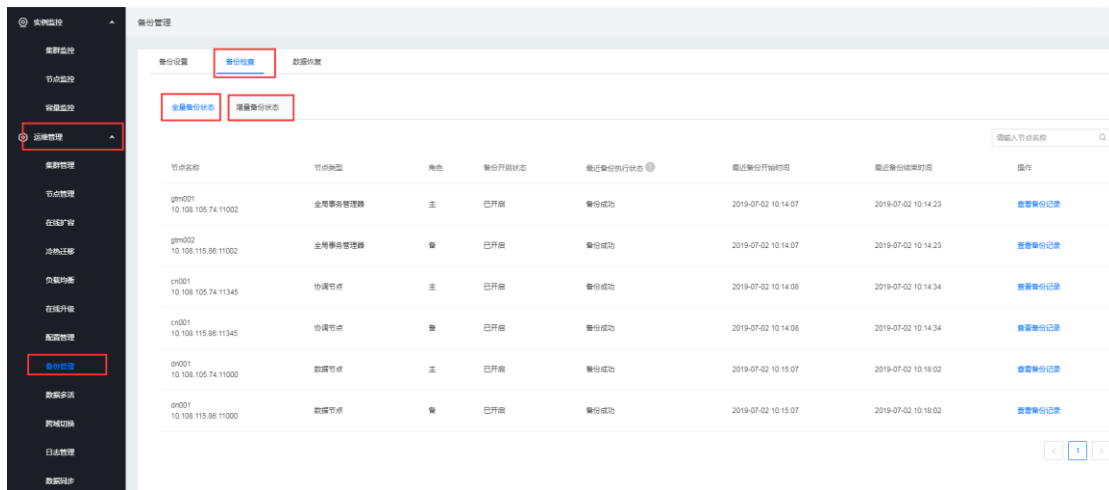
12、冷备及恢复流程

12.1、冷备流程

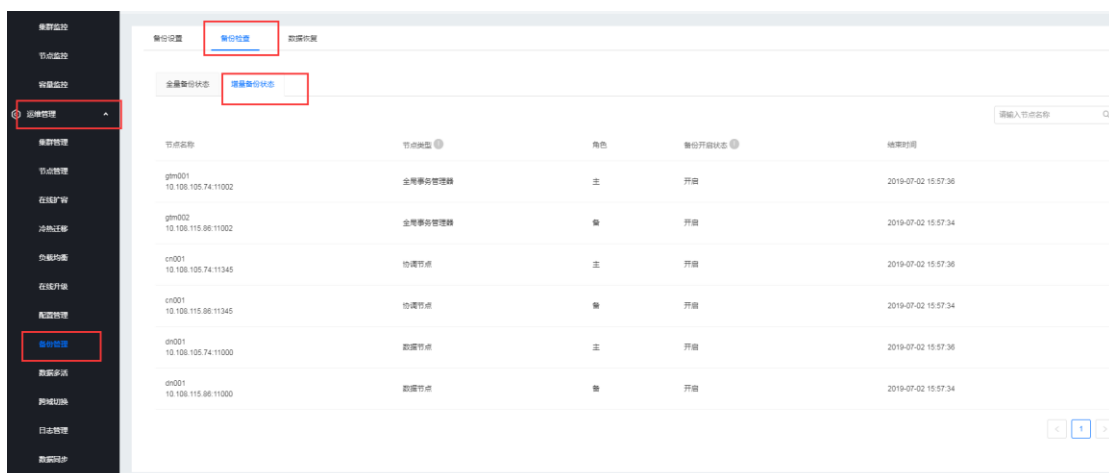
1. 使用集群管理员帐号登陆 oss，在运维管理->备份管理->备份设置中进行设置备份设置 HDFS 地址，打开全量备份及增量备份，点击保存并应用



