

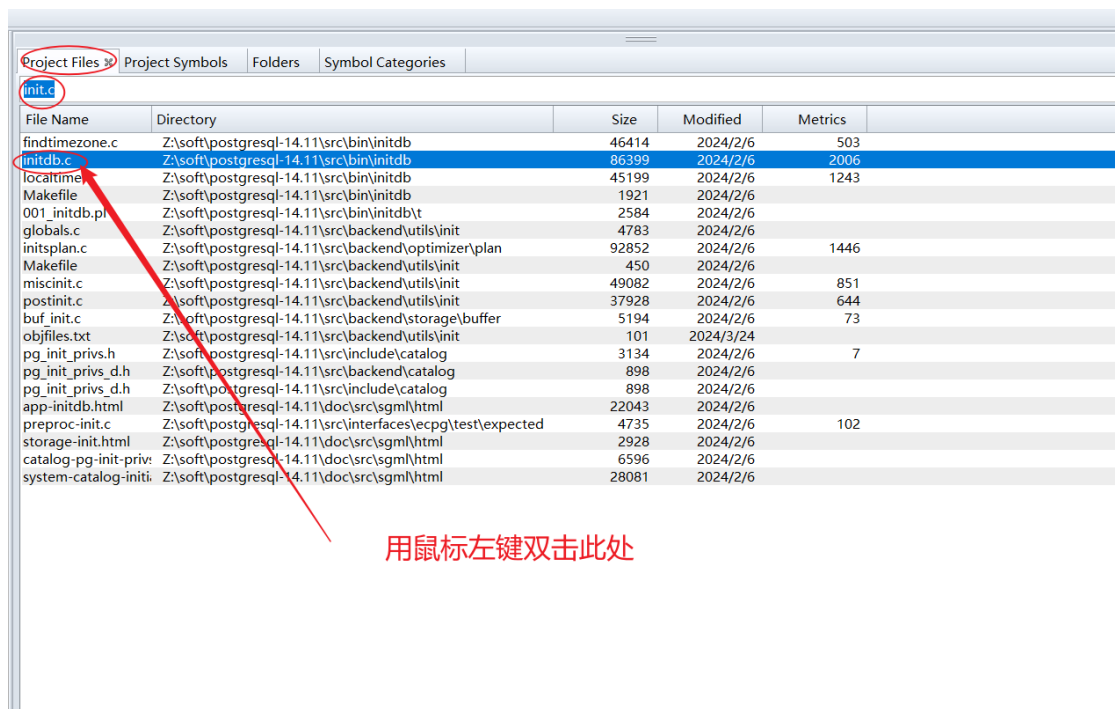
创建数据库集簇的命令 Initdb 源代码分析

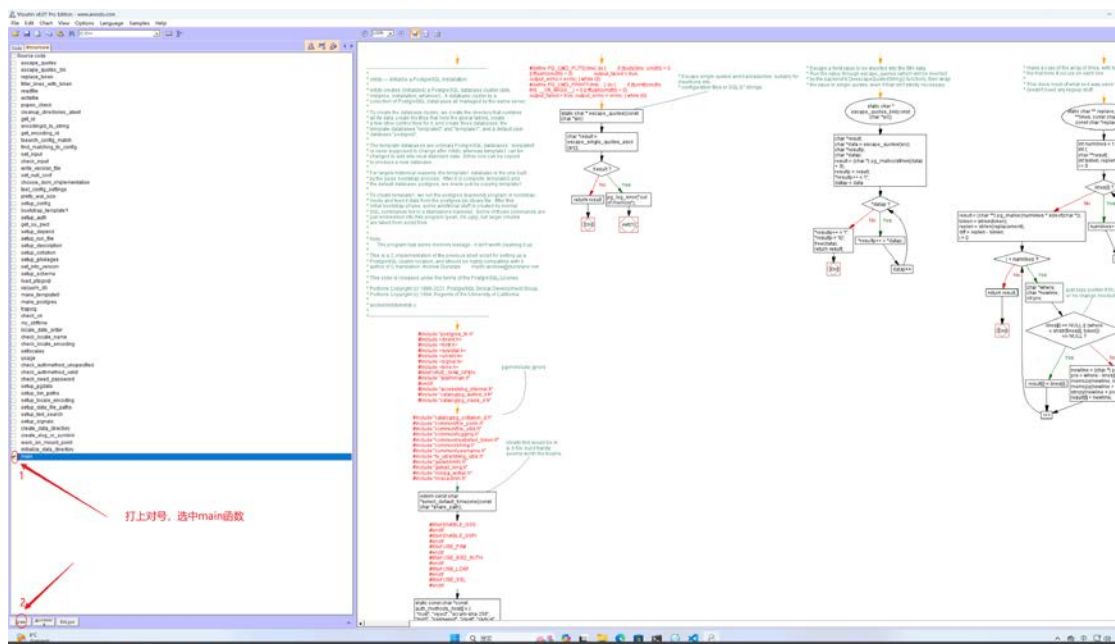
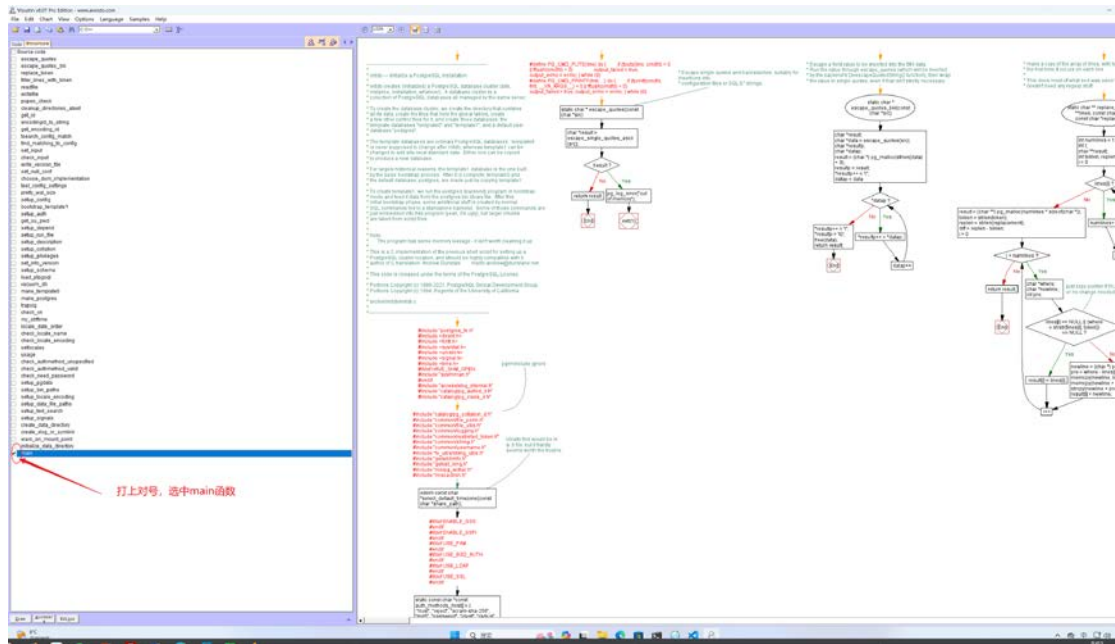
1 Initdb 概况

