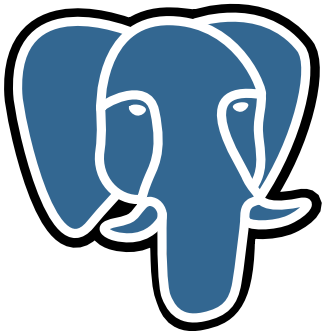
PostgreSQL 老鸟之路 - 管理篇

深入剖析 PostgreSQL 内核源码的原创之作



作者：小布

目录

自序

在 IT 行业浪迹了二十多年以后，我决定写人生第一本 IT 技术方面的书，那就是你面前的这本《PostgreSQL 老鸟之路 - 管理篇》。也许是因为命运的安排，十多年前我来到美国后，阴差阳错地进入了数据库管理员 DBA（Database Administrator）的行业，从此我的职业生涯的大部分时间都是和 Oracle/PostgreSQL 等数据库软件打交道，所以本书的内容是深入学习著名的开源数据库 PostgreSQL 运维管理方面的相关技术，大约类似“某某某从入门到精通”之流，但绝对不会教你如何从删库到跑路。

写书有三种类型，编书，著书和编著。此三者的区别显而易见。所谓编书，就是把能够找到的相关资料汇编成册，内容是别人的，自己做的工作是汇集整理。所谓著书，就是把自己消化理解后的知识和经验写出来，把茶壶里的饺子倒出来。编著则是居于“编书”和“著书”中间的层次。很显然，“编书”层次最低，“著书”层次最高。著书要花费精力和心血最多，这是个良心活。曹氏曰：“满纸荒唐言，一把辛酸泪。都云作者痴，谁解其中味”，此语真乃著书的天花板。我亦想在 PostgreSQL 领域整齐百家诸语，成一家之言，竭尽自己有限的才智来“著”一本书，然后藏之名山，以俟后世君子。至于能否达到此种我希冀的境界，只能说，尽吾志而不能至者，无悔矣。

从二十世纪中后期到二十一世纪上半期，是 IT 技术革命大爆发的时代，此乃人类之幸事。但科技的快速发展也带来了某些副作用。其中一个副作用就是劳动者需要更长的时间和精力来学习，才有能力从事专业领域的工作。君不见，学士，硕士，博士不断内卷，直逼壮士，猛士，乃至烈士。等到博士毕业后，蓦然发现自己已经宿舍明镜悲白发，可怜依然单身狗。这导致当代年轻人晚婚，不婚，少子化，躺平等社会问题越来越普遍。在 IT 技术领域，随着技术复杂度的提高，进入其中的壁垒越来越高。后来的学习者想登堂入室，难度越来越大。譬如 Linux 内核的开发，对于初学者来说，学习曲线异常陡峭，除非投入巨大的精力和时间成本，否则连老鸟的谈话都很难听懂。如果没有足够的利益吸引，想进入技术领域的年轻人只会越来越少。老鸟终究会慢慢凋零，而新鲜血液补充不足，是各种 IT 技术发展过程中普遍面临的问题。我写本书的目的就是希望能够在学习者进入 PostgreSQL 数据库领域的过程中，为他们降低学习门槛做一些力所能及的微薄贡献。另一方面也是对自己职业生涯的一个负责任的总结。

或云：坊间相关 PostgreSQL 的书籍早已汗牛充栋，你写的这本书又有什么独特的价值？答曰：因为我对目前所有的关于 PostgreSQL 的书籍都不甚满意。不满是技术进步的动力。在我学习 PostgreSQL 的过程中，充满了痛苦。很多初级读物流于肤浅，而深入的书籍因为前导概念和知识的缺乏，又让初学者不知所云。我不得不像一只勤劳的小蜜蜂，不断在互联网的世界里飞来飞去，四处艰难地采集我寻找的干货。自己要花费大量时间，不断做实验，分析源码，做学习笔记，才能够有一点点收获。我的学习笔记上寥寥数语心得的背后，都是一把辛酸泪。我不想后学者再反复重复这个痛苦过程，就想写一本书，循序渐进地把基本知识和深入知识融合，让读者只要按顺序学习本书，就能够以较少的痛苦代价，达到一定程度的“登堂入室”的境界。不让别的学习者像自己一样痛苦，是我写本书的强烈冲动。