2. 在到达配置时间后，会自动进行冷备，GTM 在主上备份，其他节点优先在备机上备份，在页面上可以看到备份状态（全量备份状态）



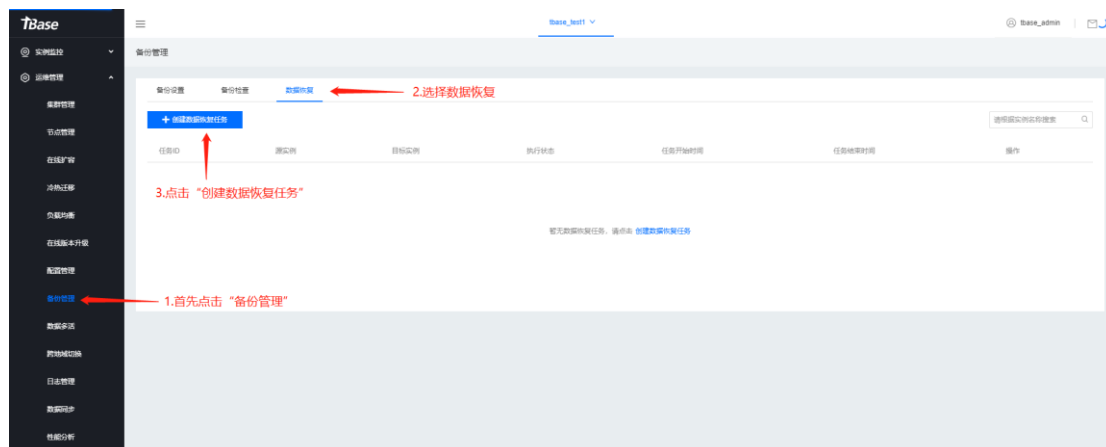
(增量备份状态)



12.2、恢复流程

在有可用的冷备后，可以进行恢复流程，具体流程如下：

1. 创建数据恢复任务



2. 选择想恢复到的时间，然后在后台返回的多个确切时间点中选择真正恢复到的精确时间

创建数据恢复任务

1 选择数据恢复时间 2 配置临时集群

* 时间选择: 2019-07-02 17:27:16

* 可恢复时间点: 2019-07-02 17:11:48, 2019-07-02 17:11:39

4. 选择想恢复到的理想时间

5. 选择理想时间附近的一个一致性时间点

6. 点击“下一步”开始配置临时集群

取消 下一步

3. 配置恢复的临时集群信息

创建数据恢复任务

1 选择数据恢复时间 2 配置临时集群

* 实例名称: tbase_recovery

全局事务管理器

* 节点模板: common (1核 CPU / 1G 内存 / 1G 磁盘)

协调节点

* 节点模板: common (1核 CPU / 1G 内存 / 1G 磁盘)

数据节点

* 节点模板: common (1核 CPU / 1G 内存 / 1G 磁盘)

