[结构图] PostgreSQL 架构介绍

- 一、逻辑结构
- 二、物理结构
  - 2.1 数据目录介绍
  - 2.2 数据库文件的布局
  - 2.3 索引与文件布局
  - 2.4 新增表空间后的布局
  - 2.5 物理文件整体布局
- 三、进程与内存结构
  - 3.1 启动流程介绍
  - 3.2 进程
    - 3.2.1 服务器进程
    - 3.2.2 backend process
    - 3.2.3 background process
  - 3.3 内存
    - 3.3.1 本地内存
    - 3.3.2 共享内存
- 四、数据页结构
- 五、buffer pool 结构
- 七、参考链接

一、逻辑结构

在 PostgreSQL 里面，实例是多个database 的集合，每个database 里面有多个 schema，每个 schema 中存放表、索引、视图等数据对象。

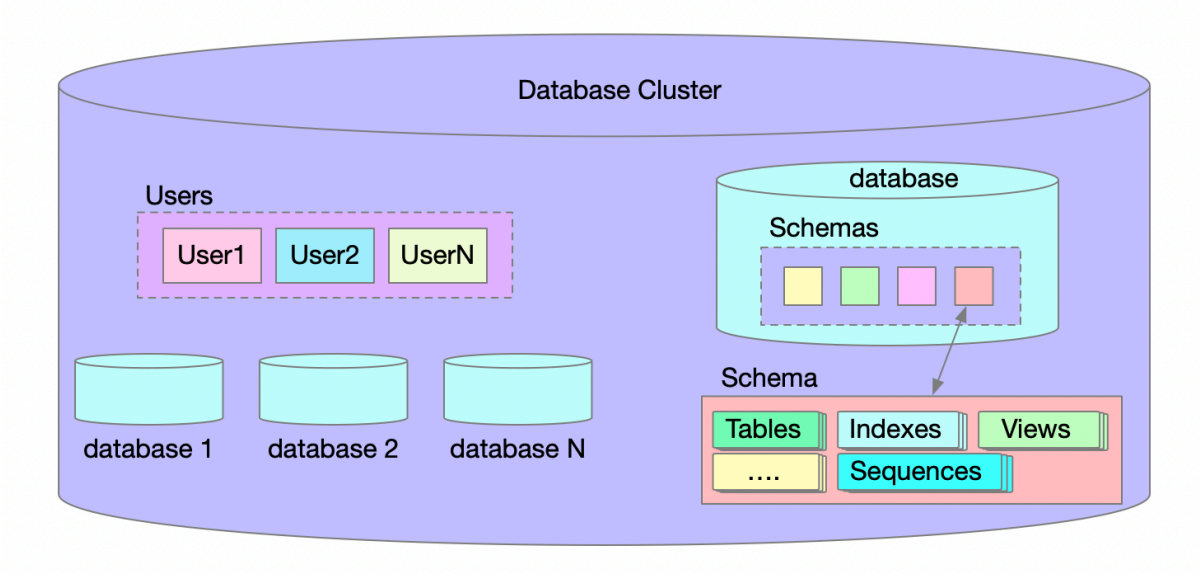
创建表、视图、过程、函数等数据对象时，默认是存放在一个名叫 public 的 schema 中。

```
postgres@[local]:1921=# \dn
 public | postgres
```

一个用户的一个 **链接** 只能操作一个 database，psql -U postgres -h 127.0.0.1 -p 1921 -d mydb 表示一个 **链接**，链接建立成功后后续的操作称作 **会话**。