本书写作遵循的原则：

- **说自己的话**：既然是著书，就是倾倒自己肚子里面货真价实的饺子。不说抄来的话，力争每一句话都是自己完全理解的，然后再传递给读者。这是对读者负责，也是对自己有益。所以我力争本书中的每一句话都是自己的所思所想，是自己的语言，是能够理解的人话。要对得起天地，对得起读者，对得起自己的良心。
- **循序渐进**：人类学习知识的过程肯定是循序渐进的。如果概念 B 是建立在概念 A 的基础上，那么学习的时候必须先学习搞懂概念 A，然后才能学习概念 B。这当然是废话。然而很多书籍和教程往往忽视这个根本性的原则，导致读者如鲠在喉，想吐槽却无处说理，吾亦深受其苦。所以“循序渐进”是本书遵循的最高原则。我的目标是：只要读者按照顺序反复学习本书，就能够从门外汉到登堂入室。整个过程非常流畅，学习时候的痛苦感亦大大减少。
- **内外兼修**：当我们学习 Oracle, SQL Server 等商业闭源软件的时候，只可能学习“外在”的功能性知识，了解这个软件的功能，却不知道它的内部是如何实现的。这就是老话说的“知其然而不知其所以然”。我从事 Oracle DBA 十几年了，但当你问我对 Oracle 数据库技术掌握到什么程度，我只能默然，脸红，承认自

己懂的太少。这是因为 Oracle 的源代码是商业机密，不可能公布天下的。所以无论我多么努力，学习闭源软件技术的时候，总有一种隔靴挠痒的感觉。我们知道，如果感觉自己彻底掌握了一个技术，就会带来巨大的成就感和自信心，这是人性。这种成就感和自信心在开源软件身上才能够获得。所以本书在讲解外在的 PostgreSQL 各种功能知识的同时，会引导读者阅读背后的关键源码，让读者能够内外兼修，通过源代码和软件功能的互相验证，达到“知其然且知其所以然”的境界，增加彻底掌握的满足感和幸福感，从而进一步增加学习的兴趣。

- **化繁为简：**大道至简，计算机的终极道理是 0 和 1 的关系。无论一个计算机技术多么复杂，包括现在当红炸子鸡的 ChatGPT 等 AI 技术，对于一个真正掌握此技术的人来说，都可以浅显易懂的方式传递给后学者。面对庞大繁杂的源代码，用简明扼要的图来描述其关键思想，再引用关键代码配合展示重要的细节和要点，是帮助读者迅速理解软件技术背后秘密的重要途径。本书所有的图都是我花费大量心血手工绘制的，而且我确保每张图都尽可能简单，易于理解和记忆，这是本书一大特色。
- **中英文夹杂：**IT 技术是西方文明的舶来品，大量高质量的知识依然存在于英语世界。很多概念，可以翻译成中文，但是不如直接使用英文更好。举一个例子：PostgreSQL 的内存管理，有几个概念，MemoryContext, Block, Chunk，可以分别翻译成“内存上下文”，“内存块”，“内存切片”。但是我在阅读别人的 IT 技术文章的过程中觉得专业概念直接引用英文单词，比用中文翻译的概念更加清晰，通用概念如进程 (Process)/线程 (Thread) 等除外。IT 从业者几乎百分百都具备一定的英文阅读能力，所以本书在表述上用中文，而涉及的专业概念直接用英文单词，中英文夹杂，不会影响读者的阅读理解，反而更加清晰准确地阐述所要表达的内容。我在本书中秉承着专业概念直接使用英文单词之原则。

未来 PostgreSQL 的发展势头肯定会越来越好，市面上正在出现越来越多的相关高薪工作岗位。当你沮丧地发现自己不是走管理路线当老板的料的时候，如果能够通过自己的勤奋学习，“彻底”掌握某种高技术，收获了满足感，薪水又超过富士康的全蛋兄，更无需站生产流水线，甚至可以在家远程上班，这难道不是很好的人生选择吗？我相信在每个普通人的内心深处肯定有一种共识：穷尽毕生钻研一门技术到极精深的境界，做一枚优秀的匠人，无需操心过多烦人的人际关系，亦是一种幸福的人生。青青子衿，悠悠我心，此何人哉？

