Overview

PostgreSQL进阶之路第一章体系架构概述

https://docs.google.com/document/d/126wj6Qo788QlXiotCBTmaVyICUI4_bEIRh5pZFV6stg/edit?tab=t.0

该文档是学习 PostgreSQL进阶之路 教材（https://bbs.cherry-creek.net/thread-1.htm）第一章的学习笔记。针对所有的实验添加了截图和必要的注释。查阅了所有感兴趣的源代码以及相应的截图。这并且针对个别PG数据库参数，列举对应的数据库SQL查询语句，Linux Shell命令，以及对应的截图。参数，查询语句，命令返回结果，和源代码之间的相互对应，可以帮助有效的加深PG内核的理解。所有实验基于以下文档配置的虚拟机（https://docs.google.com/document/d/1t1rH2vOpXD7mU_Hbuy-yqBhufN8nd0K9hiS-JdBXKVU/edit?tab=t.0）。

第一章 体系架构概述

1.1 创建实验环境

创建实验环境，可以参考一下文档。
PostgreSQL 17.5 在Ubuntu 24.0.2 x64 上的安装（macOS arm64 + UTM）

1.2 体系架构概括

编写一段简单的C语言的源代码，编译该源代码，进一步理解什么是进程。

编写一段简单的C语言的源代码。
vi hello.c
#include <stdio.h>

```
#include <unistd.h>
int main(int argc, char* argv[])
{
    printf("Hello, World!\n");
    sleep(60); /* sleep for 60 seconds */
    return 0;
}
```

```
postgres@pg-50:~$ vi hello.c
postgres@pg-50:~$
```

```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
    printf("Hello, World!\n");
    sleep(60); /* sleep for 60 seconds */
    return 0;
}
~
~
~
~
~
~
~
~
~
~
~
~
~
"hello.c" 9L, 164B                                                    9,0-1         All
```

编译该源代码。生成可执行文件。
gcc -Wall hello.c -o hello

```
postgres@pg-50:~$ gcc -Wall hello.c -o hello
postgres@pg-50:~$ ls -ltr
total 21132
-rw-rw-r--  1 postgres postgres 21595174 May  5 20:41 postgresql-17.5.tar.bz2
drwxrwxr-x  6 postgres postgres     4096 Jun  4 18:37 postgresql-17.5
drwxrwxr-x  6 postgres postgres     4096 Jun  4 19:22 pg17.5
-rw-rw-r--  1 postgres postgres      168 Jun  4 19:38 set.env
drwx------ 19 postgres postgres     4096 Jun  4 20:00 data1
-rw-------  1 postgres postgres     2611 Jun  4 20:00 logfile
-rw-rw-r--  1 postgres postgres      164 Jun  4 21:24 hello.c
-rwxrwxr-x  1 postgres postgres    16000 Jun  4 21:29 hello
postgres@pg-50:~$
```

运行编译好的可执行文件。
./hello

```
postgres@pg-50:~$ ./hello
Hello, World!
```

然后打开另外一个会话(session)，查看该进程。进程号是1286，父进程号是1076.

```
postgres@pg-50:~$ ps -ef | grep hello
postgres    1286    1076  0 21:31 pts/0    00:00:00 ./hello
postgres    1289    1274 33 21:31 pts/1    00:00:00 grep --color=auto hello
postgres@pg-50:~$
```

所有PG的所有进程。

```
postgres@pg-50:~$ ps -ef | grep postgres
root         960     959  0 20:47 ?        00:00:00 sshd: postgres [priv]
postgres     966       1  0 20:47 ?        00:00:01 /usr/lib/systemd/systemd --user
postgres     967     966  0 20:47 ?        00:00:00 (sd-pam)
postgres    1075     960  0 20:47 ?        00:00:05 sshd: postgres@pts/0
postgres    1076    1075  0 20:47 pts/0    00:00:00 -bash
root        1217     959  0 21:31 ?        00:00:00 sshd: postgres [priv]
postgres    1273    1217  0 21:31 ?        00:00:03 sshd: postgres@pts/1
postgres    1274    1273  0 21:31 pts/1    00:00:00 -bash
postgres    1315       1  2 21:38 ?        00:00:00 /home/postgres/pg17.5/bin/postgres
postgres    1316    1315  0 21:38 ?        00:00:00 postgres: checkpointer
postgres    1317    1315  0 21:38 ?        00:00:00 postgres: background writer
postgres    1319    1315  0 21:38 ?        00:00:00 postgres: walwriter
postgres    1320    1315  0 21:38 ?        00:00:00 postgres: autovacuum launcher
postgres    1321    1315  0 21:38 ?        00:00:00 postgres: logical replication launcher
postgres    1324    1274 99 21:38 pts/1    00:00:00 ps -ef
postgres    1325    1274 25 21:38 pts/1    00:00:00 grep --color=auto postgres
postgres@pg-50:~$
```

理解进程树的概念。左边是右边的父进程。

pstree -p 1315

```
postgres@pg-50:~$ pstree -p 1315
postgres(1315)─┬─postgres(1316)
               ├─postgres(1317)
               ├─postgres(1319)
               ├─postgres(1320)
               └─postgres(1321)
postgres@pg-50:~$
```

