PostgreSQL数据库源码浅析

----源码解读与开发入门

陈刚

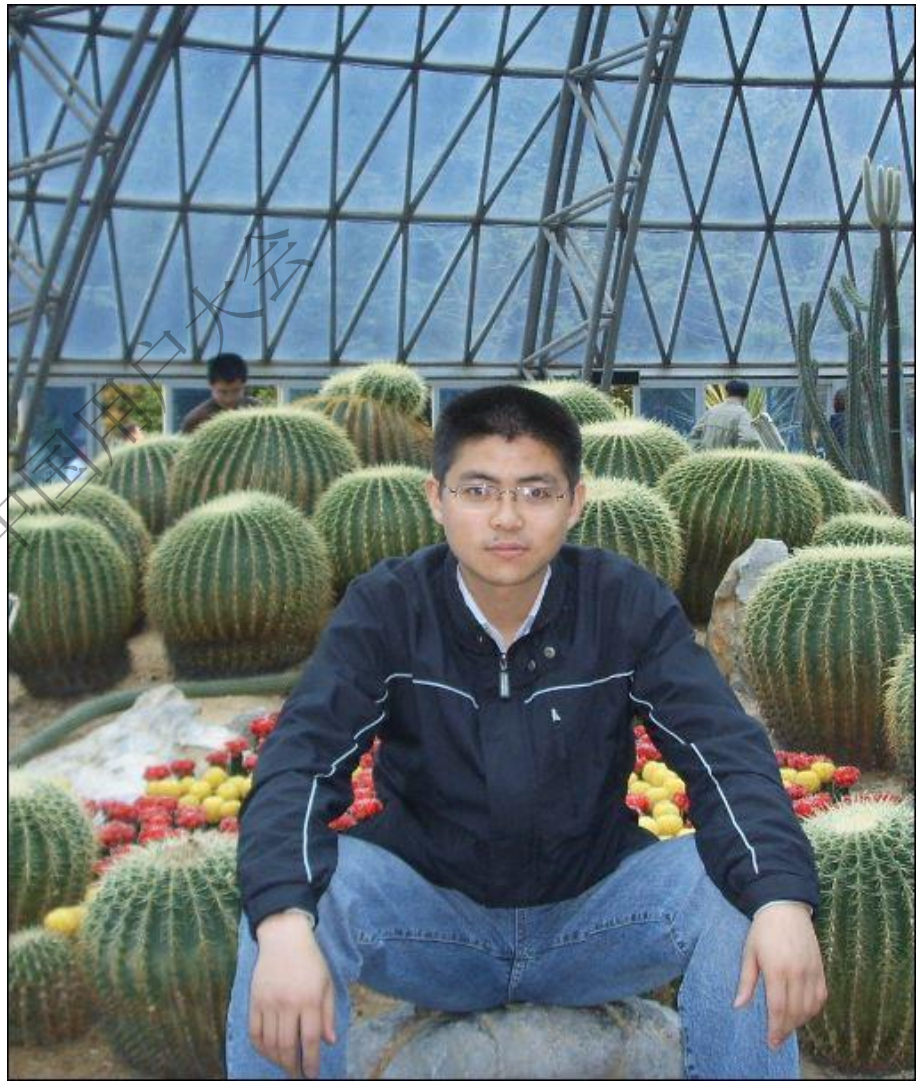
平安科技

中国平安
保险·银行·投资

平安科技
PING AN TECHNOLOGY

陈 刚

- 现就职于中国平安集团旗下平安科技
- 数据库技术专家
- 从事数据库相关工作12年，2011年加入平安科技，主要负责Oracle、PostgreSQL、MySQL、Redis、Mongodb、TimesTen等多种数据库的管理和架构设计工作。近期比较痴迷于PostgreSQL数据库源码研究，并开始从源码层面学习PG数据库。



分享大纲

- ✓ 源码概况
- ✓ 源码研究意义

- ✓ 环境说明
- ✓ 方法与心得

- ✓ 插件开发
- ✓ 案例介绍

分享大纲

PostgreSQL版权特点和源码研究的意义

PostgreSQL源码概况--V9.6.0

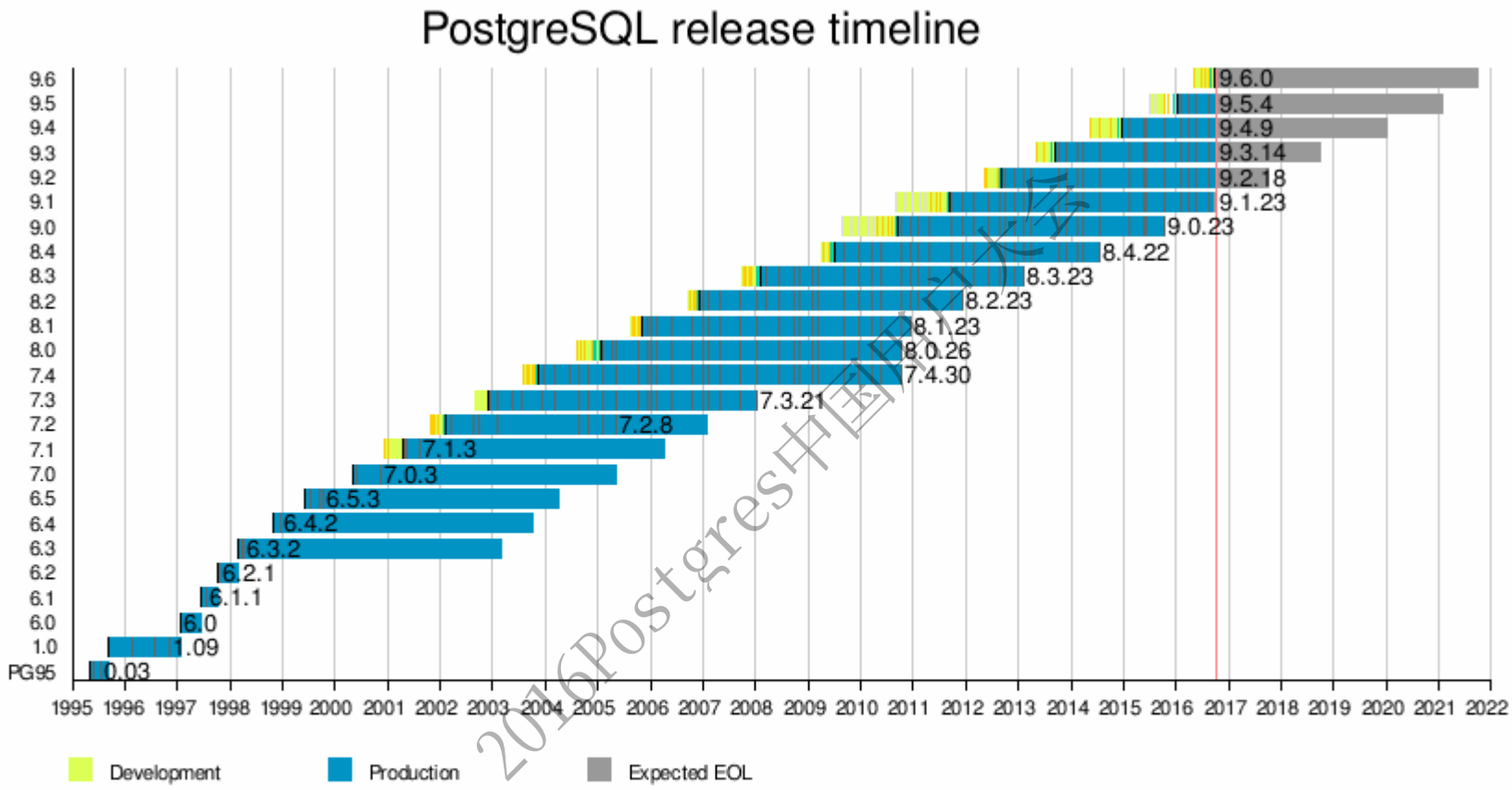
- C语言开发
- 最新版本9.6.0 (2016年9月29日发布)
- 9.6.0版本核心代码共124.9万行，有效代码72.7万行,注释31.9万，comment lines/code lines=0.43 （使用understand工具统计）
- 约30个应用