一个用户的一个 **链接** 可以操作多个 schema 中的对象。

逻辑结构如下图



在 PostgreSQL 中，所有的数据库对象都有各自的对象标识符(OID)进行内部管理，这些对象标识符是无符号的4字节整数。由于对象类型的不同步，他们的存放位置也不一样。数据库与堆表的 OID 分别存放在 pg_database 和 pg_class 中，使用下面语句可以查询到:

```
postgres@[local]:1921=# select datname, oid from pg_database;
  datname  |  oid
-----------+-------
 postgres  | 13158
 template1 |     1
 template0 | 13157
 mydb      | 16408
(4 rows)

postgres@[local]:1921=# select relname, oid from pg_class where relname='cities';
 relname |  oid
---------+-------
 cities  | 16393
(1 row)
```

二、物理结构

2.1 数据目录介绍

数据目录默认存放在 $PGDATA 中，目录中所有目录及文件介绍如下表

名称	说明
base	包含每个数据库对应的子目录的子目录。每个 database 都属于一个 oid， base 里面的子目录为数据库的 oid，可以通过 pg_database 中的 oid 或者 datlastsysoid 获取。第二层子目录为表、索引对象的目录，同样是 oid 表示，通过 pg_class 中的 oid 查看。
global	包含 cluster 范围的表的子目录，比如 pg_database
pg_commit_ts	包含事务提交的时间戳数据的子目录
pg_xact	包含事务提交状态数据的子目录
pg_dynshmem	包含被动态共享内存子系统所使用的文件的子目录
pg_logical	包含用于逻辑复制状态数据的子目录
pg_multixact	包含多事务状态数据的目录(用于共享行锁)
pg_notify	包含 Listen/Notify 状态的目录
pg_replslot	包含复制槽数据的目录
pg_serial	包含已提交的可序列化事务信息的目录
pg_snapshots	包含导出的快照目录
pg_stat	包含用于统计子系统的永久文件的目录
pg_stat_tmp	包含用于统计子系统的临时文件的目录
pg_subtrans	包含子事务状态数据的目录
pg_tblspc	包含指向表空间的符号链接的目录
pg_twophase	用于 prepare 事务状态文件的目录
pg_wal	保存预写日志
PG_VERSION	记录 PostgreSQL 主版本号的文件
pg_hba.conf	客户端认证文件

pg_ident.conf	系统用户与数据库用户映射的文件
postgresql.auto.conf	参数文件，只保存 alter system 命令修改的参数
postgresql.conf	参数文件
postmaster.opts	记录服务器最后一次启动时使用的命令行参数
postmaster.pid	数据库pid 文件
serverlog	操作日志文件

2.2 数据库文件的布局

数据库是 base 目录下的子目录，数据库的名称是使用 OID 来标识的，下面看示例:

```
postgres@[local]:1921=# select datname, oid from pg_database;
  datname  |  oid
-----------+-------
 postgres  | 13158
 template1 |     1
 template0 | 13157
 mydb      | 16408
(4 rows)

postgres@[local]:1921=# select relname, oid from pg_class where relname='cities';
 relname |  oid
---------+-------
 cities  | 16393
(1 row)
```

操作系统上查看数据

```
[root@localhost base]# ls
1  13157  13158  16408
[root@localhost base]# pwd
/opt/pgdata/pg10_8/data/base
[root@localhost base]# ll 13158/16393
-rw------- 1 postgres postgres 8192 6月   5 05:42 13158/16393
```

2.3 索引与文件布局

如果表的数据与索引文件大小小于 1GB 时，对应表与索引的 OID 是一样的，但并不总是一样。如果执行 TRUNCATE，REINDEX，CLUSTER 等命令时，会重新分析 relfilenode 值。示例如下：

```
postgres@[local]:1921=# SELECT relname,oid,relfilenode FROM pg_class WHERE
relname = 'cities';
 cities  | 16393 |      16393

postgres@[local]:1921=# truncate table cities;
TRUNCATE TABLE
postgres@[local]:1921=# SELECT relname,oid,relfilenode FROM pg_class WHERE
relname = 'cities';
 cities  | 16393 |      16418
```