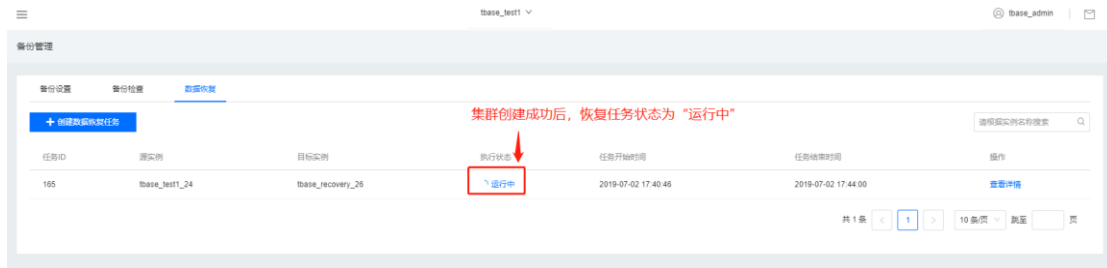
7. 配置恢复集群的实例名

8. 配置恢复集群中 gtm/cn/dn 的资源模板

9. 点击“创建按钮”开始创建临时集群, 并进入恢复流程

上一步 创建

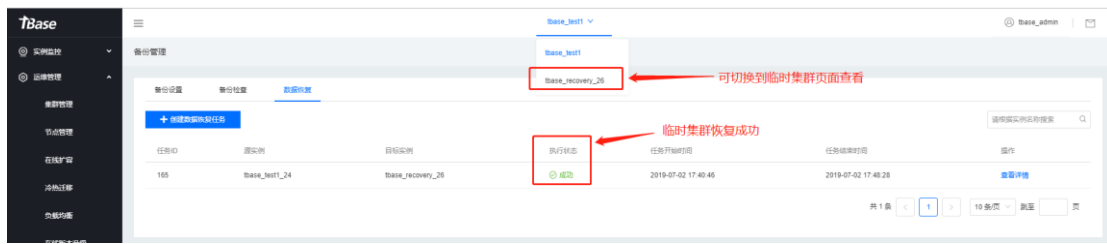
4. 查看恢复任务



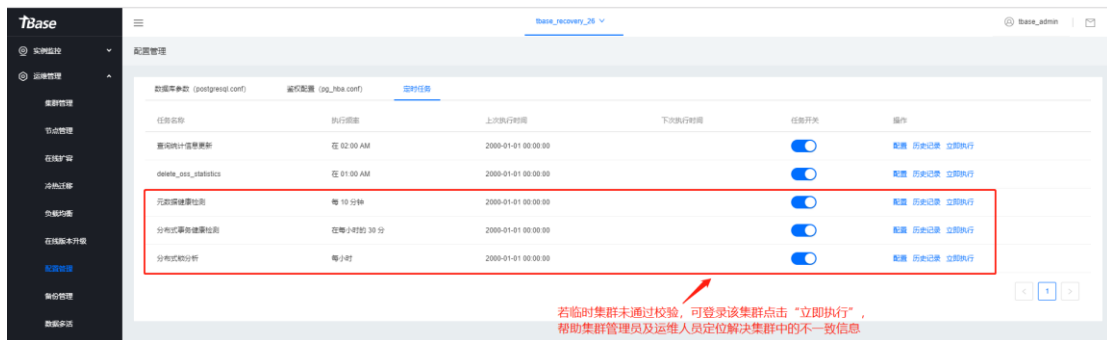
5. 集群恢复完成 (详情页信息)



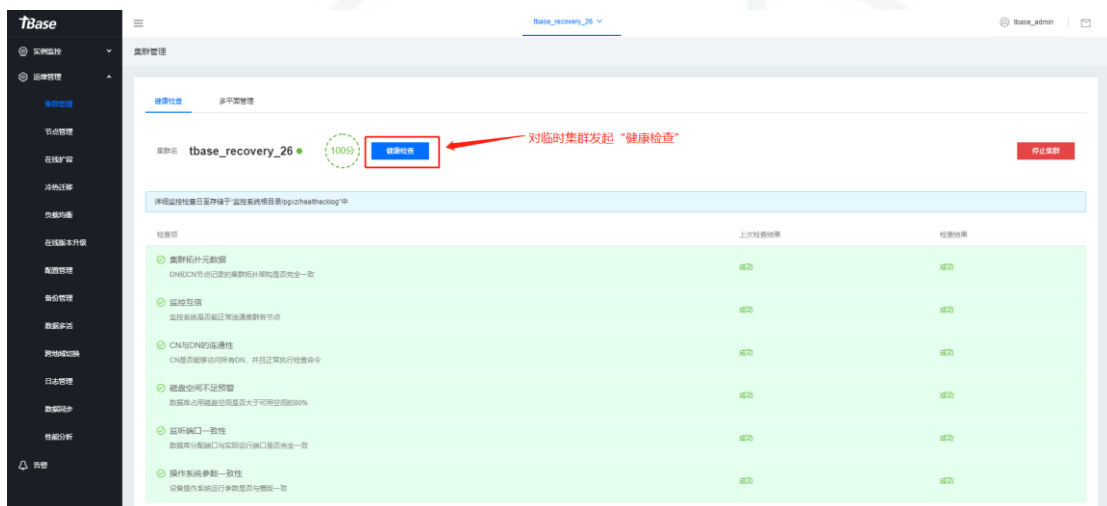
(任务状态信息)



6. 若临时集群未通过一致性校验，可在“配置管理”→“定时任务”页面手动触发一致性校验



7. 可查看临时集群进行健康检查



13、常见故障问题的分析与处理

13.1 锁等待原因分析与监控

问题描述:

Postgresql 通过锁以及多版本并发控制可以保护数据的一致性，例如 A 用户正在查询数据，B 用户就无法对 A 用户访问的对象执行 DDL（alter、truncate 等）。A 用户正在更新某条记录，B 就不能删除或更新这条记录。

当在发生这些锁的时候会出现以下几种现象：

- 1、用户无法对表进行 select、insert、delete 等操作将会失效，只会等待锁关闭之后，才能正常执行。
- 2、如果应用程序逻辑设计不慎，就可能导致严重的锁等待，或者死锁的产生。

问题分析:

如何寻找这些锁？

TBase 提供两个视图：

1. pg_locks 展示锁信息，每一个被锁或者等待锁的对象一条记录。

```
select * from pg_locks;
```

2. pg_stat_activity，每个会话一条记录，显示会话状态信息。 我们通过这两个视图可以查看锁，锁等待情况。同时可以了解发生锁冲突的情况。

```
select * from pg_stat_activity;
```

pg_stat_activity 中 query 字段内容反映的是当前正在执行或请求的 SQL，而同一个事务中以前已经执行的 SQL 不能在 pg_stat_activity 中显示出来。

所以如果你发现两个会话发生了冲突，但是他们的 pg_stat_activity.query 没有冲突的话，那就有可能是他们之间的某个事务之前的 SQL 获取的锁与另一个事务当前请求的 QUERY 发生了锁冲突。

追踪详细的锁冲突信息:

- 1、通过数据库日志(开启 lock_timeout, log_lockwait 参数)(csvlog)跟踪锁等待信息，
2. 或者通过数据库日志(开启 log_statements='all', SQL 审计)追踪事务中所有的 SQL (csvlog)，分析事务之间的锁冲突。
3. 通过 SQL 查看持锁，等锁的事务状态。

锁的释放时机：大多数锁要等待事务结束后释放，某些轻量级锁(数据库自动控制)是随用随释放的。

问题解决:

为了方便观察，创建锁监控视图：

```
create view v_locks_monitor as with t_wait as ( select
a.mode,a.locktype,a.database,a.relation,a.page,a.tuple,a.classid,a.granted,
a.objid,a.objsubid,a.pid,a.virtualtransaction,a.virtualxid,a.transactionid,a.fastpath,
b.state,b.query,b.xact_start,b.query_start,b.username,b.datname,b.client_addr,b.client_port,b.application_name
from pg_locks a,pg_stat_activity b where a.pid=b.pid and not a.granted ), t_run as ( select
a.mode,a.locktype,a.database,a.relation,a.page,a.tuple,a.classid,a.granted,
a.objid,a.objsubid,a.pid,a.virtualtransaction,a.virtualxid,a.transactionid,a.fastpath,
b.state,b.query,b.xact_start,b.query_start,b.username,b.datname,b.client_addr,b.client_port,b.application_name
from pg_locks a,pg_stat_activity b where a.pid=b.pid and a.granted ), t_overlap as ( select r.* from t_wait w join
t_run r on ( r.locktype is not distinct from w.locktype and r.database is not distinct from w.database and r.relation
is not distinct from w.relation and r.page is not distinct from w.page and r.tuple is not distinct from w.tuple and
r.virtualxid is not distinct from w.virtualxid and r.transactionid is not distinct from w.transactionid and r.classid is
not distinct from w.classid and r.objid is not distinct from w.objid and r.objsubid is not distinct from w.objsubid
and r.pid <> w.pid ) ), t_unionall as ( select r.* from t_overlap r union all select w.* from t_wait w ) select
locktype,datname,relation::regclass,page,tuple,virtualxid,transactionid::text,classid::regclass,objid,objsubid,
string_agg( 'Pid: '||case when pid is null then 'NULL' else pid::text end||chr(10)|| 'Lock_Granted: '||case when
granted is null then 'NULL' else granted::text end||' , Mode: '||case when mode is null then 'NULL' else mode::text
end||' , FastPath: '||case when fastpath is null then 'NULL' else fastpath::text end||' , VirtualTransaction: '||case when
virtualtransaction is null then 'NULL' else virtualtransaction::text end||' , Session_State: '||case when state is null
then 'NULL' else state::text end||chr(10)|| 'Username: '||case when username is null then 'NULL' else username::text
end||' , Database: '||case when datname is null then 'NULL' else datname::text end||' , Client_Addr: '||case when
client_addr is null then 'NULL' else client_addr::text end||' , Client_Port: '||case when client_port is null then
'NULL' else client_port::text end||' , Application_Name: '||case when application_name is null then 'NULL' else
application_name::text end||chr(10)|| 'Xact_Start: '||case when xact_start is null then 'NULL' else xact_start::text
end||' , Query_Start: '||case when query_start is null then 'NULL' else query_start::text end||' , Xact_Elapse: '||case
when (now()-xact_start) is null then 'NULL' else (now()-xact_start)::text end||' , Query_Elapse: '||case when
(now()-query_start) is null then 'NULL' else (now()-query_start)::text end||chr(10)|| 'SQL (Current SQL in
Transaction): '||chr(10)|| case when query is null then 'NULL' else query::text end, chr(10)||'-----'||chr(10) order
by ( case mode when 'INVALID' then 0 when 'AccessShareLock' then 1 when 'RowShareLock' then 2 when
'RowExclusiveLock' then 3 when 'ShareUpdateExclusiveLock' then 4 when 'ShareLock' then 5 when
'ShareRowExclusiveLock' then 6 when 'ExclusiveLock' then 7 when 'AccessExclusiveLock' then 8 else 0 end )
desc, (case when granted then 0 else 1 end) ) as lock_conflict from t_unionall group by
locktype,datname,relation,page,tuple,virtualxid,transactionid::text,classid,objid,objsubid ;
```