启动 PG客户端。查看客户端的进程和后端进程。

```
postgres@pg-50:~$ ps -ef | grep postgres
root         960     959  0 20:47 ?        00:00:00 sshd: postgres [priv]
postgres     966       1  0 20:47 ?        00:00:01 /usr/lib/systemd/systemd --user
postgres     967     966  0 20:47 ?        00:00:00 (sd-pam)
postgres    1075     960  0 20:47 ?        00:00:06 sshd: postgres@pts/0
postgres    1076    1075  0 20:47 pts/0    00:00:00 -bash
root        1217     959  0 21:31 ?        00:00:00 sshd: postgres [priv]
postgres    1273    1217  0 21:31 ?        00:00:07 sshd: postgres@pts/1
postgres    1274    1273  0 21:31 pts/1    00:00:00 -bash
postgres    1315       1  0 21:38 ?        00:00:00 /home/postgres/pg17.5/bin/postgres
postgres    1316    1315  0 21:38 ?        00:00:00 postgres: checkpointer
postgres    1317    1315  0 21:38 ?        00:00:00 postgres: background writer
postgres    1319    1315  0 21:38 ?        00:00:00 postgres: walwriter
postgres    1320    1315  0 21:38 ?        00:00:00 postgres: autovacuum launcher
postgres    1321    1315  0 21:38 ?        00:00:00 postgres: logical replication launcher
postgres    1362    1076  0 21:42 pts/0    00:00:00 psql
postgres    1421    1315  0 21:50 ?        00:00:00 postgres: postgres oracle [local] idle
postgres    1433    1274 99 21:52 pts/1    00:00:00 ps -ef
postgres    1434    1274  0 21:52 pts/1    00:00:00 grep --color=auto postgres
postgres@pg-50:~$ ps -ef | grep psql
postgres    1362    1076  0 21:42 pts/0    00:00:00 psql
postgres    1437    1274 33 21:52 pts/1    00:00:00 grep --color=auto psql
postgres@pg-50:~$
```

检查进程间通讯的共享内存的使用大小。

```
postgres@pg-50:~$ ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x001c0737 0          postgres   600        56         6

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

postgres@pg-50:~$
```

检查虚拟内存的使用大小。包括虚拟内存大小和实际使用的物理内存大小。
ps -ef | grep postgres

```
postgres@pg-50:~$ ps -ef | grep postgres
root         960     959  0 20:47 ?        00:00:00 sshd: postgres [priv]
postgres     966       1  0 20:47 ?        00:00:01 /usr/lib/systemd/systemd --user
postgres     967     966  0 20:47 ?        00:00:00 (sd-pam)
postgres    1075     960  0 20:47 ?        00:00:12 sshd: postgres@pts/0
postgres    1076    1075  0 20:47 pts/0    00:00:00 -bash
root        1217     959  0 21:31 ?        00:00:00 sshd: postgres [priv]
postgres    1273    1217  0 21:31 ?        00:00:07 sshd: postgres@pts/1
postgres    1274    1273  0 21:31 pts/1    00:00:00 -bash
postgres    1315       1  0 21:38 ?        00:00:02 /home/postgres/pg17.5/bin/postgres
postgres    1316    1315  0 21:38 ?        00:00:00 postgres: checkpointer
postgres    1317    1315  0 21:38 ?        00:00:00 postgres: background writer
postgres    1319    1315  0 21:38 ?        00:00:00 postgres: walwriter
postgres    1320    1315  0 21:38 ?        00:00:00 postgres: autovacuum launcher
postgres    1321    1315  0 21:38 ?        00:00:00 postgres: logical replication launcher
postgres    1822    1076 99 23:14 pts/0    00:00:00 ps -ef
postgres    1823    1076 25 23:14 pts/0    00:00:00 grep --color=auto postgres
postgres@pg-50:~$ pmap -x 1315
1315:   /home/postgres/pg17.5/bin/postgres
Address           Kbytes     RSS   Dirty Mode  Mapping
00005dd82e91b000     860     844       0 r---- postgres
00005dd82e9f2000    5536    2368       0 r-x-- postgres
00005dd82ef5a000    3060     732       0 r---- postgres
```

这里可以看到 PG虚拟内存使用是200MB，但是实际物理内存使用了21MB。

```
000075e1f13d2000     508     248       0 r-x-- libm.so.6
000075e1f1451000     352       0       0 r---- libm.so.6
000075e1f14a9000       4       4       4 r---- libm.so.6
000075e1f14aa000       4       4       4 rw--- libm.so.6
000075e1f14ab000       8       8       0 r---- libz.so.1.3
000075e1f14ad000      72      64       0 r-x-- libz.so.1.3
000075e1f14bf000      24       0       0 r---- libz.so.1.3
000075e1f14c5000       4       4       4 r---- libz.so.1.3
000075e1f14c6000       4       4       4 rw--- libz.so.1.3
000075e1f14cc000       4       4       4 rw-s-   [ shmid=0x0 ]
000075e1f14cd000       8       8       8 rw---   [ anon ]
000075e1f14cf000       4       4       0 r---- ld-linux-x86-64.so.2
000075e1f14d0000     172     172       0 r-x-- ld-linux-x86-64.so.2
000075e1f14fb000      40      40       0 r---- ld-linux-x86-64.so.2
000075e1f1505000       8       8       8 r---- ld-linux-x86-64.so.2
000075e1f1507000       8       8       8 rw--- ld-linux-x86-64.so.2
00007ffcc91b0000     132      36      36 rw---   [ stack ]
00007ffcc91f6000      16       0       0 r----   [ anon ]
00007ffcc91fa000       8       4       0 r-x--   [ anon ]
ffffffffff600000       4       0       0 --x--   [ anon ]
---------------- ------- ------- -------
total kB          201568   21704   11864
postgres@pg-50:~$
```