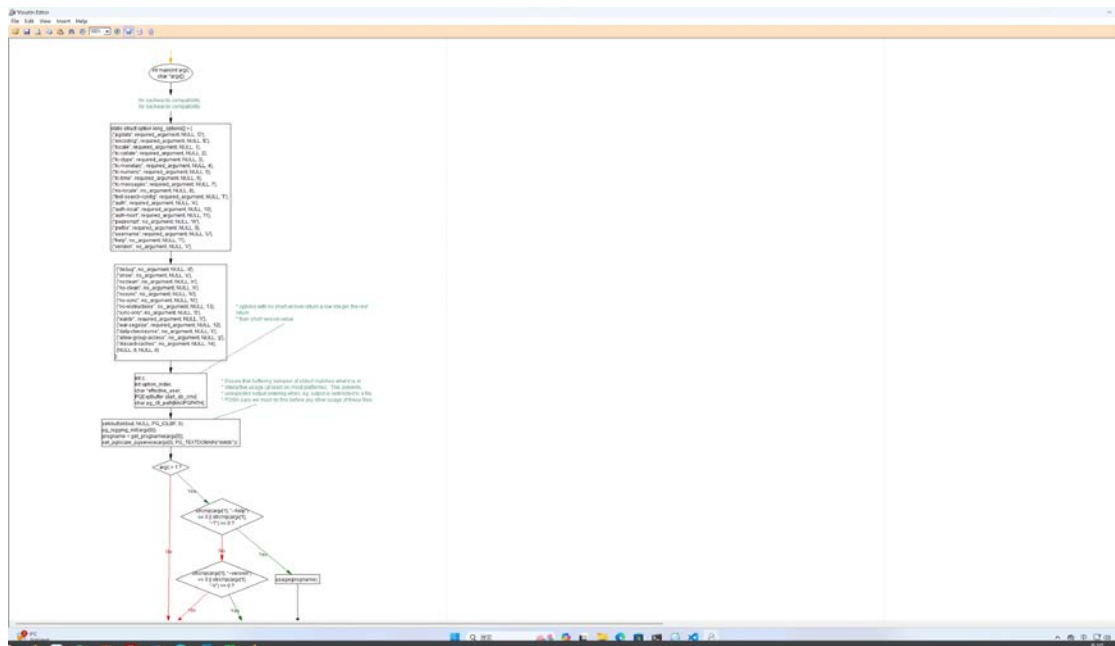
initdb 是 PostgreSQL 中一个独立的程序，它的主要工作就是对数据集簇进行初始化，创建模板数据库和系统表，并向系统表中插入初始元组。在这以后，用户创建各种数据库、表、视图、索引等数据库对象和进行其他操作时，都是在模板数据库和系统表的基础上进行的。

执行 initdb 程序时，将从 initdb.c 文件中的 main 函数开始执行。

在 source insight 工程窗口的“Project Files”标签，查找文件 initdb.c







main 函数的执行流程如图 2-3 所示。

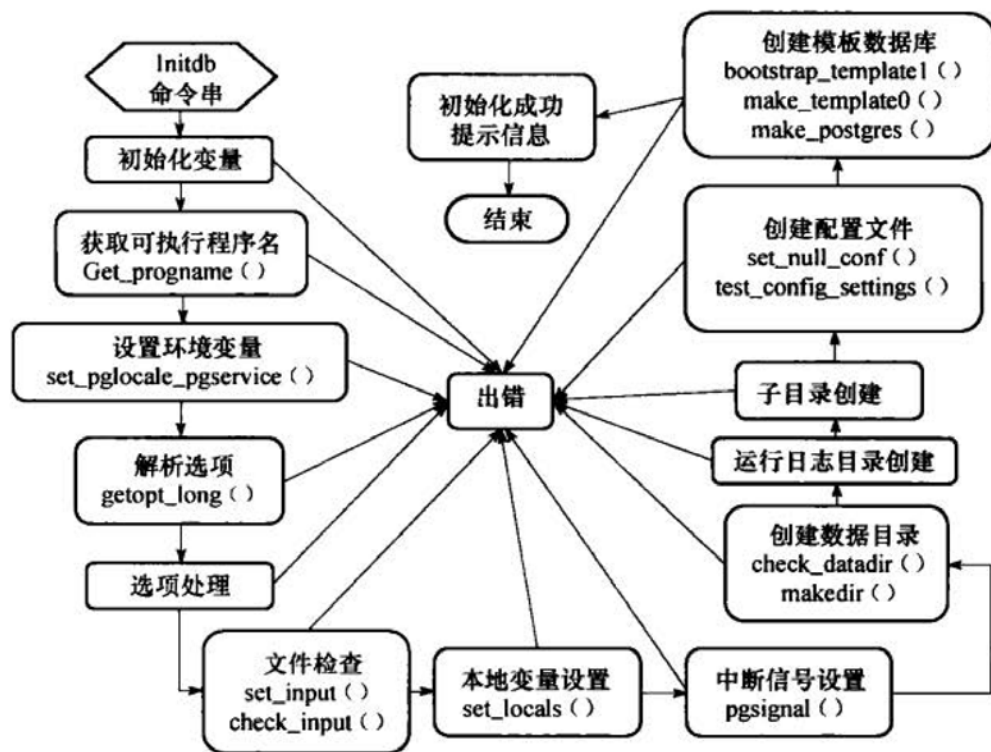


图 2-3 initdb 的运行流程

initdb 执行时将按照顺序执行下列工作：

- 1) 根据用户输入的命令行参数获取输入的命令名。
- 2) 设置系统编码为 LC_ALL，查找执行命令的绝对路径并设置该路径。
- 3) 设置环境变量(pg_data 等)，获取系统配置文件的源文件路径(postgres. bki、postgresql.conf. sample 等文件)，并检查该路径下各文件的可用性。
- 4) 设置中断信号处理函数，对终端命令行 SIGHUP、程序中中断 SIGINT、程序退出 SIGQUIT、软件中断 SIGTERM 和管道中断 SIGPIPE 等信号进行屏蔽，保证初始化工作顺利进行。
- 5) 创建数据目录，以及该目录下一些必要的子目录，如 base、global、base/等。
- 6) 测试当前服务器系统性能，由测试结果创建配置文件 postgresql.conf、pg_hba.conf、pg_ident.conf，并对其中定义的参数做一些设置。
- 7) 在 bootstrap 模式下创建数据库 template1，存储在数据目录的子目录 base/1/中。
- 8) 创建系统视图、系统表 TOAST 表等，复制 template1 来创建 template0 和 postgres，这些操作都用普通的 SQL 命令来完成。
- 9) 打印操作成功等相关信息，退出。

initdb 是 PostgreSQL 中一个独立的程序，它的主要工作就是对数据集簇进行初始化，创建模板数据库和系统表，并向系统表中插入初始元组。在这以后，用户创建各种数据库、表、视图、索引等数据库对象和进行其他操作时，都是在模板数据库和系统表的基础上进行的。

2 分析 initdb.c

2.1 基础

按照下面的文档，搭建调试环境。

北京科技大学计算机科学与技术系。

阅读 PostgreSQL 数据库源代码

搭建 PostgreSQL 源码调试环境。

如果 PostgreSQL 数据库正在运行：

```
[postgres@dbsvr db]$ pgps
postgres  51320      1  1 14:41 ?        00:00:00 /opt/db/pg14/bin/postgres
postgres  51321     51320  0 14:41 ?        00:00:00 postgres: logger
postgres  51323     51320  0 14:41 ?        00:00:00 postgres: checkpointer
postgres  51324     51320  0 14:41 ?        00:00:00 postgres: background writer
postgres  51325     51320  0 14:41 ?        00:00:00 postgres: walwriter
postgres  51326     51320  0 14:41 ?        00:00:00 postgres: autovacuum launcher
postgres  51327     51320  0 14:41 ?        00:00:00 postgres: archiver
postgres  51328     51320  0 14:41 ?        00:00:00 postgres: stats collector
postgres  51329     51320  0 14:41 ?        00:00:00 postgres: logical replication launcher
[postgres@dbsvr db]$
```

执行下面的命令，关闭 PostgreSQL 数据库：

```
[postgres@dbsvr db]$ pg_ctl stop
waiting for server to shut down... done
server stopped
[postgres@dbsvr db]$
```

执行下面的命令，删除数据库集簇：

```
[postgres@dbsvr db]$ rm -rf /opt/db/userdb/*
[postgres@dbsvr db]$
```