Project Metrics	
Files:	1723
Program Units:	17103
Lines:	**1249865**
Blank Lines:	143783
Code Lines:	**727284**
Comment Lines:	**319120**

Project Metrics	
Files:	192
Program Units:	2118
Lines:	**97721**
Blank Lines:	13647
Code Lines:	**59565**
Comment Lines:	**18358**

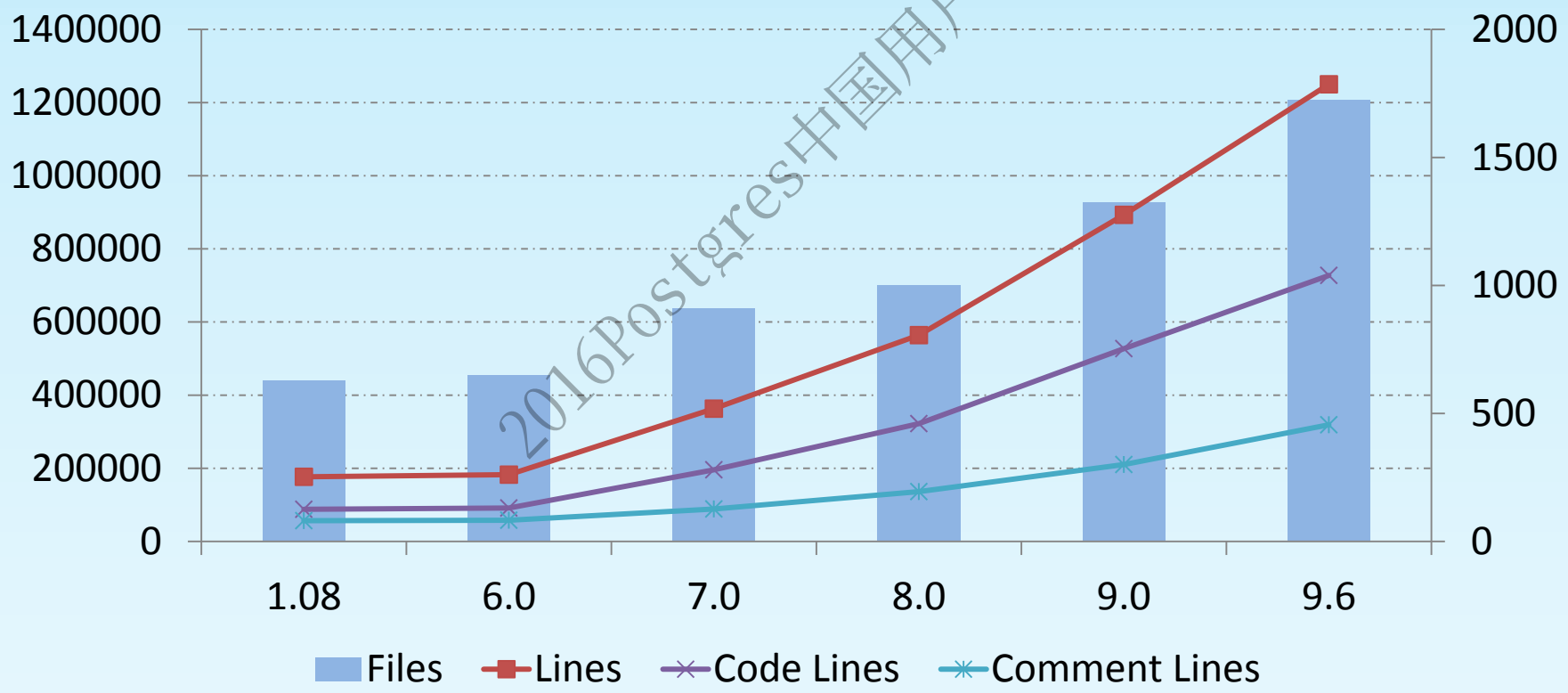
- 插件数量46个
- 插件代码行数9.7万
- 另外还包括非常多第三方组织或个人开发的插件

PG各版本时间线



PostgreSQL源码概况--各版本代码量

版本号	V1.08	V6.0	V7.0	V8.0	V9.0	V9.6
Files	628	647	909	1001	1325	1723
Lines	176755	182469	362868	564266	893033	1249865
Blank Lines	21217	21937	43458	66873	102206	143783
Code Lines	88056	91219	195774	322165	527030	727284
Comment Lines	56531	57801	88614	136169	210429	319120



分享大纲

windows开发调试环境

环境说明：

- OS：win7 64bit
- VS：visual studio2013
- Perl：ActivePerl-5.24.0.2400-MSWin32-x86-64int-300558.exe
- Bison: bison-2.4.1-setup.exe
- Flex: flex-2.5.4a-1.exe