查询建立好视图之后：

```
postgres=# select * from v_locks_monitor ;
```

1 在 string_agg 中找到 pid 以及用户的 sql，然后确认 sql 语句是否和现在的 insert、select、delete、update、copy、truncate 产生冲突后，导致表锁。

2 找到对应 sql 的 pid 之后。

前面的锁查询 SQL，已经清晰的显示了每一个发生了锁等待的对象，按锁的大小排序，要快速解出这种状态，terminate 最大的锁对应的 PID 即可。

```
postgres=# select pg_terminate_backend(PID);
```

3 在查询一下是否还有表锁：

```
postgres=# select * from v_locks_monitor ;
```

4、检查表的 ddl 是否正常。

到此完成处理。

13.2 SQL terminate 终止进程（强制停止）

问题描述:

当计算超大型任务时和类似于一个千万级表直接 `select * from tablename` 导致，数据不断的导出，IO 会一直处于一个持续上升和内存会被用完，会带来的主要影响：

- 1、主备同步的时候由于压力的问题会导致主备同步超时等问题。
- 2、严重情况下可能会导致 coordinator 节点 OOM，内存溢出挂死。

问题分析:

产生情况的原因主要是用户的 SQL 过于复杂或者产生了一些笛卡尔积导致，由于 SQL 的复杂度可能没有经过优化，导致执行的时候事务超时很长，消耗的资源比较多。

问题解决:

- 1、首先我们先从 `pg_stat_activity` 视图中寻找相关的 SQL: `select pid,* from pg_stat_activity;`
- 2、使用 `pg_cancel_backend(pid)` 执行取消，这种方式只能 kill select 查询，对 update、delete 及 DML 不生效。
- 3、使用 `pg_terminate_backend(pid)` 执行杀死进程，这种方式可以 kill 掉各种操作(select、update、delete、drop 等)操作。

以上操作可以多执行几次，直到进程被完全杀死。

如果出现 SQL 无法杀死或者无法取消执行的情况：

- 1、一般是用户事务正在执行批量事务，这个时候关闭需要用户配合停止业务程序后，然后杀死命令。
- 2、检查有没有两阶段（2PC）事务事务残留，使用 `pg_clean` 插件进行相关检测，并处理 2PC 事务。

13.3 物理块损坏排查

问题描述:

用户在做 A 和 B 表关联操作时，出现错误：`missing chunk number x for toast value x in pg_toast_x`
该问题的主要原因是：某张表关联的 toast 表的数据发生损坏。

问题分析:

该问题出现频率比较低，一般原因是硬盘故障和机房断点或者服务器强制重启导致。

- 1、定位是哪张表的 toast 有问题？
- 2、找到哪个表有问题后，先对该表做一下简单的修复。
- 3、定位该表中损坏的数据行。
- 4、用户确认数据是否可以补回后，将第 3 步中定位的记录删除

5、重复执行第 3,4 步，直到全部有问题的记录被清除。

问题解决：

1、首先定位 toast 表：

```
postgres=# select 2619::regclass;
      regclass
-----
pg_statistic
```

2、找到哪个表有问题后，先对该表做一下简单的修复：

重建索引和重新 analyze 表。

```
REINDEX table pg_toast.pg_toast_2619;
REINDEX table pg_statistic;
VACUUM ANALYZE pg_statistic;
```

3、定位该表中损坏的数据行

```
DO $$
declare
    v_rec record;
BEGIN
    for v_rec in SELECT * FROM pg_statistic loop
        raise notice 'Parameter is: %', v_rec.ctid;
        raise notice 'Parameter is: %', v_rec;
    end loop;
END;
$$

LANGUAGE plpgsql;
```

3、将第 3 步中定位的记录删除：

4、delete from pg_statistic where ctid = '(50,3)';