2.2 开始调试 initdb

```
[postgres@dbsvr ~]$ cd /opt/db/pgsql/bin
[postgres@dbsvr bin]$ gdb initdb -q
Reading symbols from initdb...
(gdb) info sources
/opt/db/pg14/bin/initdb:
(Full debug information has not yet been read for this file.)
/opt/db/soft/postgresql-14.11/src/port/thread.c, /opt/db/soft/postgresql-14.11/src/port/strerror.c,
/opt/db/soft/postgresql-14.11/src/port/snprintf.c, /opt/db/soft/postgresql-14.11/src/port/quotes.c,
/opt/db/soft/postgresql-14.11/src/port/pqsignal.c, /opt/db/soft/postgresql-14.11/src/port/pgstrsignal.c,
/opt/db/soft/postgresql-14.11/src/port/pgstrcasecmp.c, /opt/db/soft/postgresql-14.11/src/port/pgmkdirp.c,
/opt/db/soft/postgresql-14.11/src/port/pgcheckdir.c, /opt/db/soft/postgresql-14.11/src/port/path.c,
/opt/db/soft/postgresql-14.11/src/port/chklocale.c, /opt/db/soft/postgresql-14.11/src/port/strncpy.c,
/opt/db/soft/postgresql-14.11/src/common/logging.c, /opt/db/soft/postgresql-14.11/src/common/sprompt.c,
/opt/db/soft/postgresql-14.11/src/common/restricted_token.c, /opt/db/soft/postgresql-14.11/src/common/fe_memutils.c,
/opt/db/soft/postgresql-14.11/src/common/wait_error.c, /opt/db/soft/postgresql-14.11/src/common/username.c,
/opt/db/soft/postgresql-14.11/src/common/stringinfo.c, /opt/db/soft/postgresql-14.11/src/common/string.c,
/opt/db/soft/postgresql-14.11/src/common/rmtree.c, /opt/db/soft/postgresql-14.11/src/common/psprintf.c,
/opt/db/soft/postgresql-14.11/src/common/pgfnames.c, /opt/db/soft/postgresql-14.11/src/common/pg_get_line.c,
/opt/db/soft/postgresql-14.11/src/include/common/kwlookup.h, /opt/db/soft/postgresql-14.11/src/common/kwlookup.c,
/opt/db/soft/postgresql-14.11/src/common/kwlist_d.h, /opt/db/soft/postgresql-14.11/src/common/keywords.c,
/opt/db/soft/postgresql-14.11/src/common/file_utils.c, /opt/db/soft/postgresql-14.11/src/common/file_perm.c,
/opt/db/soft/postgresql-14.11/src/common/exec.c, /opt/db/soft/postgresql-14.11/src/common/encnames.c,
/opt/db/soft/postgresql-14.11/src/fe_utils/string_utils.c, /opt/db/soft/postgresql-14.11/src/timezone/localtime.c,
/opt/db/soft/postgresql-14.11/src/bin/initdb/initdb.c, /opt/db/soft/postgresql-14.11/src/bin/initdb/findtimezone.c
(gdb) list main
2912     }
2913
2914
2915     int
2916     main(int argc, char *argv[])
2917     {
2918         static struct option long_options[] = {
2919             {"pgdata", required_argument, NULL, 'D'},
2920             {"encoding", required_argument, NULL, 'E'},
2921             {"locale", required_argument, NULL, 1},
(gdb) (按回车键)
2922             {"lc-collate", required_argument, NULL, 2},
2923             {"lc-ctype", required_argument, NULL, 3},
2924             {"lc-monetary", required_argument, NULL, 4},
2925             {"lc-numeric", required_argument, NULL, 5},
2926             {"lc-time", required_argument, NULL, 6},
2927             {"lc-messages", required_argument, NULL, 7},
2928             {"no-locale", no_argument, NULL, 8},
2929             {"text-search-config", required_argument, NULL, 'T'},
2930             {"auth", required_argument, NULL, 'A'},
2931             {"auth-local", required_argument, NULL, 10},
(gdb) (按回车键)
2932             {"auth-host", required_argument, NULL, 11},
2933             {"pwprompt", no_argument, NULL, 'W'},
2934             {"pwfile", required_argument, NULL, 9},
2935             {"username", required_argument, NULL, 'U'},
2936             {"help", no_argument, NULL, '?'},
```

```

2937         {"version", no_argument, NULL, 'V'},
2938         {"debug", no_argument, NULL, 'd'},
2939         {"show", no_argument, NULL, 's'},
2940         {"noclean", no_argument, NULL, 'n'},    /* for backwards
compatibility */
2941         {"no-clean", no_argument, NULL, 'n'},
(gdb) (按回车键)
2942         {"nosync", no_argument, NULL, 'N'}, /* for backwards compatibility */
2943         {"no-sync", no_argument, NULL, 'N'},
2944         {"no-instructions", no_argument, NULL, 13},
2945         {"sync-only", no_argument, NULL, 'S'},
2946         {"waldir", required_argument, NULL, 'X'},
2947         {"wal-segsize", required_argument, NULL, 12},
2948         {"data-checksums", no_argument, NULL, 'k'},
2949         {"allow-group-access", no_argument, NULL, 'g'},
2950         {"discard-caches", no_argument, NULL, 14},
2951         {NULL, 0, NULL, 0}
(gdb) (按回车键)
2952     };
2953
2954     /*
2955     * options with no short version return a low integer, the rest return
2956     * their short version value
2957     */
2958     int                c;
2959     int                option_index;
2960     char               *effective_user;
2961     PQExpBuffer start_db_cmd;
(gdb) (按回车键)
2962     char               pg_ctl_path[MAXPGPATH];
2963
2964     /*
2965     * Ensure that buffering behavior of stdout matches what it is in
2966     * interactive usage (at least on most platforms). This prevents
2967     * unexpected output ordering when, eg, output is redirected to a file.
2968     * POSIX says we must do this before any other usage of these files.
2969     */
2970     setvbuf(stdout, NULL, PG_IOLBF, 0);
2971
(gdb) (按回车键)
2972     pg_logging_init(argv[0]);
2973     progname = get_progname(argv[0]);
2974     set_pglocale_pgservice(argv[0], PG_TEXTDOMAIN("initdb"));
2975
2976     if (argc > 1)
2977     {
2978         if (strcmp(argv[1], "--help") == 0 || strcmp(argv[1], "-?") == 0)
2979         {
2980             usage(progname);
2981             exit(0);
(gdb) (按回车键)

```

```

2982         }
2983         if (strcmp(argv[1], "--version") == 0 || strcmp(argv[1], "-V") == 0)
2984         {
2985             puts("initdb (PostgreSQL) " PG_VERSION);
2986             exit(0);
2987         }
2988     }
2989
2990     /* process command-line options */
2991
(gdb)

```

执行下面的 gdb 命令，设置断点：

```

(gdb) break main
Breakpoint 1 at 0x408cab: file initdb.c, line 2970.
(gdb)

```

执行下面的 gdb 命令，运行 initdb：

```

(gdb) run -D /opt/db/userdb/pgdata -E UTF8 -U postgres -W
Starting program: /opt/db/pg14/bin/initdb -D /opt/db/userdb/pgdata -E UTF8 -U postgres -W
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib64/libthread_db.so.1".

Breakpoint 1, main (argc=8, argv=0x7fffffffefb68) at initdb.c:2970
2970         setvbuf(stdout, NULL, PG_IOLBF, 0);
(gdb)

```

2.3 setvbuf(stdout, NULL, PG_IOLBF, 0);

```

(gdb) list 2965,2970
2965         * Ensure that buffering behavior of stdout matches what it is in
2966         * interactive usage (at least on most platforms).  This prevents
2967         * unexpected output ordering when, eg, output is redirected to a file.
2968         * POSIX says we must do this before any other usage of these files.
2969         */
2970         setvbuf(stdout, NULL, PG_IOLBF, 0);
(gdb)

```

第 2965 至 2969 行：这段注释说明了正在进行的操作。它在确保标准输出(stdout)的缓冲行为与交互式使用时相匹配，以防止在输出被重定向到文件时出现意外的输出顺序。通过

调用 `setvbuf()` 函数来实现这一点。

`setvbuf()` 函数用于设置流的缓冲方式。在这里，它被用于设置 `stdout` 的缓冲方式为行缓冲(`PG_IOLBF`)，第二个参数为 `NULL` 表示使用系统默认大小的缓冲区，最后一个参数是缓冲区的大小，这里设置为 0 表示由系统自动决定缓冲区的大小。