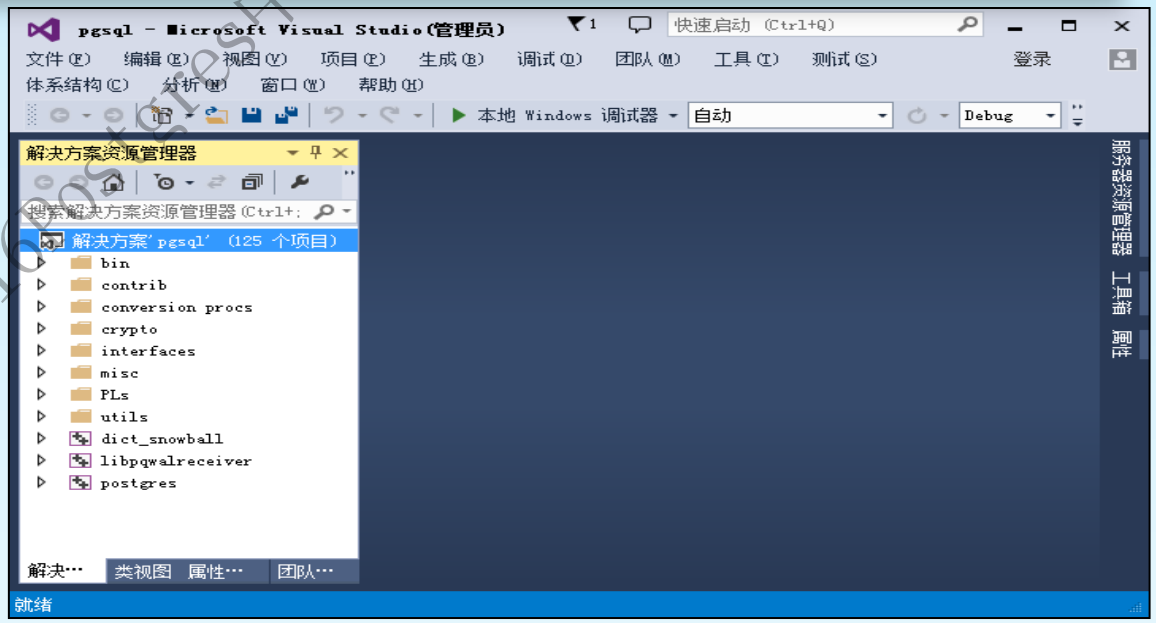
工程导入/编译/安装等方法

参见src\tools\msvc\目录下

相关脚本

辅助工具：

- understand
- source insight



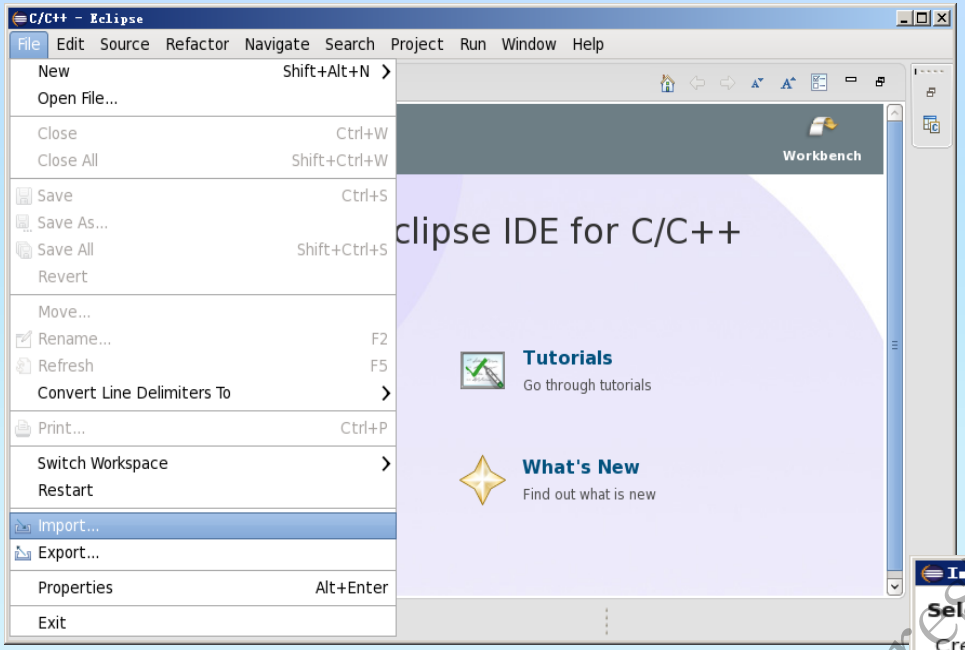
Linux开发调试环境

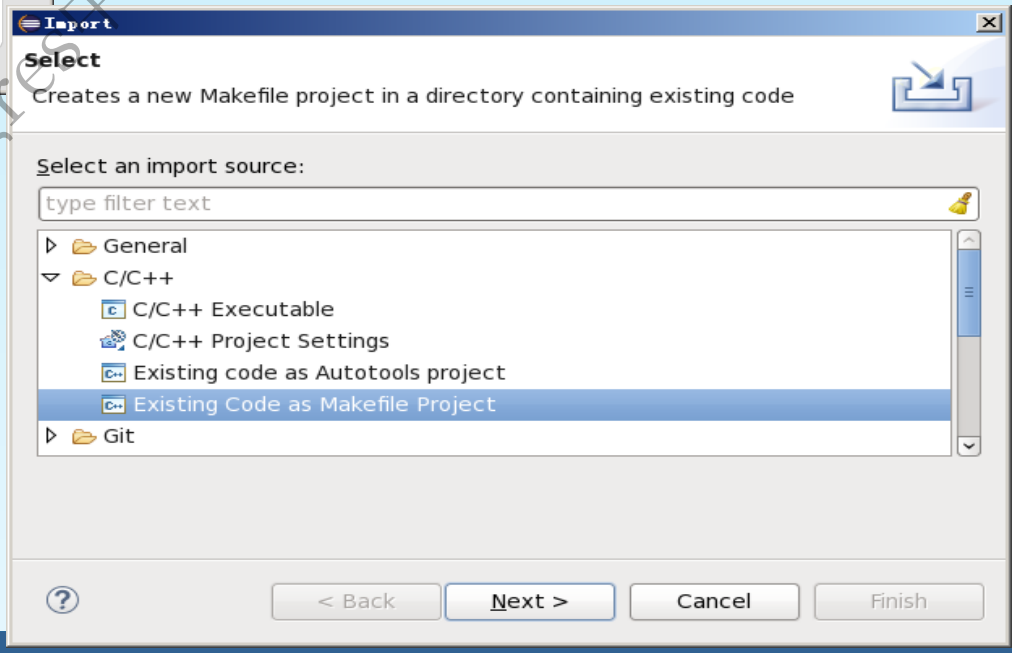
环境说明：

- OS：Linux 6.7 64bit
- Eclipse：eclipse-cpp-mars-2-linux-gtk-x86_64.tar.gz

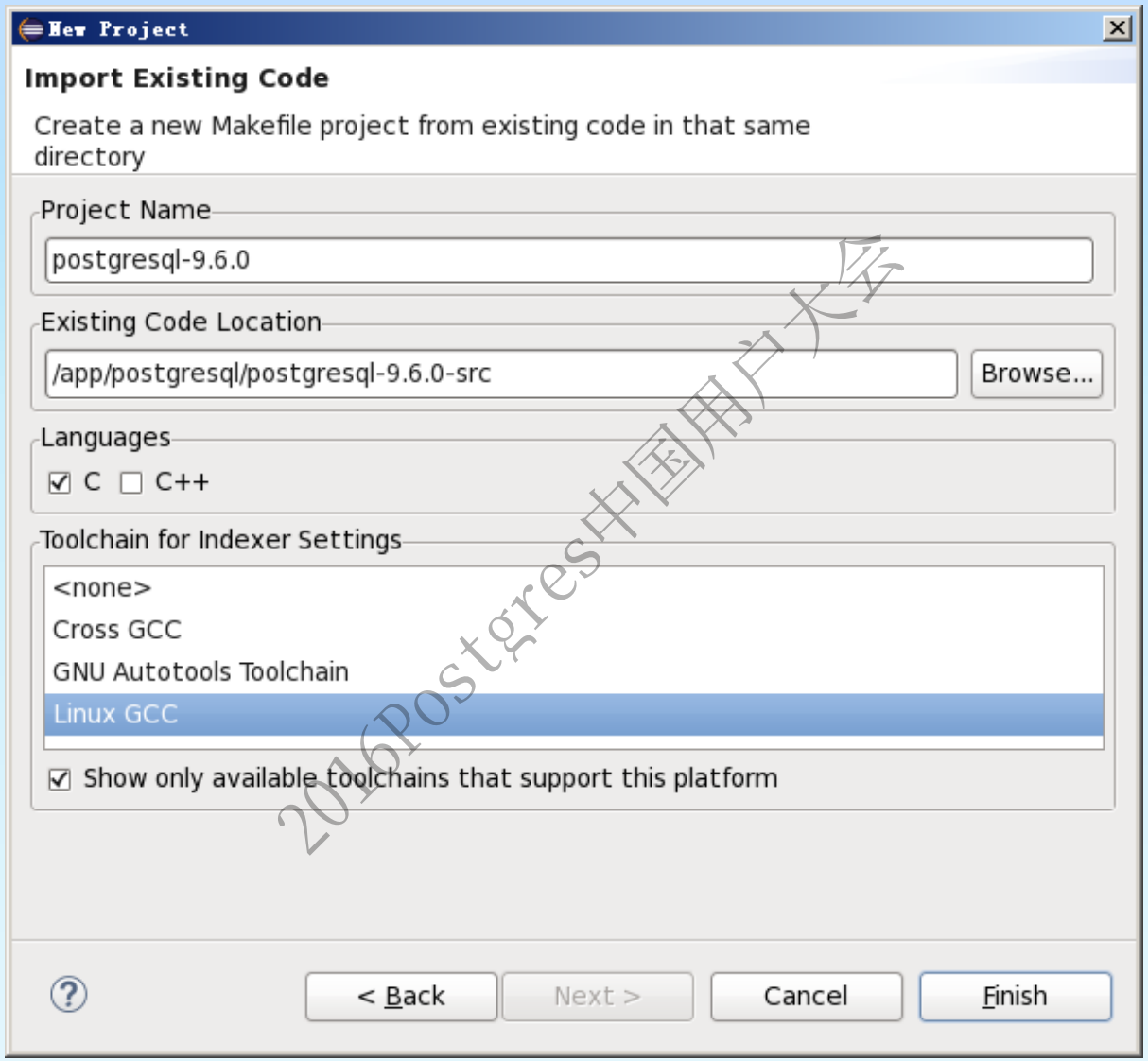
```
groupadd dba
useradd -g dba postgres –d /home/postgres
mkdir –p /app/postgresql
mkdir -p /app/postgresql/9.6.0
mkdir -p /app/postgresql/pg9600/data
mkdir -p /app/postgresql/pg9600/data/pg_log
chown -R postgres.dba /app/postgresql

--下载并解压
cd /app/postgresql
tar xzvf postgresql-9.6.0.tar.gz
mv postgresql-9.6.0 postgresql-9.6.0-src

cd /app/postgresql/postgresql-9.6.0-src
./configure --prefix=/app/postgresql/9.6.0  --enable-debug

make world
make install-world
initdb
```