是为序！

第一章 PostgreSQL 实验环境的安装

本章是开天辟地的第一章，当然要介绍如何搭建自己的 PostgreSQL(在本书中有时候简称 PG) 的学习实验环境。因为本书讲解的内容比较深入，所以只给大家介绍如何从源码开始安装 PostgreSQL。既学习 PostgreSQL 的软件功能，又学习背后的源码实现，是本书最大的特色之一。

我假定本书的读者具备一定的 Linux 使用经验和初步 C 语言的编程能力，能够看懂基本的 C 语言源代码。本书的全部实验都是在 Linux 环境下进行的，不讨论 Windows 环境。如果你不具备 Linux 基本操作和 C 编程的能力，可以快速地在互联网上寻找相关资料进行快速学习。这方面的文档和视频等学习资料在互联网上非常泛滥。你无需花费太多时间学习 Linux 和 C 的知识。恶补一个星期的时间，所掌握的 Linux 和 C 的能力就足够学习本书的内容了。

如何安装 Linux 环境，我就不重复介绍了，互联网上这方面的资料也非常多。我建议你使用 VMWare/Virtual Box/Docker 等虚拟机软件来运行你的 Linux。Linux 发行版本的选择也无所谓。我是曾经是 CentOS Linux 的重度用户，但是本书所有的实验就是在 Debian Linux 11 上完成的，所使用的用户统一为 postgres。

本书选择的 PostgreSQL 版本是 15 和 16。可能你学习本书的时候，PostgreSQL 已经发布了更高的版本。但是本书绝大部分的内容是通用的，适用几乎 PostgreSQL 9 以后的所有版本。