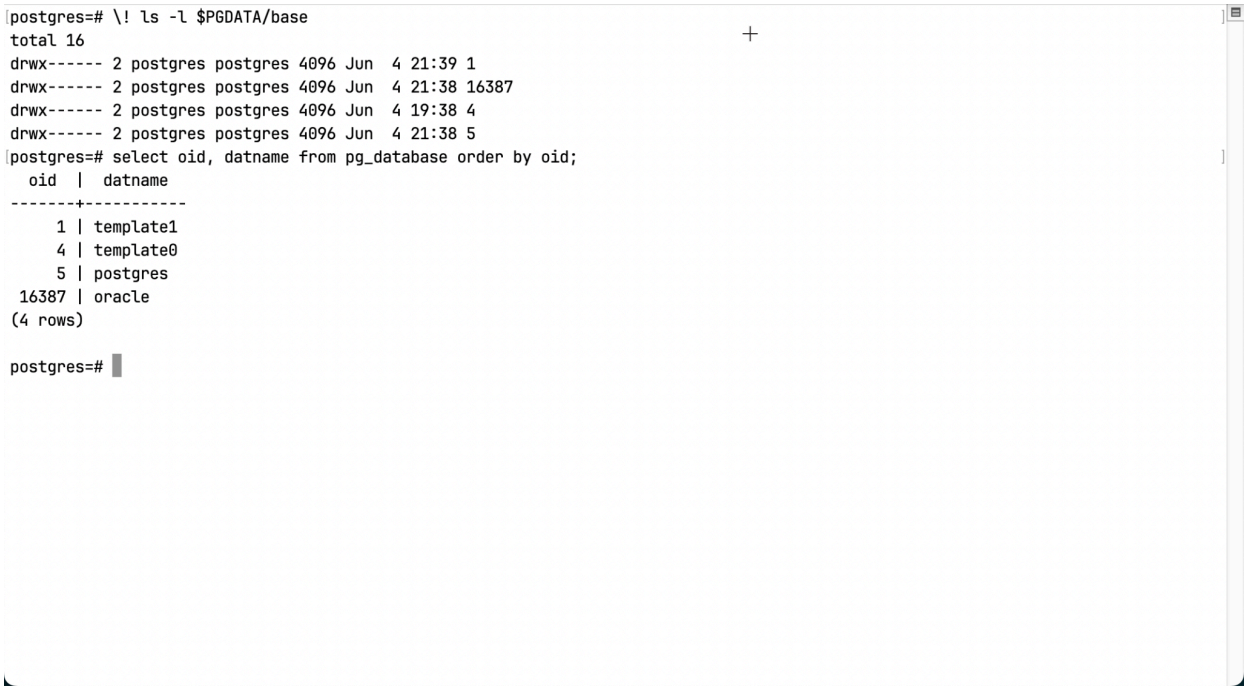
PG的主要的目录结构。

```
postgres@pg-50:~/data1$ pwd
/home/postgres/data1
postgres@pg-50:~/data1$ ls -l
total 128
drwx------ 6 postgres postgres  4096 Jun  4 19:53 base
drwx------ 2 postgres postgres  4096 Jun  4 21:39 global
drwx------ 2 postgres postgres  4096 Jun  4 19:38 pg_commit_ts
drwx------ 2 postgres postgres  4096 Jun  4 19:38 pg_dynshmem
-rw------- 1 postgres postgres  5711 Jun  4 19:38 pg_hba.conf
-rw------- 1 postgres postgres  2640 Jun  4 19:38 pg_ident.conf
drwx------ 4 postgres postgres  4096 Jun  4 21:43 pg_logical
drwx------ 4 postgres postgres  4096 Jun  4 19:38 pg_multixact
drwx------ 2 postgres postgres  4096 Jun  4 19:38 pg_notify
drwx------ 2 postgres postgres  4096 Jun  4 19:38 pg_replslot
drwx------ 2 postgres postgres  4096 Jun  4 19:38 pg_serial
drwx------ 2 postgres postgres  4096 Jun  4 19:38 pg_snapshots
drwx------ 2 postgres postgres  4096 Jun  4 21:38 pg_stat
drwx------ 2 postgres postgres  4096 Jun  4 19:38 pg_stat_tmp
drwx------ 2 postgres postgres  4096 Jun  4 19:38 pg_subtrans
drwx------ 2 postgres postgres  4096 Jun  4 19:38 pg_tblspc
drwx------ 2 postgres postgres  4096 Jun  4 19:38 pg_twophase
-rw------- 1 postgres postgres     3 Jun  4 19:38 PG_VERSION
drwx------ 4 postgres postgres  4096 Jun  4 19:38 pg_wal
drwx------ 2 postgres postgres  4096 Jun  4 19:38 pg_xact
-rw------- 1 postgres postgres    88 Jun  4 19:38 postgresql.auto.conf
-rw------- 1 postgres postgres 30703 Jun  4 19:38 postgresql.conf
-rw------- 1 postgres postgres    35 Jun  4 21:38 postmaster.opts
-rw------- 1 postgres postgres    86 Jun  4 21:38 postmaster.pid
postgres@pg-50:~/data1$
```

PG主要目录和配置文件的用途。
base：PG所有数据库的数据文件。
global：PG全局系统表的数据文件。
pg_tblspc：PG表空间符号连接信息。
pg_wal：PG日志文件。

pg_xact：PG事务文件。
postgresql.conf：PG服务器端配置文件。
pg_hba.conf：PG客户端配置文件。
postmaster.pid：PG主进程的进程号。防止重复启动。
PG_VERSION：PG版本号。

base目录存放了 PG所有数据库的数据文件。每一个数据库有一个单独的子目录。每一个字目录的目录名用对象标识符（oid）表示。在数据库中，可以查看对象标识符和数据库名之间的对应关系。
select oid, datname from pg_database order by oid;