重复执行第 3,4 步，直到全部有问题的记录被清除。

处理完 toast 问题后对该表进行一次 analyze table;

13.4 使用驱动接口从 TBase 抽取数据内存溢出

问题描述：

在做数据仓的时候，难免需要从库中大量的抽出数据到其它数据源，那么这个过程中肯定会将数据全部的抽出，数据有可能太大导致服务端 OOM 异常，这里的数据源案例使用的 Java 如何快速的从 TBase 中抽取数据，并防止内存堆栈溢出。

问题分析：

- 1、既然一次不能拿太多的数据，那就分批次去拿数据，每次只拿一定量的数据进行处理后，再去取另外一批。
- 2、由于 postgresql 默认是自动提交事务，如果要分批次，那么必须关闭自动提交事务 autocommit。

问题解决:

用户如果有大量抽取 Tbase 数据的时候, 需要对一些 ETL 工具进行相关的改造, 以下提供 java 的例子:

- 1、在创建连接的时候需要关闭自动提交事务: `conn.setAutoCommit(false);`
- 2、`PreparedStatement ps = conn.prepareStatement(sql);`
`ps.setFetchSize(1000);` //每次获取 1 千条记录 可以根据实际情况进行调配
每次从服务端拉取 1000 条记录, 防止数据过多一下子将内存承包。

13.5 重做备机主机出现: Too many open files in system

问题描述:

重做备机主机出现: Too many open files in system, 导致 dn 拉不起来, 因为 pg 表的 blocksize 默认是 8k, 因此小文件产生过多, 以致于超过了系统参数限制。

问题分析:

直观报错现象只有通过 linux 内核参数调整优化。

问题解决:

```
vim /etc/sysctl.conf
将 fs.file-max 调整为 6553600
```

13.6 如何判断 idle in transaction 的事务中有没有东西要提交

问题描述:

程序没有关闭连接, 或者正在执行一个事务, 但是事务失败, 既没有回滚和提交的语句众多, 这个时候我们就需要去判断中间过程中有没有事务提交。

问题分析:

- 1、需要清理掉一些事务时, 如何让他优雅的退出, 或者如何快速的清退那些没有什么影响的事务呢?
- 2、需要判断 idle in transaction 的事务中有没有东西要提交?

问题解决:

通过 `pg_stat_activity` 会话判断:

`pg_stat_activity` 会话中有两个字段, `backend_xid`, `backend_xmin` 用来表示会话是否申请了事务号, 以及会话的快照 ID。

情况一: `pg_stat_activity` 中查到的会话的 `backend_xid` 和 `backend_xmin` 都没有值。因为它没有对数据库有任何写操作所以不需要申请事务号, 因此 `backend_xid` 为空

情况二: `ackend_xid` 和 `backend_xmin` 都有值了, 写入操作会申请事务号, 因此 `backend_xid` 不为空。
通过 `pg_stat_activity.backend_xid` 不是为空的, 即说明这个事务有东西要提交。

14 不常见参数调优

14.1 max_stack_depath 参数

问题出现前期：大批量 insert 和 copy 的时候，JDBC 驱动中抛出异常：

```
[执行]执行过程出现异常，准备进入异常处理！,异常消息[org.postgresql.util.PSQLException: ERROR: stack depth limit exceeded  
建议：Increase the configuration parameter "max_stack_depth" (currently 2048kB), after ensuring the platform's stack depth limit is  
adequate.
```

max_stack_depath 参数的大小远远大于了默认的值，导致事务执行失败。

参数作用：决定一个数据库进程在运行时的 STACK 所占的空间的最大值。数据库进程在运行时，会自动检查自己的 STACK 大小是否超过 max_stack_depth，如果超过，会自动终止当前事务。这个值应该比操作系统设置的进程 STACK 的大小的 上限小 1MB。

问题解决：

1、首先查看 OS 中 stack 的大小，ulimit -s 单位是 KB，默认值是 100，建议增大该值到 16384

方法一：使用命令：ulimit -s 16384 进行设置，不过这只是临时设置，重启后失效

方法二：永久设置栈空间大小无限制：在/etc/profile 的最后面添加 ulimit -s unlimited，然后执行 source /etc/profile 使修改文件生效。

2、然后调整 postgresql.conf 中该参数的值，增加该参数值：max_stack_depth = 8MB 需要重启 TBase 集群才能生效。