第 2970 行：调用 `setvbuf()` 来设置了 `stdout` 的缓冲方式。

```
(gdb) break 2971
Breakpoint 2 at 0x408cc9: file initdb.c, line 2972.
(gdb) c
Continuing.

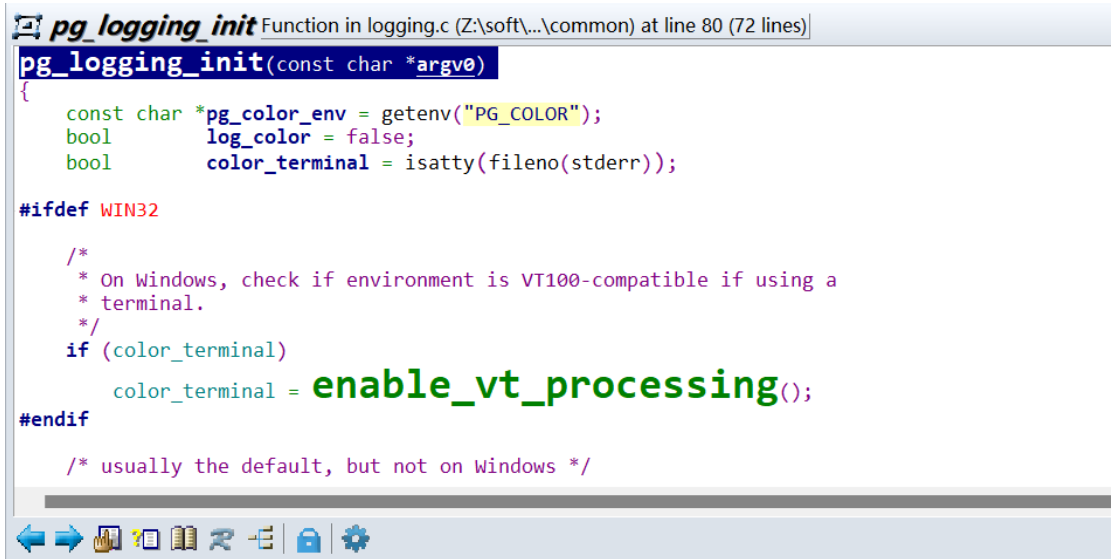
Breakpoint 2, main (argc=8, argv=0x7fffffffeac8) at initdb.c:2972
2972         pg_logging_init(argv[0]);
(gdb) next
2973         progname = get_progname(argv[0]);
(gdb)
```

2.4 日志系统初始化 `pg_logging_init()`

```
2972         pg_logging_init(argv[0]);
```

第 2972 行：调用 `pg_logging_init()` 函数进行日志系统的初始化，这可能会设置一些日志相关的配置。

可以使用 `source insight` 选中 2972 行的函数 `pg_logging_init()`，在上下文窗口查看该函数的定义。



```
pg_logging_init Function in logging.c (Z:\soft\...\common) at line 80 (72 lines)
pg_logging_init(const char *argv0)
{
    const char *pg_color_env = getenv("PG_COLOR");
    bool log_color = false;
    bool color_terminal = isatty(fileno(stderr));

#ifdef WIN32
    /*
     * On Windows, check if environment is VT100-compatible if using a
     * terminal.
     */
    if (color_terminal)
        color_terminal = enable_vt_processing();
#endif

    /* usually the default, but not on Windows */
}
```

2.5 获取程序的名字 get_progname()

```
2973         progname = get_progname(argv[0]);
```

第 2973 行：调用 `get_progname()` 函数获取程序的名称，这个名称通常会在程序中用于记录日志或者错误信息。

```
(gdb) n
2974         set_pglocale_pgservice(argv[0], PG_TEXTDOMAIN("initdb"));
(gdb) print progname
$2 = 0x42e2c0 "initdb"
(gdb)
```

2.6 设置程序的本地化信息 set_pglocale_pgservice()

```
2974         set_pglocale_pgservice(argv[0], PG_TEXTDOMAIN("initdb"));
```

第 2974 行：调用 `set_pglocale_pgservice()` 函数来设置程序的本地化信息，其中包括了 `initdb` 程序的域名（domain），这可能会影响到程序中本地化相关的输出。

```
(gdb) n
2976         if (argc > 1)
(gdb)
```

2.7 命令行参数解析

```
(gdb) list 2976,3103
2976         if (argc > 1)
2977         {
2978             if (strcmp(argv[1], "--help") == 0 || strcmp(argv[1], "-?") == 0)
2979             {
2980                 usage(progname);
2981                 exit(0);
2982             }
2983             if (strcmp(argv[1], "--version") == 0 || strcmp(argv[1], "-v") == 0)
2984             {
2985                 puts("initdb (PostgreSQL) " PG_VERSION);
2986                 exit(0);
2987             }
2988         }
2989
2990         /* process command-line options */
2991
2992         while ((c = getopt_long(argc, argv, "A:dD:E:gkL:nNsST:U:WX:", long_options,
&option_index)) != -1)
2993         {
2994             switch (c)
2995             {
2996                 case 'A':
2997                     authmethodlocal = authmethodhost = pg_strdup(optarg);
2998
2999                     /*
3000                      * When ident is specified, use peer for local
connections.
3001                      * Mirrored, when peer is specified, use ident for
TCP/IP
3002                      * connections.
3003                      */
3004                     if (strcmp(authmethodhost, "ident") == 0)
3005                         authmethodlocal = "peer";
3006                     else if (strcmp(authmethodlocal, "peer") == 0)
3007                         authmethodhost = "ident";
3008                     break;
3009                 case 10:
3010                     authmethodlocal = pg_strdup(optarg);
3011                     break;
3012                 case 11:
3013                     authmethodhost = pg_strdup(optarg);
3014                     break;
3015                 case 'D':
3016                     pg_data = pg_strdup(optarg);
3017                     break;
```

```
3018             case 'E':
3019                 encoding = pg_strdup(optarg);
3020                 break;
3021             case 'W':
3022                 pwprompt = true;
3023                 break;
3024             case 'U':
3025                 username = pg_strdup(optarg);
3026                 break;
3027             case 'd':
3028                 debug = true;
3029                 printf(_("Running in debug mode.\n"));
3030                 break;
3031             case 'n':
3032                 noclean = true;
3033                 printf(_("Running in no-clean mode.  Mistakes will not
be cleaned up.\n"));
3034                 break;
3035             case 'N':
3036                 do_sync = false;
3037                 break;
3038             case 'S':
3039                 sync_only = true;
3040                 break;
3041             case 'k':
3042                 data_checksums = true;
3043                 break;
3044             case 'L':
3045                 share_path = pg_strdup(optarg);
3046                 break;
3047             case 1:
3048                 locale = pg_strdup(optarg);
3049                 break;
3050             case 2:
3051                 lc_collate = pg_strdup(optarg);
3052                 break;
3053             case 3:
3054                 lc_ctype = pg_strdup(optarg);
3055                 break;
3056             case 4:
3057                 lc_monetary = pg_strdup(optarg);
3058                 break;
3059             case 5:
3060                 lc_numeric = pg_strdup(optarg);
3061                 break;
3062             case 6:
3063                 lc_time = pg_strdup(optarg);
3064                 break;
3065             case 7:
3066                 lc_messages = pg_strdup(optarg);
3067                 break;
```



```

3068             case 8:
3069                 locale = "C";
3070                 break;
3071             case 9:
3072                 pwfilename = pg_strdup(optarg);
3073                 break;
3074             case 's':
3075                 show_setting = true;
3076                 break;
3077             case 'T':
3078                 default_text_search_config = pg_strdup(optarg);
3079                 break;
3080             case 'X':
3081                 xlog_dir = pg_strdup(optarg);
3082                 break;
3083             case 12:
3084                 str_wal_segment_size_mb = pg_strdup(optarg);
3085                 break;
3086             case 13:
3087                 noinstructions = true;
3088                 break;
3089             case 'g':
3090                 SetDataDirectoryCreatePerm(PG_DIR_MODE_GROUP);
3091                 break;
3092             case 14:
3093                 extra_options = psprintf("%s %s",
3094
extra_options,
3095                                     "-c
debug_discard_caches=1");
3096                 break;
3097             default:
3098                 /* getopt_long already emitted a complaint */
3099                 fprintf(stderr, _("Try \"%s --help\" for more
information.\n"),
3100                             progname);
3101                 exit(1);
3102             }
3103     }
(gdb)

```

首先，代码检查是否有至少一个命令行参数 (`argc > 1`)。

如果有参数，它会进入一个条件语句，检查命令行参数是否包含 `--help` 或 `-?`，如果是的话，它会调用 `usage(progname)` 函数打印程序的使用帮助信息，并调用 `exit(0)` 退出程序。

接着，它检查命令行参数是否包含 `--version` 或 `-V`，如果是，则打印版本信息并退出程序。

紧接着，在 `while` 循环中，程序使用 `getopt_long` 函数来逐个解析命令行选项。在 `switch` 语句中，根据不同的选项执行相应的操作。比如，对于选项 'A'，它会将 `authmethodlocal` 和 `authmethodhost` 设置为相应的值，并进行一些额外的逻辑判断。