如果表的数据与索引文件大于 1GB 时，索引的 relfilenode 会发生改变。大于 1GB 的部分放到 relfilenode.1 中，如果新文件达到 1GB，那么再新建一个 relfilenode.2，依次类推。

```
[root@localhost 13158]# du -sh 16*
8.0K    16387
964M    16418
268K    16418_fsm
```

```
postgres@[local]:1921=# select count(*) from cities;
  count
----------
 18582912
(1 row)
```

```
[root@localhost 13158]# du -sh 16*
8.0K    16387
1.1G    16418
170M    16418.1
324K    16418_fsm
```

2.4 新增表空间后的布局

当使用 create tablespace 语法创建一个表空间时，会在该目录下创建一个当前数据库版本的子目录(如:PG_10_201707211)。在这个版本子目录下创建对应数据库的 OID 子目录，之后在数据库子目录下创建对应表的 OID ，示例如下:

- 提前创建对应表空间目录

```
[postgres@localhost ~]$ mkdir /opt/pgdata/pg10_8/data/test_tbs
```

- 创建表空间

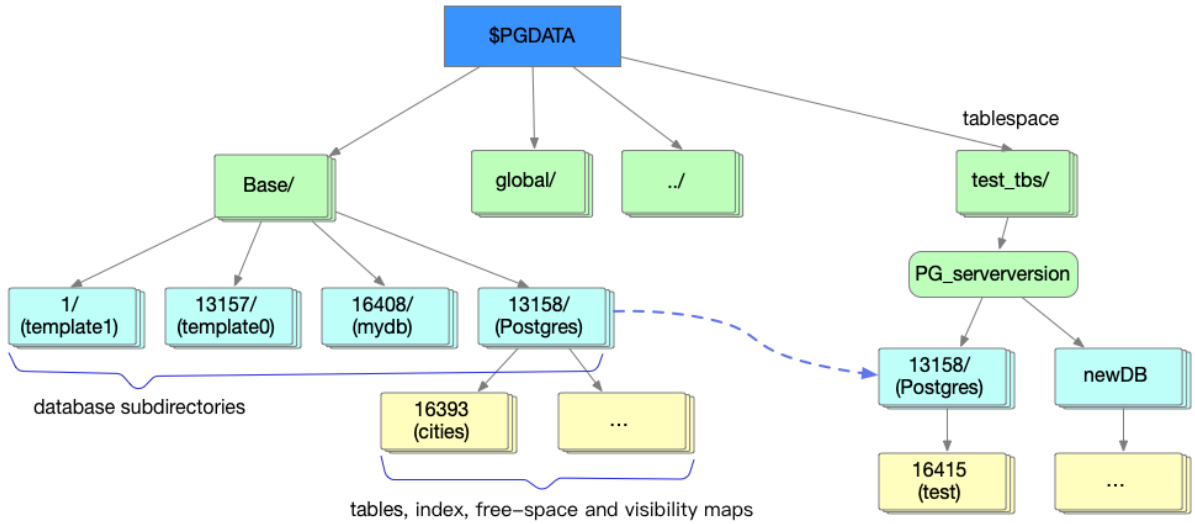
```
postgres@[local]:1921=# create tablespace test_tbs location
'/opt/pgdata/pg10_8/data/test_tbs';

# 在此表空间上创建表
postgres@[local]:1921=# create table test (id int) tablespace test_tbs;

# 查看表的 OID
postgres@[local]:1921=# select relname, oid from pg_class where relname='test';
 test    | 16415
```

- 操作系统查看

```
[root@localhost data]# ll test_tbs/PG_10_201707211/13158/
total 0
-rw------- 1 postgres postgres 0 Jun  6 07:31 16415
```