PG中架构的层次关系。注意实例在某些语境里称作集群。
Server → Instance/Cluster → Database → Schema → Table → Column

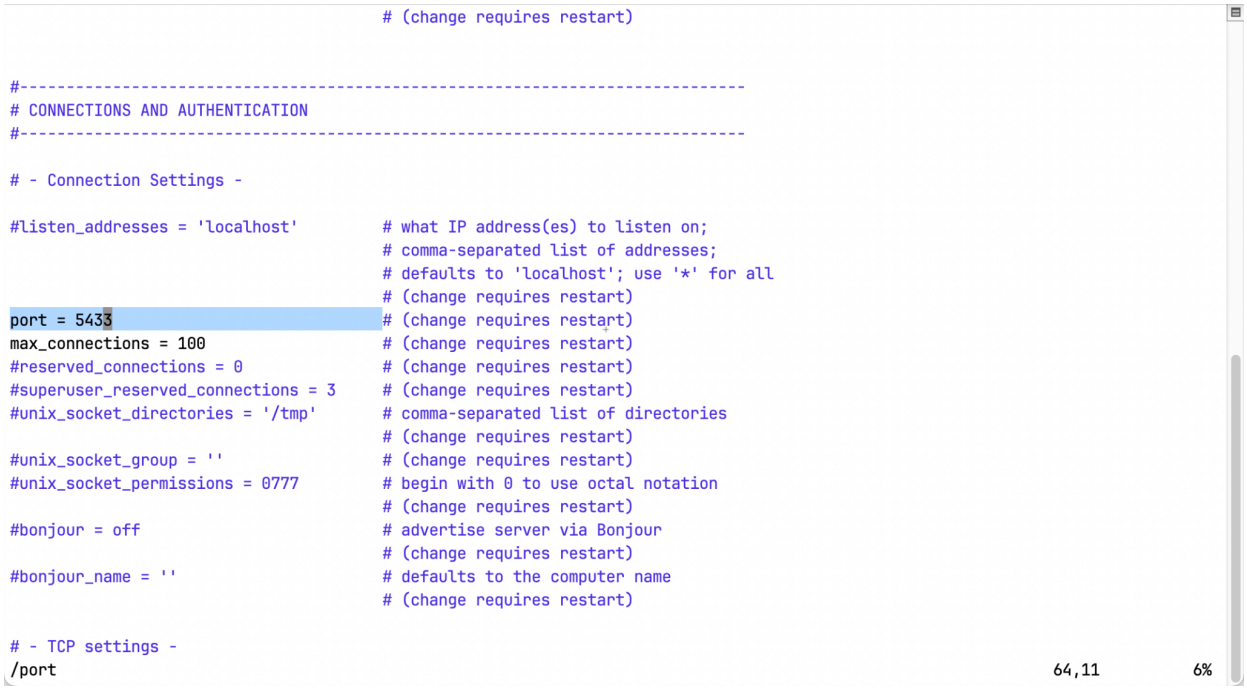
下面实验如何在同一个服务器(Server)上运行多个PG实例。

创建另外一个数据库实例。
initdb -D /home/postgres/data2

```
postgres@pg-50:~$ initdb -D /home/postgres/data2
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "en_US.UTF-8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are disabled.

creating directory /home/postgres/data2 ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default "max_connections" ... 100
selecting default "shared_buffers" ... 128MB
selecting default time zone ... Etc/UTC
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok

initdb: warning: enabling "trust" authentication for local connections
initdb: hint: You can change this by editing pg_hba.conf or using the option -A, or --auth-local and --auth-host, the next time yo
u run initdb.

Success. You can now start the database server using:

    pg_ctl -D /home/postgres/data2 -l logfile start
```

修改PG的监听端口。防止多个 PG实例运行时的监听端口冲突。这里第一个PG实例使用默认的监听端口5432，第二个PG实例的监听端口修改为5433。

```
postgres@pg-50:~/data2$ pwd
/home/postgres/data2
postgres@pg-50:~/data2$ vi postgresql.conf
```



```
                                # (change requires restart)


#------------------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------------

# - Connection Settings -

#listen_addresses = 'localhost'         # what IP address(es) to listen on;
                                        # comma-separated list of addresses;
                                        # defaults to 'localhost'; use '*' for all
                                        # (change requires restart)
port = 5433                             # (change requires restart)
max_connections = 100                   # (change requires restart)
#reserved_connections = 0               # (change requires restart)
#superuser_reserved_connections = 3     # (change requires restart)
#unix_socket_directories = '/tmp'       # comma-separated list of directories
                                        # (change requires restart)
#unix_socket_group = ''                 # (change requires restart)
#unix_socket_permissions = 0777         # begin with 0 to use octal notation
                                        # (change requires restart)
#bonjour = off                          # advertise server via Bonjour
                                        # (change requires restart)
#bonjour_name = ''                      # defaults to the computer name
                                        # (change requires restart)

# - TCP settings -
/port                                                                   64,11         6%
```

启动第二个PG实例。

```
postgres@pg-50:~/data2$ pg_ctl start -D /home/postgres/data2 -l logfile2
waiting for server to start.... done
server started
```

两个PG实例的运行状态。

```
postgres@pg-50:~/data2$ pg_ctl status -D /home/postgres/data1
pg_ctl: server is running (PID: 1315)
/home/postgres/pg17.5/bin/postgres
postgres@pg-50:~/data2$ pg_ctl status -D /home/postgres/data2
pg_ctl: server is running (PID: 2344)
/home/postgres/pg17.5/bin/postgres "-D" "/home/postgres/data2"
postgres@pg-50:~/data2$
```