1.1 PostgreSQL 的源码安装过程

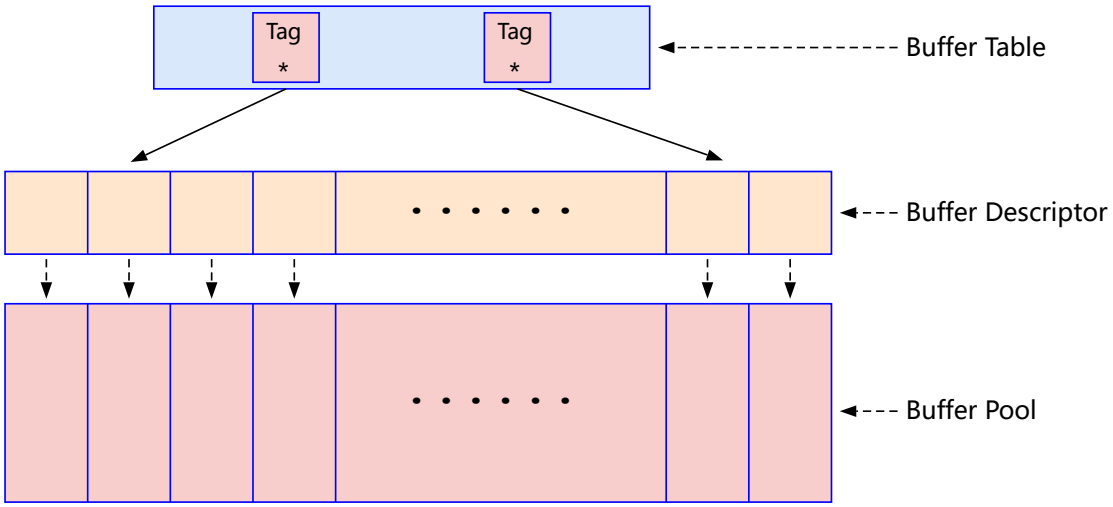
1.1.1 下载源码包

你访问 PostgreSQL 的官方网站 (www.postgresql.org)，很容易找到 Download 的入口，选择下载源码包到本地。例如我下载了 postgresql-15.2.tar.gz 或者 postgresql-15.2.tar.bz2 两个文件。它们都是 PostgreSQL 15.2 的源码包，只不过是压缩的格式不同，一个是 gzip 格式，一个是 bz2 格式。你把源码包文件上传到你的 Linux 实验服务器上，在 postgres 用户可以写的某一个目录下，例如/home/postgres。然后用 tar zxvf postgresql-15.2.tar.gz 解压缩 tar.gz 格式的源码包，用 tar jxvf postgresql-15.2.tar.bz2 解压缩 tar.bz2 格式的源码包。你运行 ls -l 命令，就可以看到解压缩后产生的 postgresql-15.2 的目录，如下所示：

```
$ id
uid=1000(postgres) gid=1000(postgres) groups=1000(postgres),.....
$ pwd
/home/postgres/postgresql-15.2
$ ls -l
total 1276
-rw-r--r-- 1 postgres postgres 397 Feb 6 14:39 aclocal.m4
drwxr-xr-x 2 postgres postgres 4096 Feb 6 14:50 config
-rw-r--r-- 1 postgres postgres 454572 Feb 23 14:20 config.log
-rwxr-xr-x 1 postgres postgres 40618 Feb 23 14:20 config.status
-rwxr-xr-x 1 postgres postgres 601519 Feb 6 14:39 configure
-rw-r--r-- 1 postgres postgres 89258 Feb 6 14:39 configure.ac
drwxr-xr-x 61 postgres postgres 4096 Feb 6 14:50 contrib
-rw-r--r-- 1 postgres postgres 1192 Feb 6 14:39 COPYRIGHT
drwxr-xr-x 3 postgres postgres 4096 Feb 6 14:50 doc
-rw-r--r-- 1 postgres postgres 4264 Feb 23 14:20 GNUmakefile
-rw-r--r-- 1 postgres postgres 4264 Feb 6 14:39 GNUmakefile.in
-rw-r--r-- 1 postgres postgres 277 Feb 6 14:39 HISTORY
-rw-r--r-- 1 postgres postgres 63842 Feb 6 14:51 INSTALL
```

```
-rw-r--r-- 1 postgres postgres 1875 Feb 6 14:39 Makefile
-rw-r--r-- 1 postgres postgres 1213 Feb 6 14:39 README
drwxr-xr-x 16 postgres postgres 4096 Feb 23 14:20 src    <---核心源代码均在此目录下
```

本示例 pdf 的主要考察重点是 LaTex 是否可以嵌入和显示 SVG 的矢量图像。现在看来没有任何问题。方法是先把 SVG 转换成 PDF 格式，再嵌入就可以了。



本示例 pdf 的主要考察重点是 LaTex 是否可以嵌入和显示 SVG 的矢量图像。现在看来没有任何问题。方法是先把 SVG 转换成 PDF 格式，再嵌入就可以了。

1.1.2 数据库的创建和启停

内容有待补充

第二章 数据文件的分析

数据库软件最主要的工作是保存数据，查询数据。数据是保存在数据文件 (Data File) 里面的。数据库可以按行存储数据，也可以按列存储数据，由此产生了行式数据库（Row Oriented Databases）和列式数据库（Columnar Database）两种类型。传统的关系型数据库，包括 Oracle,Microsoft SQL Server, MySQL 和 PostgreSQL 都是行式数据库。在本章，我们来对 PostgreSQL 的数据文件的结构进行研究。

从本章开始，我们要一边学习 PostgreSQL 的外在功能，同时要阅读和理解 PostgreSQL 的源代码，所以在本节中我们先对 PostgreSQL 源代码的基本知识做一些介绍。

2.1 关于 PostgreSQL 源代码的基础知识

我们在第一章中学习了如何通过下载和编译 PostgreSQL 的源码包完成 PostgreSQL 数据库软件的安装。在源码里面，有一个 src 子目录和一个 contrib 子目录。src 子目录中包含全部的 PostgreSQL 的核心源代码。contrib 子目录中包含了外围工具的源代码。当我引用某些数据结构和源代码的时候，我会用类似/* in src/include/storage/bufpage.h */ 的注释来标注所引用的数据结构和代码是在文件 bufpage.h 中定义的，这是一个相对路径。假设我们的源代码目录是/home/postgres/postgresql-15.2，则 bufpage.h 文件的绝对路径就是：/home/postgres/postgresql-15.2/src/include/storage/bufpage.h。这样方便大家直接打开该文件进行查找和阅读。