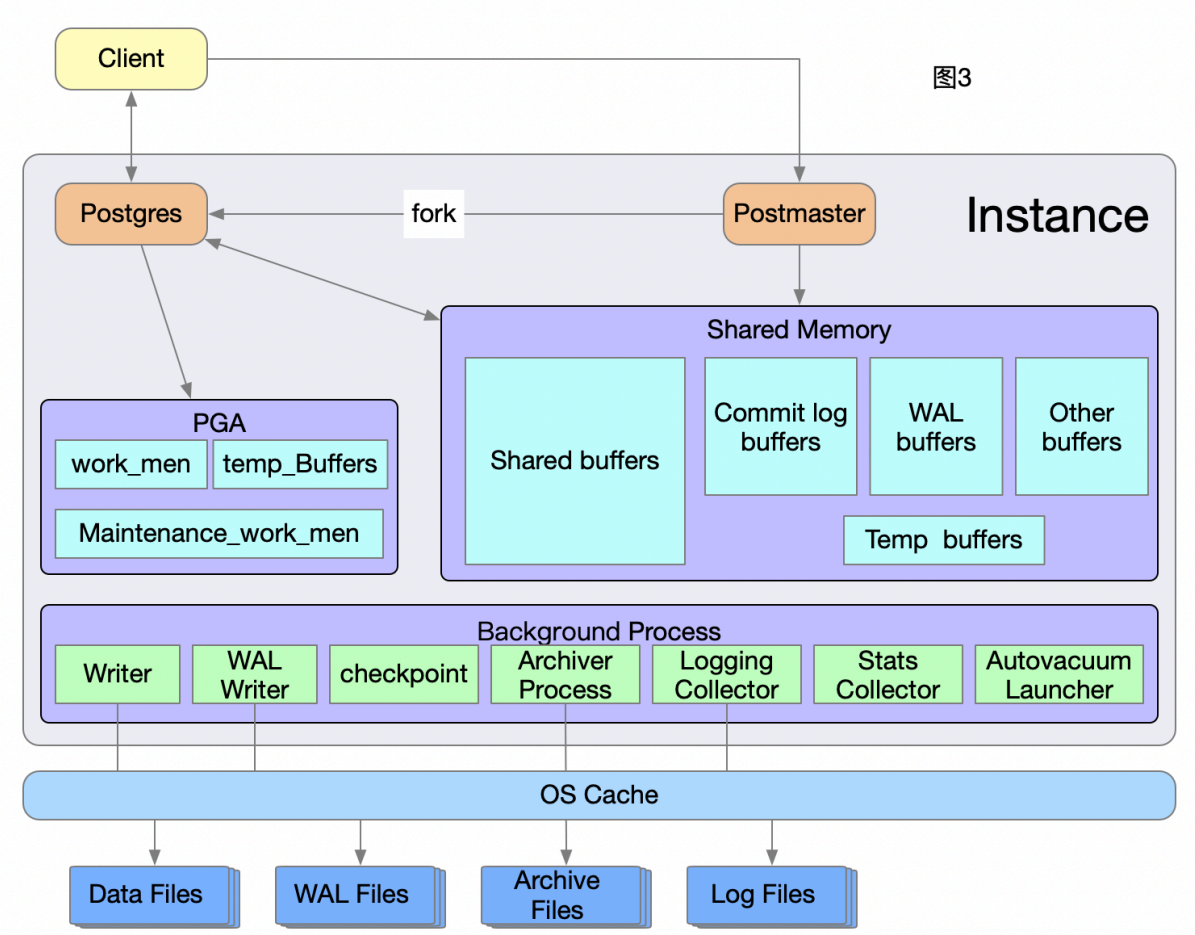
2.5 物理文件整体布局

根据上面内容介绍，整体的物理文件布局如下图



三、进程与内存结构

图3 展示了整体的内存与进程的结构信息。



3.1 启动流程介绍

PostgreSQL 是一个单机多进程的关系型数据库，在 PostgreSQL 中 postmaster 命令是 Postgres 的软链接(主进程)。
postmaster 命令启动时，会读取配置文件(postgres.conf)，根据里面配置创建 shared buffer pool 也就是图中的 Shared Memory 区域，接着会创建后台线程，最后这些线程去操作对应的数据库文件。