Linux开发调试环境（续1）--导入工程

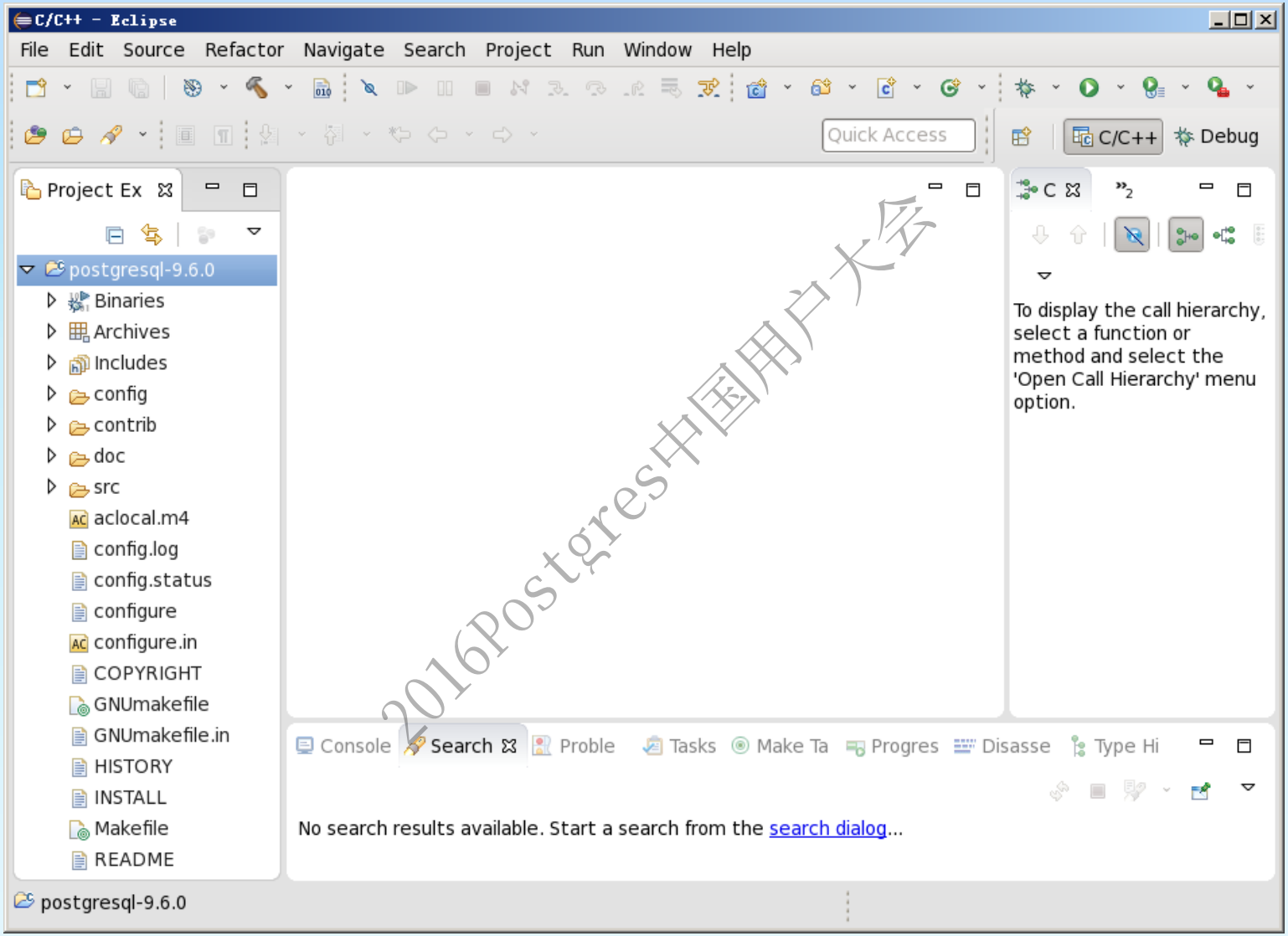




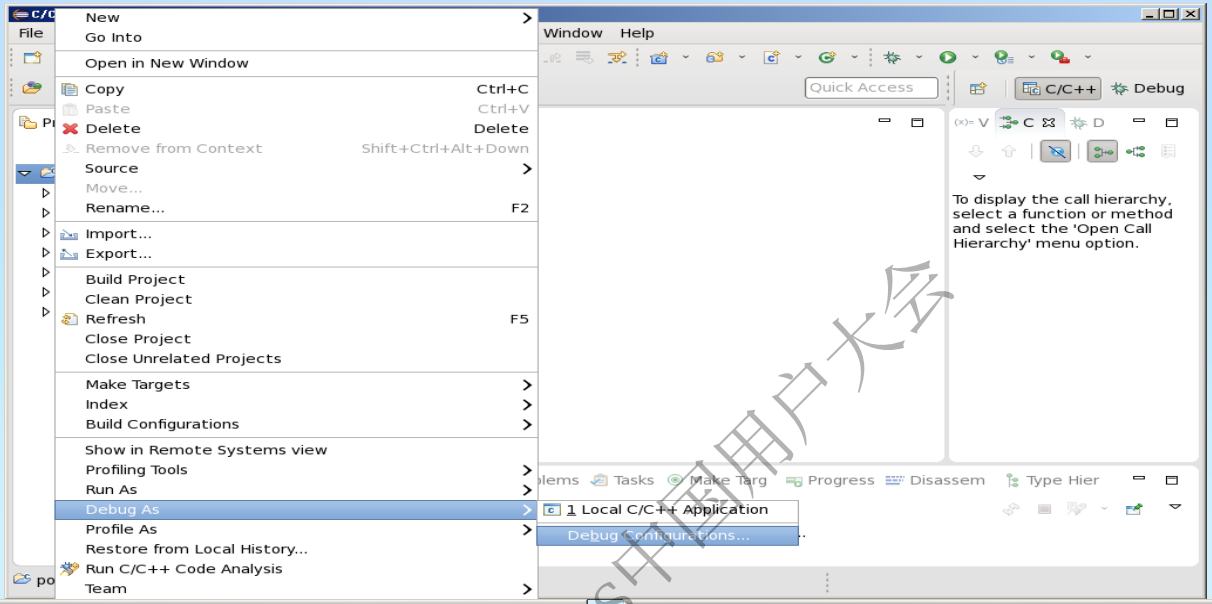
Linux开发调试环境（续1）--导入工程

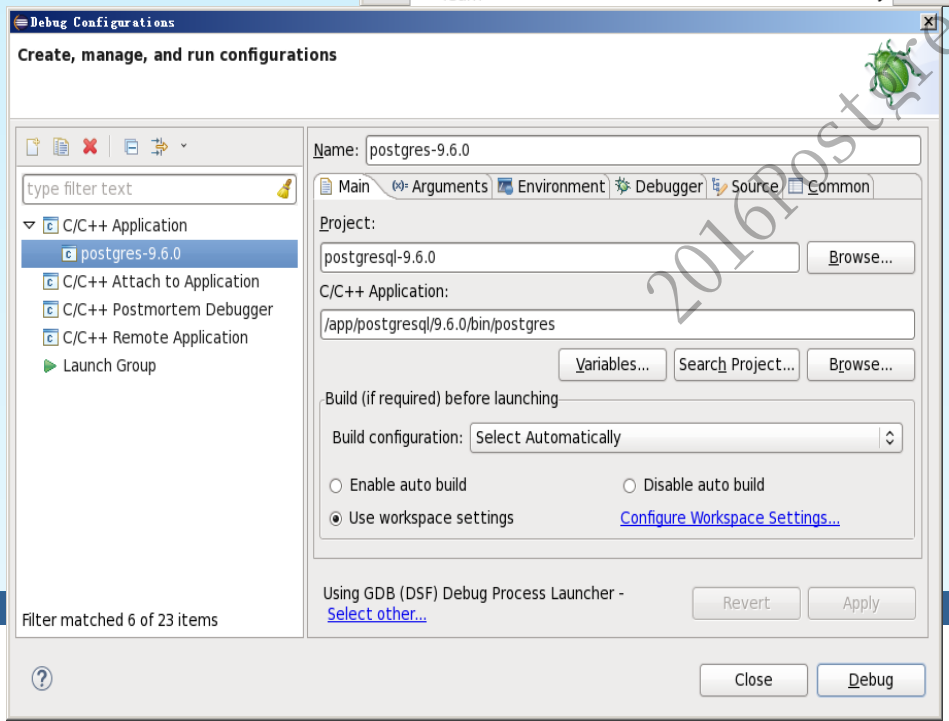


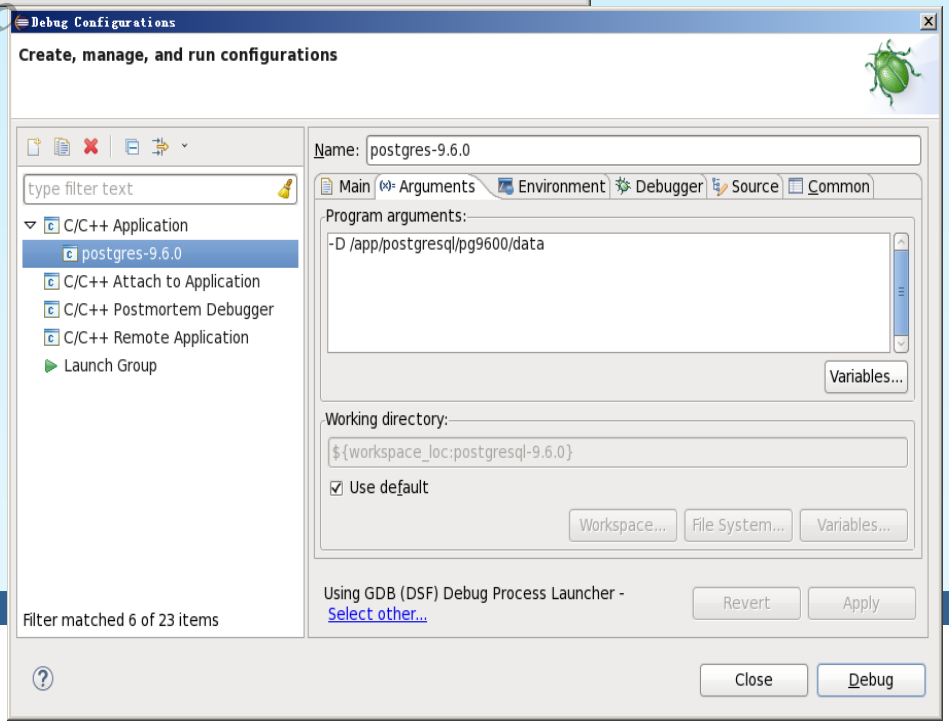
Linux开发调试环境（续1）--导入工程