2.1.1 配置头文件

在源码包中,有一些头文件里面定义了大量的宏,用来控制源码的编译行为,如 src/include/pg_config_manual.h。我们可以称之为配置头文件。有三个配置 pg_config.h，pg_config_os.h 和 pg_config_ext.h 是不在源码包中的。你必须运行 configure 命令后，configure 根据对操作系统环境的检测，自动生成这三个文件，放在 src/include 目录下。所以如果你发现无法找到某些宏的定义的时候，可以运行 configure 命令产生这三个配置头文件。可能你要找的宏就在这三个头文件中。

2.1.2 基本数据类型

在 PostgreSQL 源代码中使用了大量的基础数据，我们需要提前熟悉这些基础数据类型。

```
/* in src/include/c.h */
typedef signed char int8;         /* == 8 bits */
typedef signed short int16;       /* == 16 bits */
typedef signed int int32;         /* == 32 bits */

typedef unsigned char uint8; /* == 8 bits */
typedef unsigned short uint16; /* == 16 bits */
typedef unsigned int uint32; /* == 32 bits */

typedef uint8 bits8;              /* >= 8 bits */
typedef uint16 bits16;            /* >= 16 bits */
typedef uint32 bits32;            /* >= 32 bits */

typedef long int int64;
typedef unsigned long int uint64;

typedef size_t Size; /* 8 bytes in 64-bit Linux */
```

大家很容易熟悉上面各种自定义的基础数据类型的命名规律。intXX 是有符号整数，可以表示负数和正数。uintXX 是无符号整数，只能表示正数，最小值为 0。其中 XX 为 8,16,32,64，表示这个数据类型有多少个比特 (bit)。Size 也是在源码中被大量使用的一个基础数据类型。在 64 位平台上它有 8 个字节的长度。

2.1.3 对源码文件的搜索

PostgreSQL 15 已经有了一百多万行的源代码，分布在 2000 多个 *.h 和 *.c 文件中。当我们研读一个数据结构，一个函数的时候，我们非常想快速搜索到它的定义所在的文件。我喜欢使用一种暴力搜索方式，如下所示：

```
$ pwd
/home/postgres/postgresql-15.2/src

/* 在当前目录和子目录中的所有*.h文件中搜索字符串 PageHeaderData */
$ find . -name *.h | xargs grep PageHeaderData

/* 在当前目录和子目录中的所有*.c文件中搜索字符串 ShmemInitStruct */
$ find . -name *.c | xargs grep ShmemInitStruct
```

当然，你可以在网上找到一些源码索引软件。doxygen.postgresql.org 是 PostgreSQL 官方提供的在线源码阅读网站，使用也非常方便。

2.1.4 内存对齐

我们知道，目前常见的计算机分为 32 位（32-bit）和 64 位（64-bit），早期的计算机还有 8-bit 和 16-bit 的。由于 IT 技术的迅猛发展，现在市场上的智能手机都是 64-bit 的了，更不要说更强大的服务器了，所以本书假设 PostgreSQL 运行在 64-bit 的计算机上。所谓 32-bit 计算机，指的是 CPU 的数据线和地址线都是 32 bits(4 个字节/byte)。64-bit 的计算机的 CPU 数据总线和地址总线是 64 bits（8 个字节/byte）。以 64-bit 计算机来说，CPU 一次性从内存中会读取 8 个字节。譬如你想访问 6 号地址的一个字节，则 CPU 一条读取指令就把 0 到 7 号地址的共计 8 个字节都读入 CPU 内部，然后只挑选 6 号的一个字节使用。如果你想读取 6 号到 9 号地址之间的 4 个字节，则 CPU 需要读取两次。第一次读取 0 到 7 号地址的 8 个字节，第二次读取 8 到 15 号地址的 8 个字节，共计 16 个字节，然后再在 CPU 内部拼接后获得 6 到 9 号的 4 个字节。这种操作无疑是低效率的。