查看两个PG实例在操作系统层面的两组进程。

```
postgres@pg-50:~/data2$ ps -ef | grep postgres
root         960     959  0 Jun04 ?        00:00:00 sshd: postgres [priv]
postgres     966       1  0 Jun04 ?        00:00:01 /usr/lib/systemd/systemd --user
postgres     967     966  0 Jun04 ?        00:00:00 (sd-pam)
postgres    1075     960  0 Jun04 ?        00:00:42 sshd: postgres@pts/0
postgres    1076    1075  0 Jun04 pts/0    00:00:02 -bash
root        1217     959  0 Jun04 ?        00:00:00 sshd: postgres [priv]
postgres    1273    1217  0 Jun04 ?        00:00:22 sshd: postgres@pts/1
postgres    1274    1273  0 Jun04 pts/1    00:00:00 -bash
postgres    1315       1  0 Jun04 ?        00:00:04 /home/postgres/pg17.5/bin/postgres
postgres    1316    1315  0 Jun04 ?        00:00:00 postgres: checkpointer
postgres    1317    1315  0 Jun04 ?        00:00:01 postgres: background writer
postgres    1319    1315  0 Jun04 ?        00:00:01 postgres: walwriter
postgres    1320    1315  0 Jun04 ?        00:00:01 postgres: autovacuum launcher
postgres    1321    1315  0 Jun04 ?        00:00:00 postgres: logical replication launcher
postgres    2344       1  0 00:40 ?        00:00:00 /home/postgres/pg17.5/bin/postgres -D /home/postgres/data2
postgres    2345    2344  0 00:40 ?        00:00:00 postgres: checkpointer
postgres    2346    2344  0 00:40 ?        00:00:00 postgres: background writer
postgres    2348    2344  0 00:40 ?        00:00:00 postgres: walwriter
postgres    2349    2344  0 00:40 ?        00:00:00 postgres: autovacuum launcher
postgres    2350    2344  0 00:40 ?        00:00:00 postgres: logical replication launcher
postgres    2373    1076 99 00:43 pts/0    00:00:00 ps -ef
postgres    2374    1076 60 00:43 pts/0    00:00:00 grep --color=auto postgres
postgres@pg-50:~/data2$
```

```
postgres@pg-50:~$ pstree -p 1315
postgres(1315)─┬─postgres(1316)
               ├─postgres(1317)
               ├─postgres(1319)
               ├─postgres(1320)
               └─postgres(1321)
postgres@pg-50:~$ pstree -p 2344
postgres(2344)─┬─postgres(2345)
               ├─postgres(2346)
               ├─postgres(2348)
               ├─postgres(2349)
               └─postgres(2350)
postgres@pg-50:~$
```

使用PG客户端，通过不同的PG实例的监听端口，连接不同的PG实例。

```
postgres@pg-50:~$ ss -tulnp | grep postgres
tcp   LISTEN 0      200        127.0.0.1:5433      0.0.0.0:*    users:(("postgres",pid=2344,fd=6))
tcp   LISTEN 0      200        127.0.0.1:5432      0.0.0.0:*    users:(("postgres",pid=1315,fd=6))
postgres@pg-50:~$ psql -p 5432
psql (17.5)
Type "help" for help.

postgres=# \l
                                                  List of databases
   Name    |  Owner   | Encoding | Locale Provider |   Collate   |    Ctype    | Locale | ICU Rules |   Access privileges
-----------+----------+----------+-----------------+-------------+-------------+--------+-----------+-----------------------
 oracle    | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           |
 postgres  | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           |
 template0 | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           | =c/postgres          +
           |          |          |                 |             |             |        |           | postgres=CTc/postgres
 template1 | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           | =c/postgres          +
           |          |          |                 |             |             |        |           | postgres=CTc/postgres
(4 rows)

postgres=# \q
postgres@pg-50:~$ psql -p 5433
psql (17.5)
Type "help" for help.

postgres=# \l
                                                  List of databases
   Name    |  Owner   | Encoding | Locale Provider |   Collate   |    Ctype    | Locale | ICU Rules |   Access privileges
-----------+----------+----------+-----------------+-------------+-------------+--------+-----------+-----------------------
 postgres  | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |        |           |
```

PG中的表空间是目录的一个别名。默认包括两个表空间 pg_default 和 pg_global。分别指向 $PGDATA/base 和 $PGDATA/global。分别用来存储默认数据库文件和全局系统表文件。

```
postgres=# SELECT * FROM pg_tablespace;
  oid  |  spcname   | spcowner | spcacl | spcoptions
-------+------------+----------+--------+------------
  1663 | pg_default |       10 |        |
  1664 | pg_global  |       10 |        |
(2 rows)

postgres=# \! ls -l $PGDATA/base
total 16
drwx------ 2 postgres postgres 4096 Jun  4 21:39 1
drwx------ 2 postgres postgres 4096 Jun  4 21:38 16387
drwx------ 2 postgres postgres 4096 Jun  4 19:38 4
drwx------ 2 postgres postgres 4096 Jun  4 21:38 5
postgres=# \! ls -l $PGDATA/pg_tblspc
total 0
postgres=# \! ls -l $PGDATA/global
total 572
-rw------- 1 postgres postgres  8192 Jun  4 19:38 1213
-rw------- 1 postgres postgres 24576 Jun  4 19:38 1213_fsm
-rw------- 1 postgres postgres  8192 Jun  4 19:38 1213_vm
-rw------- 1 postgres postgres     0 Jun  4 19:38 1214
-rw------- 1 postgres postgres  8192 Jun  4 19:38 1232
-rw------- 1 postgres postgres  8192 Jun  4 19:38 1233
-rw------- 1 postgres postgres  8192 Jun  4 19:38 1260
-rw------- 1 postgres postgres 24576 Jun  4 19:38 1260_fsm
-rw------- 1 postgres postgres  8192 Jun  4 19:38 1260_vm
-rw------- 1 postgres postgres  8192 Jun  4 19:38 1261
-rw------- 1 postgres postgres 24576 Jun  4 19:38 1261_fsm
-rw------- 1 postgres postgres  8192 Jun  4 19:38 1261_vm
```