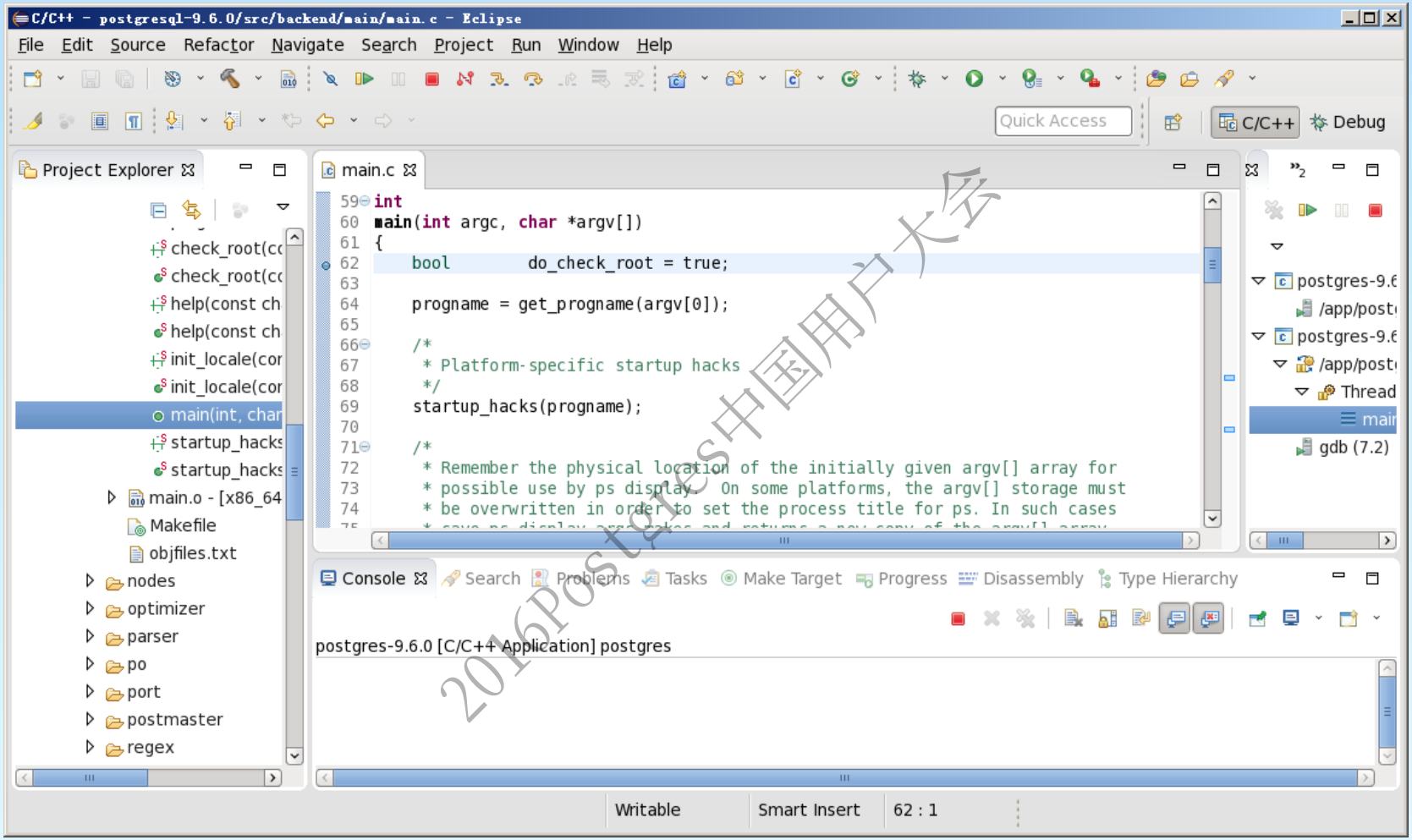
Linux开发调试环境（续2）--调试代码







Linux开发调试环境（续2）--调试代码

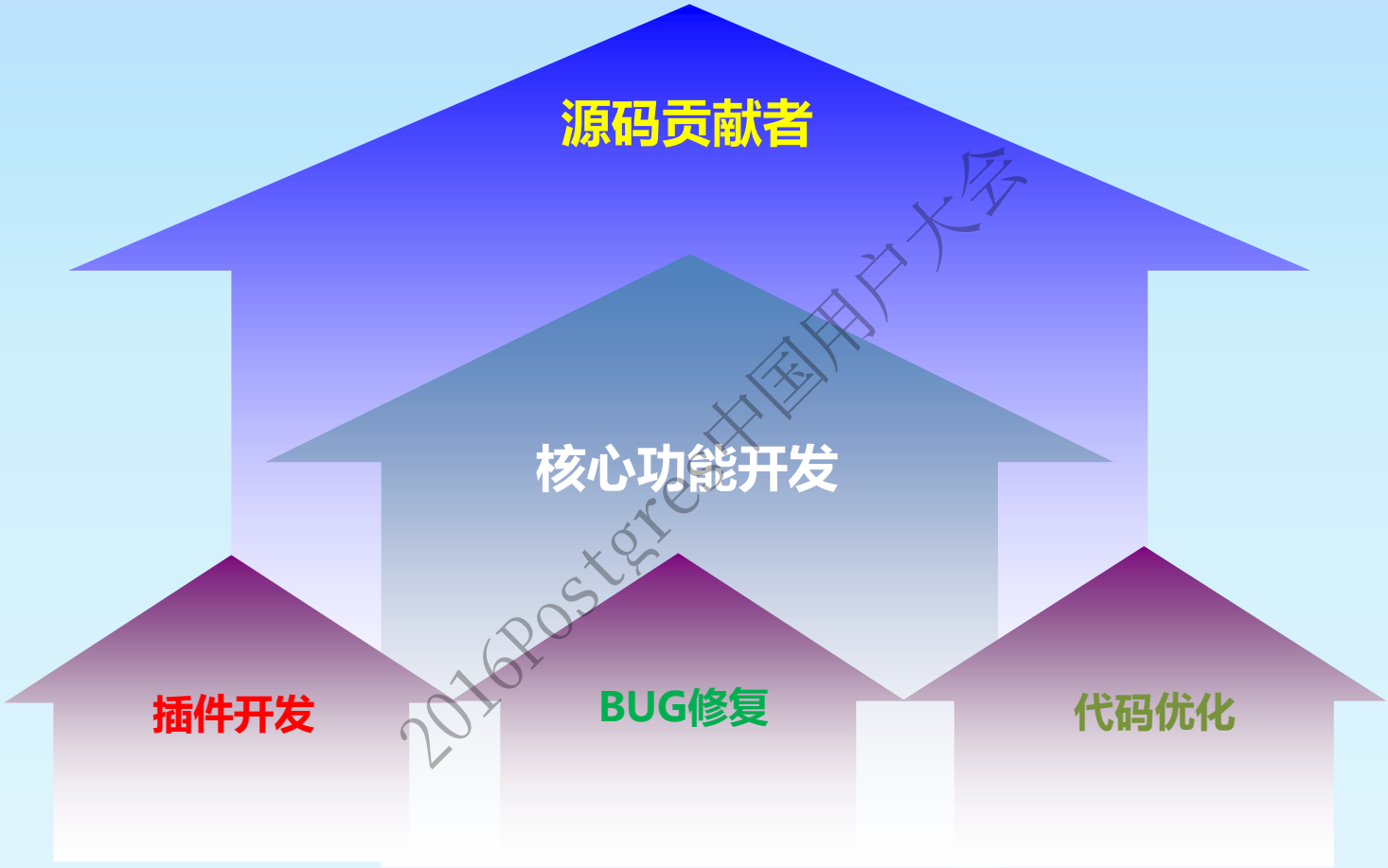


PostgreSQL源码解读心得

1. 搭建开发调试环境
2. 找到入口、单步调试
3. 先研究森林、后研究树木
4. 先泛读、后精读
5. 选择性、针对性的研究重点
6. 尽量将代码翻译成流程图或图表
7. 及时整理分析结果