对于未知的选项，程序会打印错误信息，并调用 `exit(1)` 退出程序。

这段代码的作用是解析命令行参数，并根据不同的选项执行相应的操作，比如设置认证方法、设置数据目录、设置编码等。

```
(gdb) break 3104
Breakpoint 3 at 0x409133: file initdb.c, line 3110.
(gdb) c
Continuing.

Breakpoint 3, main (argc=8, argv=0x7ffffffeac8) at initdb.c:3110
3110         if (optind < argc && !pg_data)
(gdb)
```

```
(gdb) list 3111,3123
3104
3105
3106     /*
3107      * Non-option argument specifies data directory as long as it wasn't
3108      * already specified with -D / --pgdata
3109      */
3110     if (optind < argc && !pg_data)
3111     {
3112         pg_data = pg_strdup(argv[optind]);
3113         optind++;
3114     }
3115
3116     if (optind < argc)
3117     {
3118         pg_log_error("too many command-line arguments (first is \"%s\")",
3119                     argv[optind]);
3120         fprintf(stderr, _("Try \"%s --help\" for more information.\n"),
3121                 progname);
3122         exit(1);
3123     }
(gdb)
```

行号 3104-3123: 这部分代码主要处理非选项参数。首先, 在程序的参数列表中, 非选项参数通常是用来指定某些必要的参数值, 比如文件名、目录名等。在这段代码中, 首先判断是否还有剩余的非选项参数需要处理, 如果存在, 则进行处理; 如果不存在, 则说明参数已经处理完毕。

行号 3107-3110: 这里是一个注释, 说明非选项参数通常用来指定数据目录, 但是前提是数据目录尚未通过 `-D` 或 `--pgdata` 参数指定。

行号 3110-3114: 如果还有剩余的非选项参数, 并且 `pg_data` 尚未被设置, 那么将当前非选项参数赋值给 `pg_data`, 然后将 `optind` 加一, 表示已经处理了一个参数。

行号 3116-3123: 如果在处理完非选项参数后, 还有多余的参数剩余, 那么说明参数传递有误, 打印错误信息并提示用户查看帮助文档, 然后退出程序。

2.8 注册一个在退出时执行的函数

```
(gdb) list 3124,3126
3124
3125         atexit(cleanup_directories_atexit);
3126
(gdb)
```

行号 3124-3125: 注册一个在程序退出时执行的函数 `cleanup_directories_atexit`, 用于在程序结束时清理目录。

2.9 如果设置了 `sync_only` 标志, 执行文件同步操作

```
(gdb) list 3127,3145
3127         /* If we only need to fsync, just do it and exit */
3128         if (sync_only)
3129         {
3130             setup_pgdata();
3131
3132             /* must check that directory is readable */
3133             if (pg_check_dir(pg_data) <= 0)
```

```

3134          {
3135              pg_log_error("could not access directory \"%s\": %m",
pg_data);
3136              exit(1);
3137          }
3138
3139          fputs_("syncing data to disk ... "), stdout);
3140          fflush(stdout);
3141          fsync_pgdata(pg_data, PG_VERSION_NUM);
3142          check_ok();
3143          return 0;
3144      }
3145
(gdb)

```

行号 3127-3145: 如果设置了 `sync_only` 标志, 表示只需要执行文件同步操作, 则执行以下操作:

- 调用 `setup_pgdata()` 函数设置数据目录。
- 检查数据目录是否可读, 如果不可读则打印错误信息并退出程序。
- 打印同步数据到磁盘的提示信息。
- 调用 `fsync_pgdata()` 函数执行数据同步操作。
- 调用 `check_ok()` 函数, 通知操作成功完成。
- 返回退出码 0, 表示正常退出。

2.10 如果同时设置了 `pwprompt` 和 `pwfilename` 标志

```

(gdb) list 3146,3151
3146          if (pwprompt && pwfilename)
3147          {
3148              pg_log_error("password prompt and password file cannot be specified
together");
3149              exit(1);
3150          }
3151
(gdb)

```

行号 3146-3150: 如果同时设置了 `pwprompt` 和 `pwfilename` 标志, 表示用户同时指定了密码提示和密码文件, 这是不允许的, 因此打印错误信息并退出程序。

2.11 对认证方法进行验证

```
(gdb) list 3152,3159
3152     check_authmethod_unspecified(&authmethodlocal);
3153     check_authmethod_unspecified(&authmethodhost);
3154
3155     check_authmethod_valid(authmethodlocal, auth_methods_local, "local");
3156     check_authmethod_valid(authmethodhost, auth_methods_host, "host");
3157
3158     check_need_password(authmethodlocal, authmethodhost);
3159
(gdb)
```

行号 3152-3158：对认证方法进行验证：

- 首先，调用 `check_authmethod_unspecified()` 函数检查认证方法是否已经指定，如果未指定则打印警告信息。
- 然后，调用 `check_authmethod_valid()` 函数验证认证方法的有效性，比如本地认证方法是否合法等。
- 最后，调用 `check_need_password()` 函数检查是否需要输入密码。

2.12 设置 WAL 段的大小

```
(gdb) list 3159,3182
3159
3160     /* set wal segment size */
3161     if (str_wal_segment_size_mb == NULL)
3162         wal_segment_size_mb = (DEFAULT_XLOG_SEG_SIZE) / (1024 * 1024);
3163     else
3164     {
3165         char      *endptr;
3166
3167         /* check that the argument is a number */
3168         wal_segment_size_mb = strtol(str_wal_segment_size_mb, &endptr, 10);
3169
3170         /* verify that wal segment size is valid */
3171         if (endptr == str_wal_segment_size_mb || *endptr != '\0')
3172         {
3173             pg_log_error("argument of --wal-segsize must be a number");
```

```

3174             exit(1);
3175         }
3176         if (!IsValidWalSegSize(wal_segment_size_mb * 1024 * 1024))
3177         {
3178             pg_log_error("argument of --wal-segsize must be a power of 2
between 1 and 1024");
3179             exit(1);
3180         }
3181     }
3182
(gdb)

```

行号 3159-3181：这部分代码用于设置 WAL 段的大小。

行号 3160-3162：首先检查 `str_wal_segment_size_mb` 是否为 `NULL`，如果为 `NULL`，表示未指定 WAL 段大小，则将 `wal_segment_size_mb` 设置为默认值，这里的默认值是 `DEFAULT_XLOG_SEG_SIZE`（默认的 WAL 段大小），除以 `1024 * 1024`，即将其转换为以兆字节（MB）为单位。

行号 3163-3180：如果指定了 `str_wal_segment_size_mb`，则需要将其解析为整数，并进行有效性验证。

行号 3165-3171：使用 `strtol()` 函数将 `str_wal_segment_size_mb` 转换为整数，同时使用 `endptr` 来检查是否转换成功。如果转换失败（即 `endptr` 和 `str_wal_segment_size_mb` 相同，或者 `endptr` 指向非空字符），则打印错误信息，并退出程序。

行号 3176-3179：验证 WAL 段大小是否有效。调用 `IsValidWalSegSize()` 函数来判断 WAL 段大小是否为有效值，如果不是有效值，则打印错误信息，并退出程序。

2.13 获取受限制的令牌

```

(gdb) list 3183,3384
3183         get_restricted_token();
3184
(gdb)

```

行号 3183：调用 `get_restricted_token()` 函数。这个函数用于获取受限制的令牌。

2.14 设置 PostgreSQL 数据目录

```
(gdb) list 3185,3186
3185         setup_pgdata();
3186
(gdb)
```

行号 3185：调用 `setup_pgdata()` 函数。这个函数用于设置 PostgreSQL 数据目录。

2.15 设置 PostgreSQL 的二进制文件路径

```
(gdb) list 3187,3188
3187         setup_bin_paths(argv[0]);
3188
(gdb)
```