3.2 进程

进程主要分为三类：
第一类为服务器进程(postmaster)：主要负责与数据库集群管理相关的所有进程的父进程。
第二类为 backend process ： 主要负责所有客户端的请求。由服务器进程 fork 出来的。
第三类为 background process ： 主要负责处理数据库中的每个子模块管理功能。

3.2.1 服务器进程

postgres 服务器进程(对应图中的 Postmaster)是 PostgreSQL 服务器中所有进程的父进程。在早期版本中，它被称为 postmaster 。当前大版本号为 10 , 为了向下兼容， postmaster 为 postgres 的软链接。

通过使用 start 选项执行 pg_ctl 实用程序，启动 postgres 服务器进程。然后，它在内存中分配共享内存区域，启动各种后台进程，在必要时启动复制关联进程和后台工作进程，并等待来自客户端的连接请求。每当从客户端接收连接请求时，它就会启动后端进程。（然后，启动的后端进程处理由连接的客户端发出的所有查询。）

postgres 服务器进程侦听一个网络端口，默认端口为 5432 。虽然可以在同一主机上运行多个 PostgreSQL 服务器，但每个服务器应设置为侦听彼此不同的端口号，例如，5432,5433 等。

3.2.2 backend process

后端进程（也称为 postgres ， 对应图中的 postgres ）由 Postmaster 服务器进程启动，并处理由一个连接的客户端发出的所有查询。它通过单个 TCP 连接与客户端通信，并在客户端断开连接时终止。

由于只允许操作一个数据库，因此在连接到 PostgreSQL 服务器时必须显式指定要使用的数据库。

PostgreSQL 允许多个客户端同时连接; 配置参数 max_connections 控制最大客户端数（默

认为100）。

如果许多客户端（如WEB应用程序）经常重复与 PostgreSQL 服务器的连接和断开连接，则会增加建立连接和创建后端进程的成本，因为 PostgreSQL 尚未实现本机连接池功能。这种情况对数据库服务器的性能有负面影响。为了处理这种情况，通常使用池化中间件（pgbouncer 或 pgpool-II ）。

```
postgre  94909  17883  0  2019 ?       00:00:00 postgres: opensips opensips
172.16.2.83(2368) idle
postgre  94911  17883  0  2019 ?       00:00:00 postgres: opensips opensips
172.16.2.83(2370) idle
postgre  94920  17883  0  2019 ?       00:01:24 postgres: opensips opensips
172.16.2.83(2372) idle
postgre  94923  17883  0  2019 ?       00:01:51 postgres: opensips opensips
172.16.2.83(2374) idle
postgre  94924  17883  0  2019 ?       00:00:00 postgres: opensips opensips
172.16.2.83(2376) idle
postgre  94925  17883  0  2019 ?       00:00:51 postgres: opensips opensips
172.16.2.83(2378) idle
postgre  94926  17883  0  2019 ?       00:02:49 postgres: opensips opensips
172.16.2.83(2379) idle
postgre  94927  17883  0  2019 ?       00:15:40 postgres: opensips opensips
172.16.2.83(2382) idle
postgre  94928  17883  0  2019 ?       00:00:54 postgres: opensips opensips
172.16.2.83(2384) idle
postgre  94929  17883  0  2019 ?       00:00:56 postgres: opensips opensips
172.16.2.83(2386) idle
postgre  94930  17883  0  2019 ?       00:01:02 postgres: opensips opensips
172.16.2.83(2388) idle
postgre  94931  17883  0  2019 ?       00:01:13 postgres: opensips opensips
172.16.2.83(2390) idle
postgre 114695  17883  0 09:32 ?        00:00:00 postgres: opensips opensips
172.16.6.58(64462) idle
```