分享大纲

源码开发方面我们能做点什么？



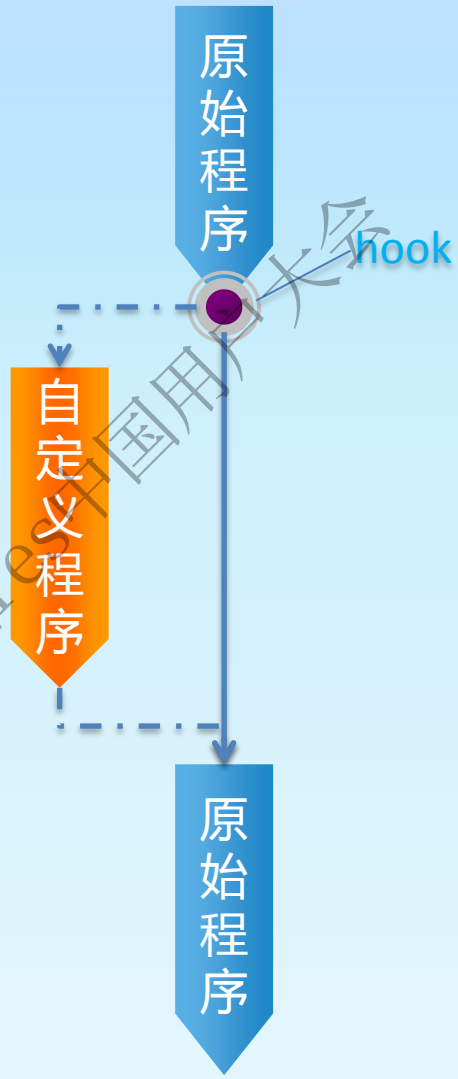
Hook技术简介

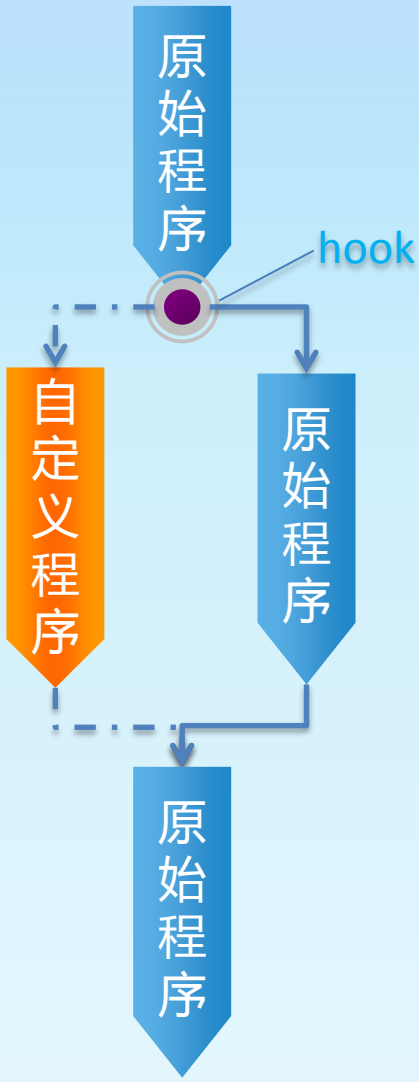
概念

hook机制，可以让用户有机会切入到PostgreSQL的内部运行机制中，进行中断、增加或修改原来的程序逻辑，从而实现一些用户自定义的功能。

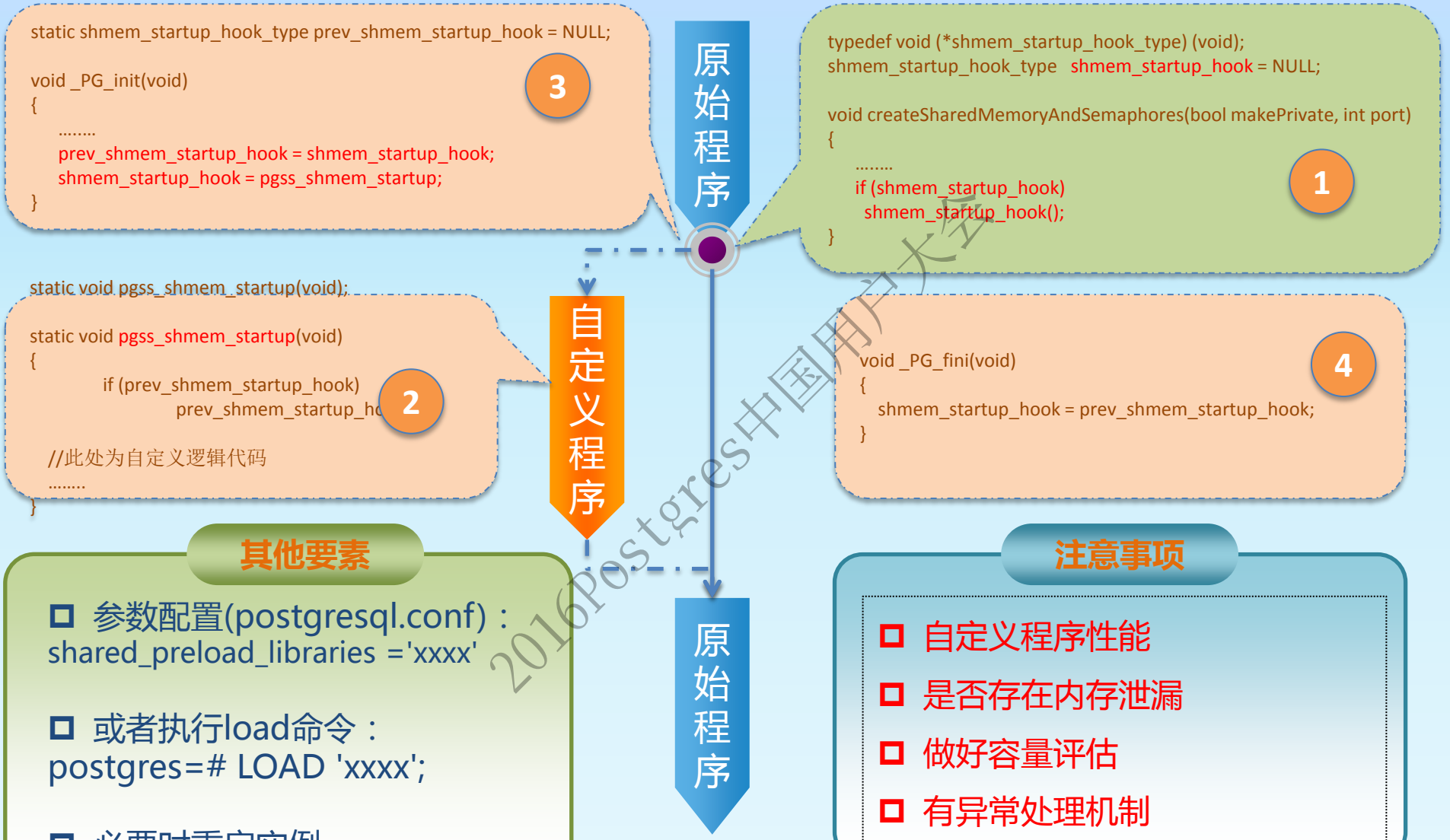
四要素

- 定义hook
- 自定义程序
- 安装hook
- 卸载hook





Hook技术简介（续1）



其他要素

- 参数配置(postgresql.conf)：
shared_preload_libraries ='xxxx'
- 或者执行load命令：
postgres=# LOAD 'xxxx';
- 必要时重启实例