行号 3187：调用 `setup_bin_paths(argv[0])` 函数。这个函数用于设置 PostgreSQL 的二进制文件路径。

2.16 获取当前用户的标识符

```
(gdb) list 3189,3203
3189         effective_user = get_id();
3190         if (!username)
3191             username = effective_user;
3192
3193         if (strncmp(username, "pg_", 3) == 0)
3194         {
3195             pg_log_error("superuser name \"%s\" is disallowed; role names cannot
begin with \"pg_\"", username);
```



```
3196             exit(1);
3197         }
3198
3199         printf(_("The files belonging to this database system will be owned "
3200                "by user \"%s\".\n"
3201                "This user must also own the server process.\n\n"),
3202                effective_user);
3203
3204 (gdb)
```

行号 3189: 调用 `get_id()` 函数, 并将返回的值赋给 `effective_user` 变量。这个函数用于获取当前用户的标识符。

行号 3190-3191: 如果 `username` 为空, 则将 `effective_user` 的值赋给 `username`。这是为了确保 `username` 变量有值。

行号 3193-3197: 检查 `username` 是否以 "pg_" 开头。如果是, 表示用户名不合法, 打印错误信息并退出程序。

行号 3199-3202: 打印一条消息, 指示数据库系统中的文件将由哪个用户所有。消息包含了 `effective_user` 的值。

```
3204         set_info_version();
3205
3206         setup_data_file_paths();
3207
3208         setup_locale_encoding();
3209
3210         setup_text_search();
3211
3212 (gdb)
```

2.17 设置 PostgreSQL 的信息版本

```
(gdb) list 3204,3205
3204         set_info_version();
3205
3206 (gdb)
```

行号 3204: 调用 `set_info_version()` 函数。这个函数用于设置 PostgreSQL 的信息版本。

2.18 设置数据文件的路径

```
(gdb) list 3206,3207
3206         setup_data_file_paths();
3207
(gdb)
```

行号 3206: 调用 `setup_data_file_paths()` 函数。这个函数用于设置数据文件的路径。

2.19 设置区域和编码

```
(gdb) list 3208,3209
3208         setup_locale_encoding();
3209
(gdb)
```

行号 3208: 调用 `setup_locale_encoding()` 函数。这个函数用于设置区域和编码。

2.20 设置文本搜索相关的配置

```
(gdb) list 3210,3211
3210         setup_text_search();
3211
(gdb)
```

行号 3210: 调用 `setup_text_search()` 函数。这个函数用于设置文本搜索相关的配置。

2.21 检查是否支持数据页校验和

```
(gdb) list 3212,3218
3212         printf("\n");
3213
3214         if (data_checksums)
3215             printf(_("Data page checksums are enabled.\n"));
3216         else
3217             printf(_("Data page checksums are disabled.\n"));
3218
(gdb)
```

行号 3214: 这是一个条件语句的开始, 检查变量 `data_checksums` 是否为真。

如果 `data_checksums` 为真, 则打印一条消息表示数据页校验和已启用。这里使用了 `printf` 函数和 `_("")` 宏来支持国际化和本地化, 确保消息可以被翻译成其他语言。

如果 `data_checksums` 不为真, 则打印一条消息表示数据页校验和已禁用。

2.22 检查是否需要获取超级用户密码

```
(gdb) list 3219,3223
3219         if (pwprompt || pwfilename)
3220             get_su_pwd();
3221
3222         printf("\n");
3223
(gdb)
```

行号 3219: 这是另一个条件语句的开始, 检查 `pwprompt` 或 `pwfilename` 是否为真。如果其中任何一个为真, 则调用 `get_su_pwd()` 函数。这个函数可能用于获取超级用户的密码, 这取决于程序的具体逻辑。

2.23 初始化数据目录 (`initialize_data_directory()`)

```
(gdb) list 3224,3225
```

```
3224         initialize_data_directory();
3225
(gdb)
```

行号 3224: 调用 `initialize_data_directory()` 函数，初始化数据目录。

2.24 检查变量 `do_sync`

```
(gdb) list 3226,3235
3226         if (do_sync)
3227         {
3228             fputs(_("syncing data to disk ... "), stdout);
3229             fflush(stdout);
3230             fsync_pgdata(pg_data, PG_VERSION_NUM);
3231             check_ok();
3232         }
3233         else
3234             printf(_("\nSync to disk skipped.\nThe data directory might become
corrupt if the operating system crashes.\n"));
3235
(gdb)
```

行号 3226: 这是一个条件语句的开始，检查变量 `do_sync` 是否为真。

如果 `do_sync` 为真，则执行下面的代码块：

- 行号 3228: 在执行同步操作之前，在标准输出中打印一条消息表示正在将数据同步到磁盘，并将缓冲区的内容刷新到标准输出流。
- 行号 3230: 调用 `fsync_pgdata()` 函数，该函数可能用于将数据目录的内容同步到磁盘上，参数 `pg_data` 是数据目录的路径，`PG_VERSION_NUM` 是 PostgreSQL 的版本号。
- 行号 3231: 调用 `check_ok()` 函数，这可能用于检查操作是否成功，并在成功时进行一些操作。

如果 `do_sync` 为假，则执行 `else` 语句中的代码块：

- 在标准输出中打印一条消息，表示同步到磁盘的操作被跳过了。这条消息提醒用户，如果操作系统崩溃，数据目录可能会变得损坏。

2.25 检查变量 authwarning

```
(gdb) list 3236,3243
3236         if (authwarning)
3237         {
3238             printf("\n");
3239             pg_log_warning("enabling \"trust\" authentication for local
connections");
3240             fprintf(stderr, _("You can change this by editing pg_hba.conf or
using the option -A, or\n"
3241                             "--auth-local and --auth-host, the
next time you run initdb.\n"));
3242         }
3243
(gdb)
```

行号 3236: 是一个条件语句的开始, 检查变量 `authwarning` 是否为真。

如果为真, 则执行下面的代码块。

行号 3238: 在标准输出中打印一个空行。

行号 3239: 调用 `pg_log_warning()` 函数, 打印一条警告消息, 说明正在启用 “trust” 认证以允许本地连接。

行号 3240-3241: 在标准错误输出中打印一条消息, 提醒用户可以通过编辑 `pg_hba.conf` 文件或使用 `-A` 选项来修改认证设置。

2.26 检查变量 noinstructions

```
(gdb) list 3244,3279
3244         if (!noinstructions)
3245         {
3246             /*
3247              * Build up a shell command to tell the user how to start the server
3248              */
3249             start_db_cmd = createPQExpBuffer();
3250
```

```

3251      /* Get directory specification used to start initdb ... */
3252      strcpy(pg_ctl_path, argv[0], sizeof(pg_ctl_path));
3253      canonicalize_path(pg_ctl_path);
3254      get_parent_directory(pg_ctl_path);
3255      /* ... and tag on pg_ctl instead */
3256      join_path_components(pg_ctl_path, pg_ctl_path, "pg_ctl");
3257
3258      /* Convert the path to use native separators */
3259      make_native_path(pg_ctl_path);
3260
3261      /* path to pg_ctl, properly quoted */
3262      appendString(start_db_cmd, pg_ctl_path);
3263
3264      /* add -D switch, with properly quoted data directory */
3265      appendPQExpBufferStr(start_db_cmd, " -D ");
3266      appendString(start_db_cmd, pgdata_native);
3267
3268      /* add suggested -l switch and "start" command */
3269      /* translator: This is a placeholder in a shell command. */
3270      appendPQExpBuffer(start_db_cmd, " -l %s start", _("logfile"));
3271
3272      printf(_("\nSuccess. You can now start the database server
using:\n\n"
3273              "    %s\n\n"),
3274             start_db_cmd->data);
3275
3276      destroyPQExpBuffer(start_db_cmd);
3277      }
3278
3279
(gdb)

```

行号 3244: 这是一个条件语句的开始, 检查变量 `noinstructions` 是否为假 (即为真)。

行号 3246-3269: 在 `noinstructions` 为假时执行的代码块。该代码块的作用是构建一个 `shell` 命令, 用于告知用户如何启动数据库服务器。

行号 3249: 声明一个 `start_db_cmd` 变量, 用于存储构建的 `shell` 命令。

行号 3251-3259: 获取 `pg_ctl` 可执行文件的路径, 并将路径转换为本地格式。

行号 3261-3269: 构建启动数据库服务器的 `shell` 命令, 包括 `pg_ctl` 可执行文件路径、数据目录路径、日志文件路径等信息。

行号 3272-3274: 在成功构建 `shell` 命令后, 打印成功消息, 并在消息中包含构建的 `shell` 命令。

行号 3276: 释放 `start_db_cmd` 所占用的内存。

2.27 设置成功标志并退出 `initdb`

```
(gdb) list 3280,3282
3280         success = true;
3281         return 0;
3282     }
(gdb)
```