3.2.3 background process

根据 图3 中 background process 中的顺序介绍每个进程的功能

进程名	功能说明
background writer	负责将 shared buffer 中的脏页按一定比率定期写入到磁盘文件中。
WAL writer	定期将 WAL buffer 上的 WAL 数据写入并刷新到磁盘中。
checkpoint	执行检查点过程。
archiver	将 WAL 日志进行归档操作。
logging collector	将错误消息写入日志文件。
stats collector	收集诸如 pg_stat_activity 和 pg_stat_database 等统计信息。
autovacuum launcher	定期调用 autovacuum-worker 进程进行 vacuum 处理。（更确切地说，它要求为 postgres 服务器创建 autovacuum wroker 进程。）

以下示例展示了 PostgreSQL 服务器的实际进程。

```
[root@localhost opt]# ps -ef | grep post
root      1486  1474  0 02:07 pts/2    00:00:00 grep post
postgres  8167     1  0 Jun04 ?        00:00:05 /opt/pgsql/bin/postmaster -D
/opt/pgdata/pg10_8/data
postgres  8175  8167  0 Jun04 ?        00:00:01 postgres: checkpointer process
postgres  8176  8167  0 Jun04 ?        00:00:06 postgres: writer process
postgres  8177  8167  0 Jun04 ?        00:00:07 postgres: wal writer process
postgres  8178  8167  0 Jun04 ?        00:00:07 postgres: autovacuum launcher process
postgres  8179  8167  0 Jun04 ?        00:00:13 postgres: stats collector process
postgres  8180  8167  0 Jun04 ?        00:00:00 postgres: bgworker: logical replication
launcher
```

3.3 内存

在 PostgreSQL 中内存可以分为两大类：

- 本地内存区域(图3 中 PGA)：由每个后端进程分配给自己使用的内存。每个链接都会进行一次分配。

- 共享内存区域(图3 Shared Memory)：由 PostgreSQL 服务器的所有进程使用。

3.3.1 本地内存

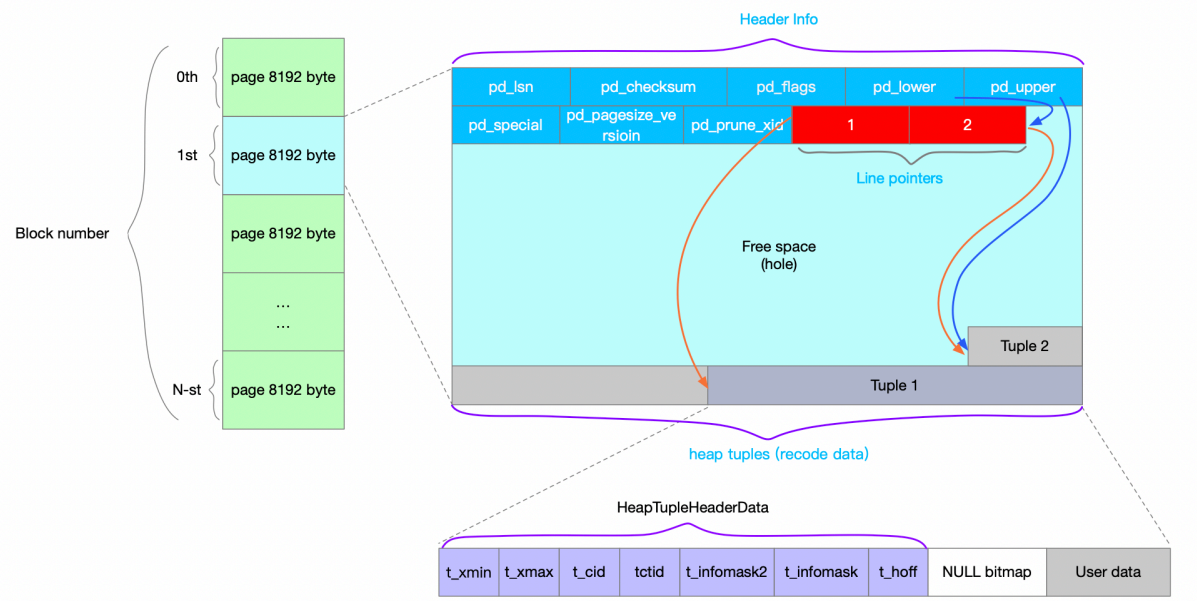
每个后端进程为查询处理分配一个本地内存区域; 每个区域分为几个子区域 - 其大小是固定的或可变的