接下来我们创建一个额外的表空间。可以看到我们新创建的一个表空间 mytbs，指向 /home/postgres/data1_2 。并且在 pg_tbspc 创建了一个符号链接指向新的目录。

```
postgres@pg-50:~$ mkdir /home/postgres/data1_2
postgres@pg-50:~$ ls -l /home/postgres/data1_2
total 0
postgres@pg-50:~$ psql
psql (17.5)
Type "help" for help.

postgres=# create tablespace mytbs location '/home/postgres/data1_2';
CREATE TABLESPACE
postgres=# SELECT * FROM pg_tablespace;
  oid  |  spcname   | spcowner | spcacl | spcoptions
-------+------------+----------+--------+------------
  1663 | pg_default |       10 |        |
  1664 | pg_global  |       10 |        |
 16393 | mytbs      |       10 |        |
(3 rows)

postgres=# SELECT oid, spcname, pg_tablespace_location(oid) FROM pg_tablespace;
  oid  |  spcname   | pg_tablespace_location
-------+------------+------------------------
  1663 | pg_default |
  1664 | pg_global  |
 16393 | mytbs      | /home/postgres/data1_2
(3 rows)

postgres=# \! ls -l $PGDATA/pg_tblspc
total 0
lrwxrwxrwx 1 postgres postgres 22 Jun  5 01:08 16393 → /home/postgres/data1_2
postgres=#
```

在默认数据库中的指定的表空间中创建表。
\c postgres
CREATE TABLE department(id INT PRIMARY KEY, name VARCHAR(32)) TABLESPACE mytbs;
SELECT pg_relation_filepath('department');

```
postgres=# \c
You are now connected to database "postgres" as user "postgres".
postgres=# CREATE TABLE department(id INT PRIMARY KEY, name VARCHAR(32)) TABLESPACE mytbs;
CREATE TABLE
```

```
postgres=# SELECT pg_relation_filepath('department');
          pg_relation_filepath
----------------------------------------
 pg_tblspc/16393/PG_17_202406281/5/16394
(1 row)
```

在oracle数据库中的指定的表空间中创建表。
\c oracle

CREATE TABLE department(id INT PRIMARY KEY, name VARCHAR(32)) TABLESPACE mytbs;
SELECT pg_relation_filepath('department');

```
oracle=# \c
You are now connected to database "oracle" as user "postgres".
oracle=#
oracle=#
oracle=# CREATE TABLE department(id INT PRIMARY KEY, name VARCHAR(32)) TABLESPACE mytbs;
CREATE TABLE
oracle=# SELECT pg_relation_filepath('department');
            pg_relation_filepath
-----------------------------------------
 pg_tblspc/16393/PG_17_202406281/16387/16399
(1 row)
```

在oracle数据库中的默认表空间中创建同名表。说明在一个数据库中，不能够创建同名的表，即使是在不同的空间中。
\c oracle
CREATE TABLE department(id INT PRIMARY KEY, name VARCHAR(32));
CREATE TABLE department2(id INT PRIMARY KEY, name VARCHAR(32));

```
oracle=# \c oracle
You are now connected to database "oracle" as user "postgres".
oracle=# CREATE TABLE department(id INT PRIMARY KEY, name VARCHAR(32));
ERROR:  relation "department" already exists
oracle=# CREATE TABLE department2(id INT PRIMARY KEY, name VARCHAR(32));
CREATE TABLE
oracle=# SELECT pg_relation_filepath('department2');
 pg_relation_filepath
----------------------
 base/16387/16404
(1 row)

oracle=#
```

我们来理解上面生成三个相对文件路径。
pg_tblspc/16393/PG_17_202406281/5/16394
pg_tblspc/16393/PG_17_202406281/16387/16399
base/16387/16404

首先解析一下 pg_tblspc/16393/PG_17_202406281/5/16394。pg_tblspc 是表空间文件夹。16393是之前生成的 mytbs 表空间。PG_17_202406281 是 PG+版本号+初始化时间戳。5是默认数据库postgres。16394是该表的oid。

然后解析一下 pg_tblspc/16393/PG_17_202406281/16387/16399。pg_tblspc 是表空间文件夹。16393是之前生成的 mytbs 表空间。PG_17_202406281 是 PG+版本号+初始化时间戳。16387是新创建的oracle数据库。16399是该表的oid。