行号 3278-3281: 设置成功标志为真，并返回退出码 0，表示程序正常结束。

3 `initialize_data_directory()` 函数

```
(gdb) list 2807,2818
2807         /*
2808          * Set mask based on requested PGDATA permissions.  pg_mode_mask, and
2809          * friends like pg_dir_create_mode, are set to owner-only by default and
2810          * then updated if -g is passed in by calling SetDataDirectoryCreatePerm()
2811          * when parsing our options (see above).
2812          */
2813         umask(pg_mode_mask);
2814
2815         create_data_directory();
2816
2817         create_xlog_or_symlink();
2818
2819         /* Create required subdirectories (other than pg_wal) */
2820         printf(_("creating subdirectories ... "));
2821         fflush(stdout);
2822
2823         for (i = 0; i < lengthof(subdirs); i++)
2824         {
2825             char        *path;
2826
2827             path = psprintf("%s/%s", pg_data, subdirs[i]);
2828
```



```

2829          /*
2830          * The parent directory already exists, so we only need mkdir() not
2831          * pg_mkdir_p() here, which avoids some failure modes; cf bug #13853.
2832          */
2833          if (mkdir(path, pg_dir_create_mode) < 0)
2834          {
2835              pg_log_error("could not create directory \"%s\": %m", path);
2836              exit(1);
2837          }
2838
2839          free(path);
2840      }
2841
2842      check_ok();
2843
2844      /* Top level PG_VERSION is checked by bootstrapper, so make it first */
2845      write_version_file(NULL);
2846
2847      /* Select suitable configuration settings */
2848      set_null_conf();
2849      test_config_settings();
2850
2851      /* Now create all the text config files */
2852      setup_config();
2853
2854      /* Bootstrap templatel */
2855      bootstrap_templatel();
2856
2857      /*
2858      * Make the per-database PG_VERSION for templatel only after init'ing it
2859      */
2860      write_version_file("base/1");
2861
2862      /*
2863      * Create the stuff we don't need to use bootstrap mode for, using a
2864      * backend running in simple standalone mode.
2865      */
2866      fputs_("performing post-bootstrap initialization ... "), stdout);
2867      fflush(stdout);
2868
2869      snprintf(cmd, sizeof(cmd),
2870              "\"%s\" %s %s templatel >%s",
2871              backend_exec, backend_options, extra_options,
2872              DEVNULL);
2873
2874      PG_CMD_OPEN;
2875
2876      setup_auth(cmdfd);
2877
2878      setup_run_file(cmdfd, system_constraints_file);
2879

```

```

2880     setup_run_file(cmdfd, system_functions_file);
2881
2882     setup_depend(cmdfd);
2883
2884     /*
2885      * Note that no objects created after setup_depend() will be "pinned".
2886      * They are all droppable at the whim of the DBA.
2887      */
2888
2889     setup_run_file(cmdfd, system_views_file);
2890
2891     setup_description(cmdfd);
2892
2893     setup_collation(cmdfd);
2894
2895     setup_run_file(cmdfd, dictionary_file);
2896
2897     setup_privileges(cmdfd);
2898
2899     setup_schema(cmdfd);
2900
2901     load_plpgsql(cmdfd);
2902
2903     vacuum_db(cmdfd);
2904
2905     make_template0(cmdfd);
2906
2907     make_postgres(cmdfd);
2908
2909     PG_CMD_CLOSE;
2910
2911     check_ok();
2912 }
(gdb)

```

```

(gdb) list 2800,2804
2800     initialize_data_directory(void)
2801     {
2802         PG_CMD_DECL;
2803         int i;
2804
(gdb)

```

行 2799-2800: 定义了一个名为 `initialize_data_directory` 的函数，它没有参数，也没有返

返回值。

行 2802: 使用了宏 `PG_CMD_DECL`，可能用于声明 PostgreSQL 命令相关的变量。

行 2803: 声明了一个整型变量 `i`，用于循环计数。

3.1 设置信号处理

```
(gdb) list 2805,2806
2805         setup_signals();
2806
(gdb)
```

行 2805: 调用了 `setup_signals()` 函数，用于设置信号处理。

3.2 创建 PostgreSQL 数据目录及其子目录

```
(gdb) list 2807,2843
2807         /*
2808          * Set mask based on requested PGDATA permissions.  pg_mode_mask, and
2809          * friends like pg_dir_create_mode, are set to owner-only by default and
2810          * then updated if -g is passed in by calling SetDataDirectoryCreatePerm()
2811          * when parsing our options (see above).
2812          */
2813         umask(pg_mode_mask);
2814
2815         create_data_directory();
2816
2817         create_xlog_or_symlink();
2818
2819         /* Create required subdirectories (other than pg_wal) */
2820         printf(_("creating subdirectories ... "));
2821         fflush(stdout);
2822
2823         for (i = 0; i < lengthof(subdirs); i++)
2824         {
2825             char        *path;
2826
2827             path = psprintf("%s/%s", pg_data, subdirs[i]);
2828
2829             /*
2830              * The parent directory already exists, so we only need mkdir() not
2831              * pg_mkdir_p() here, which avoids some failure modes; cf bug #13853.
```

```
2832         */
2833         if (mkdir(path, pg_dir_create_mode) < 0)
2834         {
2835             pg_log_error("could not create directory \"%s\": %m", path);
2836             exit(1);
2837         }
2838
2839         free(path);
2840     }
2841
2842     check_ok();
2843
2844 (gdb)
```

行 2807-2811: 设置了文件创建时的掩码, 即 `umask`, 用于确定新文件的默认权限。

行 2815: 调用了 `create_data_directory()` 函数, 用于创建数据目录。

行 2817: 调用了 `create_xlog_or_symlink()` 函数, 可能用于创建 WAL 日志目录或符号链接。

行 2823-2840: 进入一个循环, 用于创建数据目录中的子目录。在循环中, 构造子目录的路径, 并使用 `mkdir()` 函数创建子目录。如果创建失败, 则输出错误信息并退出程序。

数组 `subdirs[i]`, 包括了要创建的子目录名字。

行 2842: 执行一些检查操作。

3.3 写入 PG_VERSION 文件

```
(gdb) list 2844,2846
2844     /* Top level PG_VERSION is checked by bootstrapper, so make it first */
2845     write_version_file(NULL);
2846
2847 (gdb)
```

行 2844: 写入 PG_VERSION 文件。

3.4 创建 PostgreSQL 的配置文件

```
(gdb) list 2847,2853
2847      /* Select suitable configuration settings */
2848      set_null_conf();
2849      test_config_settings();
2850
2851      /* Now create all the text config files */
2852      setup_config();
2853
(gdb)
```

行 2847-2850: 设置空配置并测试配置设置。

行 2851: 设置文本配置文件。

3.5 创建模板数据库 template1

```
(gdb) list 2854,2861
2854      /* Bootstrap template1 */
2855      bootstrap_template1();
2856
2857      /*
2858      * Make the per-database PG_VERSION for template1 only after init'ing it
2859      */
2860      write_version_file("base/1");
2861
(gdb)
```

行 2855: 运行 BKI 脚本以创建 template1 数据库

行 2858-2860: 为 template1 创建 PG_VERSION 文件。

3.6 执行创建模板数据库 template1 之后的一系列操作（需要挖细节）

```
(gdb) list 2862,2913
2862      /*
2863      * Create the stuff we don't need to use bootstrap mode for, using a
```

```
2864      * backend running in simple standalone mode.
2865      */
2866      fputs(_("performing post-bootstrap initialization ... "), stdout);
2867      fflush(stdout);
2868
2869      snprintf(cmd, sizeof(cmd),
2870               "\\\"%s\\\" %s %s template1 >%s",
2871               backend_exec, backend_options, extra_options,
2872               DEVNULL);
2873
2874      PG_CMD_OPEN;
2875
2876      setup_auth(cmdfd);
2877
2878      setup_run_file(cmdfd, system_constraints_file);
2879
2880      setup_run_file(cmdfd, system_functions_file);
2881
2882      setup_depend(cmdfd);
2883
2884      /*
2885       * Note that no objects created after setup_depend() will be "pinned".
2886       * They are all droppable at the whim of the DBA.
2887       */
2888
2889      setup_run_file(cmdfd, system_views_file);
2890
2891      setup_description(cmdfd);
2892
2893      setup_collation(cmdfd);
2894
2895      setup_run_file(cmdfd, dictionary_file);
2896
2897      setup_privileges(cmdfd);
2898
2899      setup_schema(cmdfd);
2900
2901      load_plpgsql(cmdfd);
2902
2903      vacuum_db(cmdfd);
2904
2905      make_template0(cmdfd);
2906
2907      make_postgres(cmdfd);
2908
2909      PG_CMD_CLOSE;
2910
2911      check_ok();
2912  }
2913
2914  (gdb)
```