为了提高 CPU 读写内存的速度，就产生了“对齐”的概念，其思想就是确保每次要访问的数据的起始地址和内存大小都是 8 的整数倍，也叫做按 8 字节对齐。PostgreSQL 的源代码中大量充斥着对齐的操作。我们来分析它的技术实现。其中使用最多的的 MAXALIGN 宏。下面是它的定义：

```
/* in src/include/pg_config.h */
#define MAXIMUM_ALIGNOF 8 /* 8个字节表示PG运行在64-bit操作系统上 */

/* in src/include/c.h */
#define TYPEALIGN(ALIGNVAL,LEN) \
    (((uintptr_t) (LEN) + ((ALIGNVAL) - 1)) & ~((uintptr_t) ((ALIGNVAL) - 1)))

#define MAXALIGN(LEN) TYPEALIGN(MAXIMUM_ALIGNOF, (LEN))
```

根据以上的定义，大家心算一下，就很容计算出如下公式：

```
MAXALIGN(x) = ((uintptr_t) (x) + 7) & ~((uintptr_t) (7))

/* ~((uintptr_t) (7) = 0xFFFFFFFFFFFFFFF8 */
```

那么 uintptr_t 又是什么鬼呢？它实际上是系统库定义的一种数据类型。大家可以在互联网上自行搜索它的定义。在 PostgreSQL 官方文档中，有这么一句话：

命题 2.1

Code in PostgreSQL should only rely on language features available in the C99 standard. ♠

它明确无误地告诉想为 PostgreSQL 添砖加瓦的 C 程序员，PostgreSQL 的源代码必须遵循 C99 的标准。这个标准是 20 多年前的产物了，已经比较古老了。之所以有这个规定，是为了确保 PostgreSQL 可以运行在各种操作系统上，包括比较古老的操作系统。在 C99 的标准中，uintptr_t 是系统库头文件 <stdint.h> 定义的一个变量。在 64-bit 平台上，你可以把它理解为一个 8 字节的无符号整数。ƫ((uintptr_t) (7) 则表示为 0xFFFFFFFFFFFFFFF8。由此我们可以看到，如果一个值 x 是 8 的整数倍，则 MAXALIGN(x) = x。如果 x 不是 8 的整数倍，MAXALIGN(x) 就往比它大的且是 8 的整数倍的那个数上凑。例如 x = 21，它介于 16 (= 2 X 8) 和 24 (= 3 X 8) 之间，它就往 24 上凑：MAXALIGN(21) = 24，果然是按 8 个字节对齐。MAXALIGN(x) 肯定是大于或等于 x 的，最多比 x 大 7。MAXALIGN(0) = 0。

我们在 PostgreSQL 的源代码中常常看到这样的代码：alignedSize = MAXALIGN(size); 其中 size 表示要分配的内存大小 (单位是 byte)。首先要通过类似的语句把它按 8 字节做齐，得到一个新的内存大小尺寸的变量 alignedSize。alignedSize 肯定就是 8 的整数倍了。这样申请下来的内存块的大小就是按照 8 字节对齐的了。虽然浪费了几个字节，但是提高了软件的性能。这种编程手法也值得我们借鉴和运用。PostgreSQL 的源码中还有其它类似的对齐定义的宏，如 CACHELINEALIGN 是按照 128 字节对齐的。我们在分析源码的时候遇到了再手动分析理解一下也不迟。这里就不过多介绍了。

2.2 PostgreSQL 数据文件的结构

我们都知道对于所有的数据库而言，真正的数据是存放在数据文件中的。对于数据文件结构的理解，是我们学习更加深入知识的前提。在本节中，我们展开对 PostgreSQL 数据文件结构的基本分析工作。