区域名称	说明
work_mem	Executor 使用此区域通过 ORDER BY 和 DISTINCT 操作对元组进行排序，以及通过 merge-join 和 hash-join 操作连接表。默认 4MB 。
maintenance_work_mem	某些维护操作（例如，VACUUM，REINDEX ）使用此区域。
temp_buffers	执行程序使用此区域存储临时表。

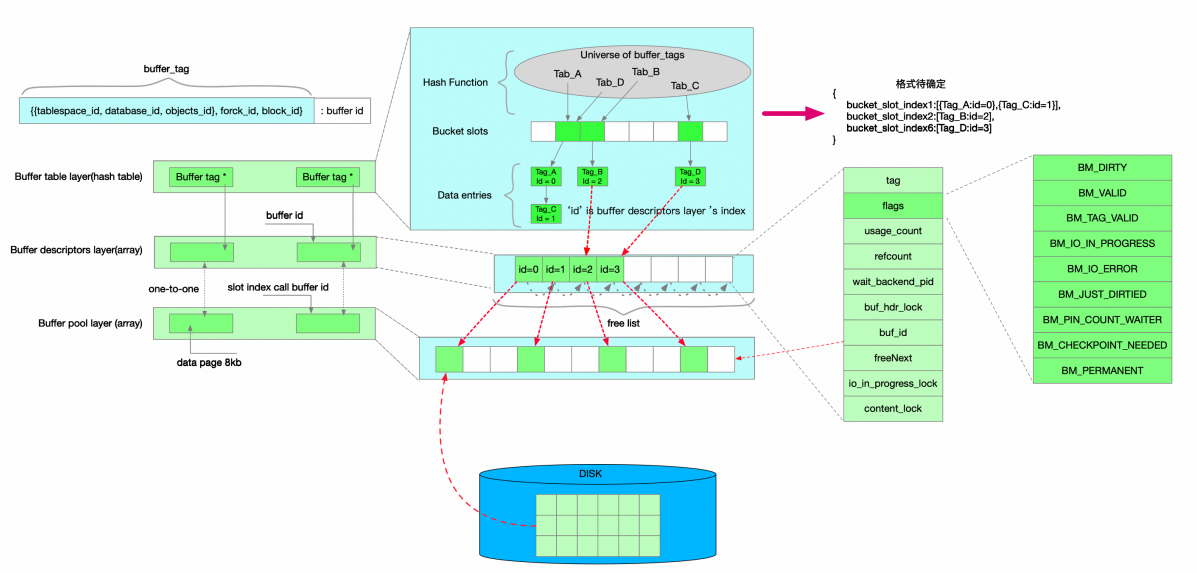
3.3.2 共享内存

区域名称	说明
shared bufer pool	PostgreSQL 将表和索引中的页面从持久存储加载到此处，并直接操作它们。
WAL buffer	为了确保服务器故障没有丢失任何数据，PostgreSQL 支持 WAL 机制。WAL 数据（也称为 XLOG 记录）是 PostgreSQL 中的事务日志; 在写入持久存储之前，WAL 缓冲区是 WAL 数据的缓冲区。
Commit log buffer	提交日志（CLOG ）保持并发控制（CC ）机制的所有事务（例如，in_progress ，已提交，已中止）的状态。

四、数据页结构



五、buffer pool 结构



七、参考链接

[The Internals of PostgreSQL]

[PostgreSQL Architecture]