行 2862-2867: 输出一些提示信息到标准输出, 并构造一个命令字符串 `cmd`。

行 2869-2911: 打开 PostgreSQL 命令处理的管道, 并执行一系列操作, 包括设置权限、运行文件、视图等。最后关闭管道并再次执行一些检查操作。

4 bootstrap_template1()函数

该函数用于运行 BKI 脚本以创建 `template1` 数据库

```
(gdb) list bootstrap_template1
1347  /*
1348   * run the BKI script in bootstrap mode to create template1
1349   */
1350  static void
1351  bootstrap_template1(void)
1352  {
1353      PG_CMD_DECL;
1354      char    **line;
1355      char    **bki_lines;
1356      char      headerline[MAXPGPATH];
(gdb)
1357      char      buf[64];
1358
1359      printf(_("running bootstrap script ... "));
1360      fflush(stdout);
1361
1362      bki_lines = readfile(bki_file);
1363
1364      /* Check that bki file appears to be of the right version */
1365
1366      snprintf(headerline, sizeof(headerline), "# PostgreSQL %s\n",
(gdb)
1367              PG_MAJORVERSION);
1368
1369      if (strcmp(headerline, *bki_lines) != 0)
1370      {
1371          pg_log_error("input file \"%s\" does not belong to PostgreSQL %s",
1372                      bki_file, PG_VERSION);
1373          fprintf(stderr,
1374                  _("Check your installation or specify the correct path
1375                      using the option -L. \n"));
1376          exit(1);
(gdb)
1377      }
1378
```



```

1379      /* Substitute for various symbols used in the BKI file */
1380
1381      sprintf(buf, "%d", NAMEDATALEN);
1382      bki_lines = replace_token(bki_lines, "NAMEDATALEN", buf);
1383
1384      sprintf(buf, "%d", (int) sizeof(Pointer));
1385      bki_lines = replace_token(bki_lines, "SIZEOF_POINTER", buf);
1386
1387      bki_lines = replace_token(bki_lines, "ALIGNOF_POINTER",
1388                                (sizeof(Pointer) == 4) ?
1389                                "i" : "d");
1390
1391      bki_lines = replace_token(bki_lines, "FLOAT8PASSBYVAL",
1392                                FLOAT8PASSBYVAL ? "true" :
1393                                "false");
1394
1395      bki_lines = replace_token(bki_lines, "POSTGRES",
1396                                escape_quotes_bki(username));
1397
1398      bki_lines = replace_token(bki_lines, "ENCODING",
1399                                (gdb)
1400                                encodingid_to_string(encodingid));
1401
1402      bki_lines = replace_token(bki_lines, "LC_COLLATE",
1403                                escape_quotes_bki(lc_collate));
1404
1405      bki_lines = replace_token(bki_lines, "LC_CTYPE",
1406                                escape_quotes_bki(lc_ctype));
1407
1408      /* Also ensure backend isn't confused by this environment var: */
1409      unsetenv("PGCLIENTENCODING");
1410
1411      snprintf(cmd, sizeof(cmd),
1412               "\\\"%s\\\" --boot -x1 -X %u %s %s %s %s",
1413               backend_exec,
1414               wal_segment_size_mb * (1024 * 1024),
1415               data_checksums ? "-k" : "",
1416               boot_options, extra_options,
1417               debug ? "-d 5" : "");
1418
1419      PG_CMD_OPEN;
1420
1421      for (line = bki_lines; *line != NULL; line++)
1422      {
1423          PG_CMD_PUTS(*line);

```

```
1422         free(*line);
1423     }
1424
1425     PG_CMD_CLOSE;
1426
1427     (gdb)
1427     free(bki_lines);
1428
1429     check_ok();
1430 }
1431
```

行 1347-1351: 注释说明, 定义了一个静态函数 `bootstrap_template1`, 该函数用于运行 BKI 脚本以创建 `template1` 数据库。

行 1353-1358: 声明了变量 `line` 和 `bki_lines`, 以及一个字符数组 `headerline` 和 `buf`。
`headerline` 用于存储 BKI 文件的头部信息。

行 1359-1360: 输出提示信息到标准输出, 指示正在运行引导脚本。

行 1362: 调用 `readfile(bki_file)` 函数读取 BKI 文件的内容, 并将结果存储在 `bki_lines` 中。

行 1366-1376: 检查 BKI 文件的头部信息是否匹配 PostgreSQL 的版本号, 若不匹配则输出错误信息并退出程序。

行 1381-1405: 替换 BKI 文件中的一些标记符号, 如 `NAMEDATALEN`、`SIZEOF_POINTER` 等。

行 1406: 取消 `PGCLIENTENCODING` 环境变量的设置, 以确保后端不会受到其影响。

行 1408-1414: 构造了一个命令字符串 `cmd`, 该命令用于执行后端程序以引导 `template1` 数据库的创建。

行 1417-1425: 通过循环逐行向后端程序发送 BKI 文件的内容, 并释放每行的内存。

行 1427: 释放 `bki_lines` 所占用的内存。

行 1429: 执行一些检查操作, 可能是检查引导是否成功。

这个函数主要用于执行 BKI 脚本以创建 `template1` 数据库, 并在过程中对一些标记符号进行替换。

5 setup_run_file()函数——系统视图和数据字典

```
setup_run_file(cmdfd, system_views_file)
```

```
setup_run_file(cmdfd, dictionary_file)
```

6 make_template0()函数

创建 template0 数据库

7 make_postgres()函数

创建 postgres 数据库

8 打点调试示例（函数 initialize_datra_directory）

```
[postgres@dbsvr ~]$ cd /opt/db/pgsql/bin
[postgres@dbsvr bin]$ rm -rf /opt/db/userdb/*
[postgres@dbsvr bin]$
[postgres@dbsvr bin]$ gdb -q /opt/db/pgsql/bin/initdb
Reading symbols from /opt/db/pgsql/bin/initdb...
(gdb) break initialize_data_directory
Breakpoint 1 at 0x4089fb: file initdb.c, line 2805.
(gdb) run -D /opt/db/userdb/pgdata -E UTF8 -U postgres -W
Starting program: /opt/db/pg14/bin/initdb -D /opt/db/userdb/pgdata -E UTF8 -U postgres -W
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib64/libthread_db.so.1".
[Detaching after vfork from child process 91378]
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 text search configuration will be set to "english".
```

Data page checksums are disabled.

Enter new superuser password: (输入密码 “dba123”)

Enter it again: (输入密码 “dba123”)

Breakpoint 1, initialize_data_directory () at initdb.c:2805

2805 setup_signals();

Missing separate debuginfos, use: dnf debuginfo-install sssd-2.6.1-1.oe2203.x86_64
(gdb)

我测试了这些断点

```
(gdb) info break
Num    Type           Disp Enb Address          What
1      breakpoint     keep y   0x00000000004089fb in initialize_data_directory at initdb.c:2805
      breakpoint already hit 1 time
2      breakpoint     keep y   0x0000000000408a12 in initialize_data_directory at initdb.c:2817
      breakpoint already hit 1 time
3      breakpoint     keep y   0x0000000000408ada in initialize_data_directory at initdb.c:2842
      breakpoint already hit 1 time
4      breakpoint     keep y   0x0000000000408ae9 in initialize_data_directory at initdb.c:2848
      breakpoint already hit 1 time
5      breakpoint     keep y   0x0000000000408af8 in initialize_data_directory at initdb.c:2855
      breakpoint already hit 1 time
6      breakpoint     keep y   0x0000000000408b07 in initialize_data_directory at initdb.c:2866
      breakpoint already hit 1 time
(gdb) |
```

大家还可以更细节一些。