最后解析一下 base/16387/16404。base 是默认表空间文件夹。16387是新创建的oracle数据库。16404是该表的oid。

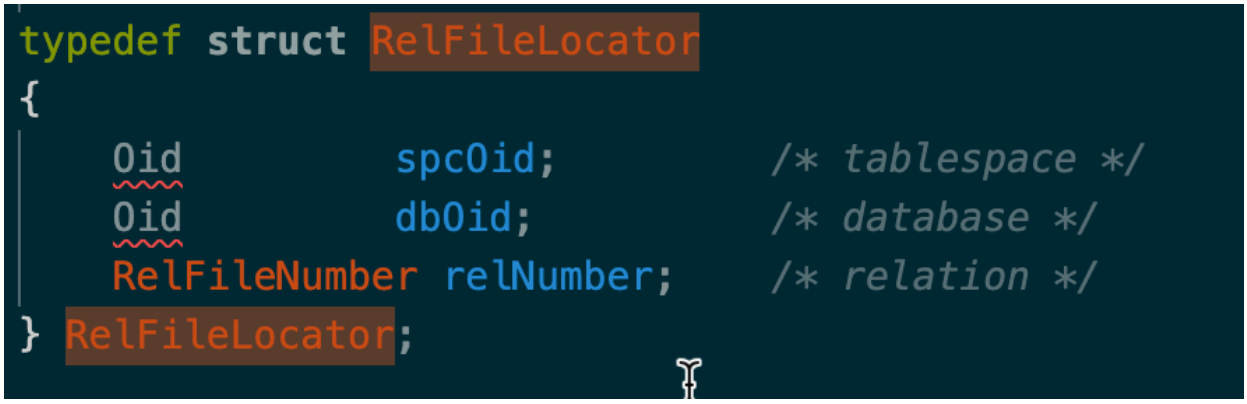
可以使用以下语句查询oid对应的表空间名，数据库名和表名。

```
-- 表空间名
SELECT oid, spcname FROM pg_tablespace WHERE oid = 16393;
-- 数据库名
SELECT oid, datname FROM pg_database WHERE oid = 16387;
-- 表名
SELECT oid, relname FROM pg_class WHERE relfilenode = 16399;
```

```
oracle=# SELECT oid, spcname FROM pg_tablespace WHERE oid = 16393;
  oid  | spcname
-------+---------
 16393 | mytbs
(1 row)

oracle=# SELECT oid, datname FROM pg_database WHERE oid = 16387;
  oid  | datname
-------+---------
 16387 | oracle
(1 row)

oracle=# SELECT oid, relname FROM pg_class WHERE relfilenode = 16399;
  oid  |  relname
-------+------------
 16399 | department
(1 row)

oracle=#
oracle=#
oracle=#
oracle=#
oracle=#
oracle=#
oracle=#
oracle=#
oracle=#
oracle=#
oracle=#
```

查看PG的源代码得知，PG中的物理文件有一个三元组来定义。分别是空间的oid，数据库的oid，和物理相对文件号rel-file-number。这里注意PG中一个表，可以对应一个或多个物理文件。

=



```c
typedef struct RelFileLocator
{
    Oid         spcOid;         /* tablespace */
    Oid         dbOid;          /* database */
    RelFileNumber relNumber;    /* relation */
} RelFileLocator;
```

修改数据库参数，并且使其生效。

```
\! cat $PGDATA/postgresql.auto.conf
show work_mem;
ALTER SYSTEM SET work_mem = 10240;
SELECT pg_reload_conf();
```

```
oracle=# \! cat $PGDATA/postgresql.auto.conf
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
oracle=# show work_mem;
 work_mem
----------
 4MB
(1 row)

oracle=# ALTER SYSTEM SET work_mem = 10240;
ALTER SYSTEM
oracle=# SHOW work_mem;
 work_mem
----------
 4MB
(1 row)

oracle=# \! cat $PGDATA/postgresql.auto.conf
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
work_mem = '10240'
oracle=# SELECT pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)

oracle=# SHOW work_mem;
 work_mem
----------
 10MB
(1 row)

oracle=# \! cat $PGDATA/postgresql.auto.conf
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
work_mem = '10240'
oracle=#
```

查看源代码中的文档信息。数据库启动的时候，postgres.auto.conf 中的设置信息会覆盖 postgres.conf 中的设置信息。

```
<para>
 In addition to <filename>postgresql.conf</filename>,
 a <productname>PostgreSQL</productname> data directory contains a file
 <filename>postgresql.auto.conf</filename><indexterm><primary>postgresql.auto.conf</primary></indexterm>,
 which has the same format as <filename>postgresql.conf</filename> but
 is intended to be edited automatically, not manually.  This file holds
 settings provided through the <link linkend="sql-altersystem"><command>ALTER SYSTEM</command></link> command.
 This file is read whenever <filename>postgresql.conf</filename> is,
 and its settings take effect in the same way.  Settings
 in <filename>postgresql.auto.conf</filename> override those
 in <filename>postgresql.conf</filename>.
 </para>
```

查看控制文件中的信息。
pg_controldata -D /home/postgres/data1

```
postgres@pg-50:~/data1/global$ pwd
/home/postgres/data1/global
postgres@pg-50:~/data1/global$ ls -l pg_control
-rw------- 1 postgres postgres 8192 Jun  5 01:38 pg_control
postgres@pg-50:~/data1/global$ pg_controldata -D /home/postgres/data1
pg_control version number:            1700
Catalog version number:               202406281
Database system identifier:           7512180834185686062
Database cluster state:               in production
pg_control last modified:             Thu 05 Jun 2025 01:38:06 AM UTC
Latest checkpoint location:           0/19BD9F8
Latest checkpoint's REDO location:    0/19BD9A0
Latest checkpoint's REDO WAL file:    000000010000000000000001
Latest checkpoint's TimeLineID:       1
Latest checkpoint's PrevTimeLineID:   1
Latest checkpoint's full_page_writes: on
Latest checkpoint's NextXID:          0:750
Latest checkpoint's NextOID:          24585
Latest checkpoint's NextMultiXactId:  1
Latest checkpoint's NextMultiOffset:  0
Latest checkpoint's oldestXID:        730
Latest checkpoint's oldestXID's DB:   1
Latest checkpoint's oldestActiveXID:  750
Latest checkpoint's oldestMultiXid:   1
Latest checkpoint's oldestMulti's DB: 1
Latest checkpoint's oldestCommitTsXid:0
Latest checkpoint's newestCommitTsXid:0
Time of latest checkpoint:            Thu 05 Jun 2025 01:38:05 AM UTC
Fake LSN counter for unlogged rels:   0/3E8
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:                    replica
wal_log_hints setting:                off
max_connections setting:              100
max_worker_processes setting:         8
```