2.2.1 一个简单的实验

首先我们做一个简单的小实验，其过程如下所示：

```
$ id     /* 本实验以postgres用户直接登录到PG服务器上开始 */
uid=1000(postgres) gid=1000(postgres) groups=1000(postgres) ...
$ psql
psql (15.2)
Type "help" for help.

postgres=# CREATE DATABASE oracle; /* 创建一个叫做oracle的数据库 */
CREATE DATABASE
postgres=# \c oracle             /* 连接到oracle数据库中 */
You are now connected to database "oracle" as user "postgres".
oracle=# CREATE TABLE state(id INT, name CHAR(2)); /* 在oracle数据库中创建一个简单的表state，只有id和
    name两列 */
CREATE TABLE
oracle=# SELECT pg_relation_filepath('state'); /* 通过使用pg_relation_filepath()函数拿到表文件的路径 */
 pg_relation_filepath
----------------------
 base/16388/16389
(1 row)
```

```
oracle=# \! ls -l $PGDATA/base/16388/16389 /* 通过操作系统ls -l的命令查看这个文件在磁盘的信息 */
-rw------- 1 postgres postgres 0 Feb 24 14:38 /opt/data/pgdata1/base/16388/16389
/* 请注意上述文件的的大小，为0个字节，是因为这是一个空表，PG还没有为它分配真正的磁盘空间 */

oracle=# INSERT INTO state VALUES(0, 'TX'); /* 现在往state表中插入一条简单的记录 */
INSERT 0 1
oracle=# \! ls -l $PGDATA/base/16388/16389
-rw------- 1 postgres postgres 8192 Feb 24 14:40 /opt/data/pgdata1/base/16388/16389
/* 再次查看该数据文件，发现它的大小为8192个字节 */
```

在上述实验中，我用 C 语言的注释风格，用/* XXXX * 来注释命令和输出结果的含义或者需要关注的要点。大家可以很清晰地看到，一个表对应磁盘上的一个文件，且该文件的名字是纯数字。当一个表拥有了第一个记录后，它的大小变成了 8192 个字节。为什么是 8192 个字节呢？这是因为 PostgreSQL 的数据文件是按照 Block 来划分的，每一个 Block 的大小都是 8192 个字节。在 PostgreSQL 的源码中，有一个常量 BLCKSZ，它是 Block Size 的缩写。其定义如下：

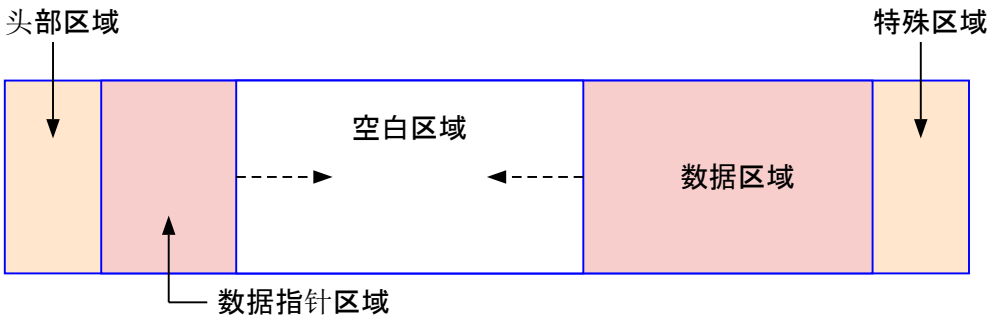
```
/* in src/include/pg_config.h */
#define BLCKSZ 8192
```

2.2.2 Block/Page 的结构分析

Block 和 Page 是 PostgreSQL 世界里面经常互相替换使用的两个术语。我们已经有了一个基本概念，即 PostgreSQL 的数据文件是按照 Block 划分的，其缺省大小为 8K（8 X 1024 = 8192 个字节）。当磁盘上的一个 Block 被读入到内存中后，内存中也会分配 8K 的空间来保存它，称之为 Page（页）。简而言之，在磁盘上是 Block，在内存中则为 Page，两者的内容是一模一样的。下面我们就来分析 Block 或者 Page 的基本结构。

我们可以把整个 Page 划分为四个区域：

Block/Page 的整体结构可以用下图来表示：在上图中，两个带箭头的虚线表明数据指针区域和数据区域的增



长方向。它们是相向而生长的。中间的空白部分是没有任何数据的空白区域。

2.2.3 页头 (Page Header) 之分析

我们来研究一下 Page Header 的具体组成。它是一个 C 语言的结构体（struct），叫做 PageHeaderData。其相关定义如下。