了解PG主进程的进程ID文件，也叫进程锁文件。了解PGzhu4进程锁文件中内容的含义。

```
postgres@pg-50:~/data1$ pwd
/home/postgres/data1
postgres@pg-50:~/data1$ ls -l postmaster.pid
-rw------- 1 postgres postgres 86 Jun  4 21:38 postmaster.pid
postgres@pg-50:~/data1$ cat postmaster.pid
1315
/home/postgres/data1
1749073082
5432
/tmp
localhost
  1836855         0
ready
postgres@pg-50:~/data1$ ps -ef | grep 1315
postgres    1315       1  0 Jun04 ?        00:00:24 /home/postgres/pg17.5/bin/postgres
postgres    1316    1315  0 Jun04 ?        00:00:00 postgres: checkpointer
postgres    1317    1315  0 Jun04 ?        00:00:06 postgres: background writer
postgres    1319    1315  0 Jun04 ?        00:00:07 postgres: walwriter
postgres    1320    1315  0 Jun04 ?        00:00:09 postgres: autovacuum launcher
postgres    1321    1315  0 Jun04 ?        00:00:00 postgres: logical replication launcher
postgres    6399    1315  0 10:57 ?        00:00:00 postgres: postgres oracle [local] idle
postgres    6573    1274 50 11:21 pts/1    00:00:00 grep --color=auto 1315
postgres@pg-50:~/data1$ echo $PGDATA
/home/postgres/data1
postgres@pg-50:~/data1$ ss -tulnp | grep 5432
tcp   LISTEN 0      200        127.0.0.1:5432      0.0.0.0:*    users:(("postgres",pid=1315,fd=6))
postgres@pg-50:~/data1$ ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x001c0737 0          postgres   600        56         7

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

postgres@pg-50:~/data1$ printf "%d\n" 0x001c0737
1836855
postgres@pg-50:~/data1$
```

为了帮助理解PG的源代码。这里重点理解一下C语言中main函数的。argc和argv。argc 记录参数的个数。argv记录每个参数的内容。argv[0] 表示运行程序的文件名。

vi arg.c

#include <stdio.h>

int main(int argc, char* argv[])

{ for(int i=0; i<argc; i++) printf("argv[%d] = [%s]\n", i, argv[i]); return 0; }

```
postgres@pg-50:~$ cat arg.c
#include <stdio.h>
int main(int argc, char* argv[])
{ for(int i=0; i<argc; i++) printf("argv[%d] = [%s]\n", i, argv[i]); return 0; }

postgres@pg-50:~$ gcc -Wall arg.c -o arg
postgres@pg-50:~$ ./arg -D $PGDATA
argv[0] = [./arg]
argv[1] = [-D]
argv[2] = [/home/postgres/data1]
postgres@pg-50:~$
```

对于C语言输入参数的理解。这里不使用 pg_ctl启动数据库，而是手动启动数据库，并且查看启动参数。postmaster.opts 是PG的启动参数文件记录了启动PG的参数。
nohup $PGHOME/bin/postgres -D /home/postgres/data1 > logfile1 2>&1 &

```
postgres@pg-50:~$ cd $PGDATA
postgres@pg-50:~/data1$ ls -l postmaster.opts
-rw------- 1 postgres postgres 35 Jun  4 21:38 postmaster.opts
postgres@pg-50:~/data1$ cat postmaster.opts
/home/postgres/pg17.5/bin/postgres
postgres@pg-50:~/data1$ pg_ctl stop
waiting for server to shut down.... done
server stopped
postgres@pg-50:~/data1$ nohup $PGHOME/bin/postgres -D /home/postgres/data1 > logfile1 2>&1 &
[1] 6680
postgres@pg-50:~/data1$ cat postmaster.opts
/home/postgres/pg17.5/bin/postgres "-D" "/home/postgres/data1"
postgres@pg-50:~/data1$
```

每次启动数据库的时候，PG会根据启动数据库的参数来重写 postmaster.opts。

```c
/*
 * Create the opts file
 */
static bool
CreateOptsFile(int argc, char *argv[], char *fullprogname)
{
    FILE       *fp;
    int         i;

#define OPTS_FILE   "postmaster.opts"

    if ((fp = fopen(OPTS_FILE, "w")) == NULL)
    {
        ereport(LOG,
                (errcode_for_file_access(),
                 errmsg("could not create file \"%s\": %m", OPTS_FILE)));
        return false;
    }

    fprintf(fp, "%s", fullprogname);
    for (i = 1; i < argc; i++)
        fprintf(fp, " \"%s\"", argv[i]);
    fputs("\n", fp);

    if (fclose(fp))
    {
        ereport(LOG,
                (errcode_for_file_access(),
                 errmsg("could not write file \"%s\": %m", OPTS_FILE)));
        return false;
    }

    return true;
}
```

End