注意事项

- 自定义程序性能
- 是否存在内存泄漏
- 做好容量评估
- 有异常处理机制

Hook技术简介（续2）

HOOK名	源码位置
object_access_hook	src/backend/catalog/objectaccess.c
ExplainOneQuery_hook	src/backend/commands/explain.c
explain_get_index_name_hook	src/backend/commands/explain.c
check_password_hook	src/backend/commands/user.c
ExecutorStart_hook	src/backend/executor/execMain.c
ExecutorRun_hook	src/backend/executor/execMain.c
ExecutorFinish_hook	src/backend/executor/execMain.c
ExecutorEnd_hook	src/backend/executor/execMain.c
ExecutorCheckPerms_hook	src/backend/executor/execMain.c
ClientAuthentication_hook	src/backend/libpq/auth.c
set_rel_pathlist_hook	src/backend/optimizer/path/allpaths.c
join_search_hook	src/backend/optimizer/path/allpaths.c
set_join_pathlist_hook	src/backend/optimizer/path/joinpath.c
planner_hook	src/backend/optimizer/plan/planner.c
create_upper_paths_hook	src/backend/optimizer/plan/planner.c
get_relation_info_hook	src/backend/optimizer/util/plancat.c
post_parse_analyze_hook	src/backend/parser/analyze.c
shmem_startup_hook	src/backend/storage/ipc/ipci.c
ProcessUtility_hook	src/backend/tcop/utility.c
get_relation_stats_hook	src/backend/utils/adt/selfuncs.c
get_index_stats_hook	src/backend/utils/adt/selfuncs.c
get_attavgwidth_hook	src/backend/utils/cache/lsyscache.c
emit_log_hook	src/backend/utils/error/elog.c
needs_fmgr_hook	src/backend/utils/fmgr/fmgr.c
fmgr_hook	src/backend/utils/fmgr/fmgr.c

注：基于V9.6.0版本统计

插件案例：queryid问题

PG库的pg_stat_activity视图中缺少queryid字段，由此，当pg_stat_activity与pg_stat_statements视图做关联查询时极其不方便

```
postgres=# \d pg_stat_activity
          View "pg_catalog.pg_stat_activity"
      Column      |           Type           |
------------------+--------------------------+
 datid            | oid                      |
 datname          | name                     |
 pid              | integer                  |
 usesysid         | oid                      |
 usename          | name                     |
 application_name | text                     |
 client_addr      | inet                     |
 client_hostname  | text                     |
 client_port      | integer                  |
 backend_start    | timestamp with time zone |
 xact_start       | timestamp with time zone |
 query_start      | timestamp with time zone |
 state_change     | timestamp with time zone |
 wait_event_type  | text                     |
 wait_event       | text                     |
 state            | text                     |
 backend_xid      | xid                      |
 backend_xmin     | xid                      |
 query            | text                     |
```

```
postgres=# \d pg_stat_statements;
         View "public.pg_stat_statements"
       Column        |       Type       |
---------------------+------------------+
 userid              | oid              |
 dbid                | oid              |
 queryid             | bigint           |
 query               | text             |
 calls               | bigint           |
 total_time          | double precision |
 min_time            | double precision |
 max_time            | double precision |
 mean_time           | double precision |
 stddev_time         | double precision |
 rows                | bigint           |
 shared_blks_hit     | bigint           |
 shared_blks_read    | bigint           |
 shared_blks_dirtied | bigint           |
 shared_blks_written | bigint           |
 local_blks_hit      | bigint           |
 local_blks_read     | bigint           |
 local_blks_dirtied  | bigint           |
 local_blks_written  | bigint           |
 temp_blks_read      | bigint           |
 temp_blks_written   | bigint           |
 blk_read_time       | double precision |
 blk_write_time      | double precision |
```

分析问题

- pg_stat_activity视图
  - 关联pg_database和pg_authid两张表和pg_stat_get_activity函数
  - pg_stat_get_activity函数最终调用C程序，用于获取内存中所有连接会话的信息
  - 内存中的连接会话信息中并无queryid信息

(源码位置: src/backend/utils/adt/pgstatfuncs.c)

```
CREATE VIEW pg_stat_activity AS
    SELECT
            S.datid AS datid,
            D.datname AS datname,
            S.pid,
            S.usesysid,
            U.rolname AS usename,
            S.application_name,
            S.client_addr,
            S.client_hostname,
            S.client_port,
            S.backend_start,
            S.xact_start,
            S.query_start,
            S.state_change,
            S.wait_event_type,
            S.wait_event,
            S.state,
            S.backend_xid,
            s.backend_xmin,
            S.query
    FROM pg_database D, pg_stat_get_activity(NULL) AS S, pg_authid U
    WHERE S.datid = D.oid AND
            S.usesysid = U.oid;
```

- 核心源码中有定义queryid变量
  - 但核心代码中没有任何地方对其赋值
  - 可以被第三方插件赋值

(源码位置:src/include/nodes/parsenodes.h)

```
typedef struct Query
{
    NodeTag     type;

    CmdType     commandType;    /* select|insert|update|delete|utility */

    QuerySource querySource;    /* where did I come from? */

    uint32      queryId;        /* query identifier (can be set by plugins) */

    bool        canSetTag;      /* do I set the command result tag? */

    Node       *utilityStmt;    /* non-null if this is DECLARE CURSOR or a
                                 * non-optimizable statement */

    int         resultRelation; /* rtable index of target relation for
                                 * INSERT/UPDATE/DELETE; 0 for SELECT */
```

分析问题(续)

- pg_stat_statements插件
  - 用来记录SQL的性能数据
  - 在SQL解析之后，生成并保存queryid信息

(源码位置：

contrib/pg_stat_statements/pg_stat_statements.c

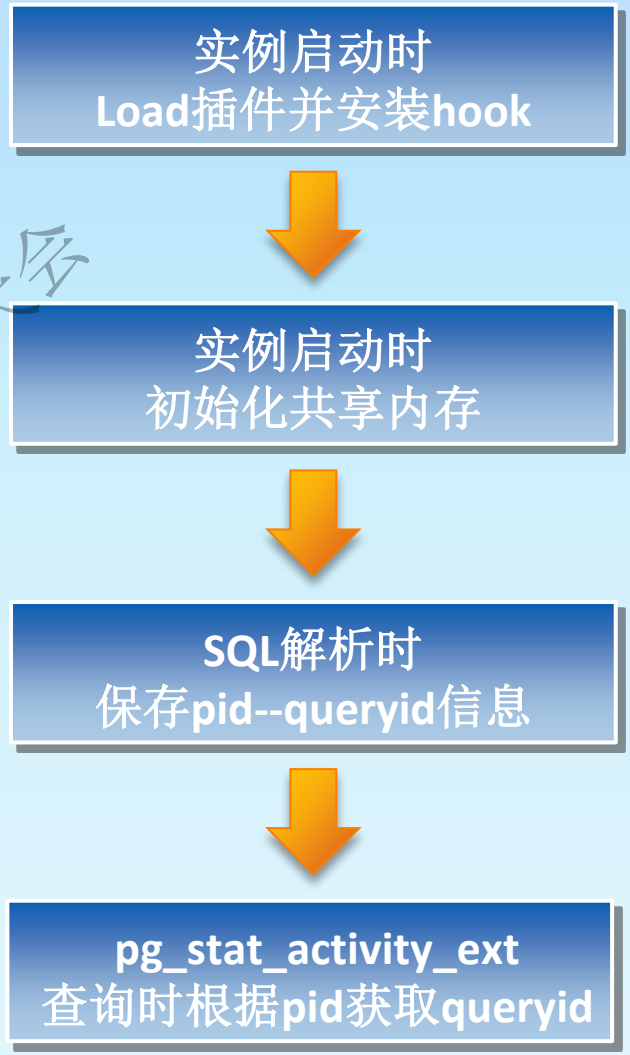
src/backend/tcop/postgres.c)

```
List *
pg_analyze_and_rewrite_params(Node *parsetree,
                              const char *query_string,
                              ParserSetupHook parserSetup,
                              void *parserSetupArg)
{
    ParseState *pstate;
    Query      *query;
    List       *querytree_list;

    Assert(query_string != NULL);       /* required as of 8.4 */

    TRACE_POSTGRESQL_QUERY_REWRITE_START(query_string);

    /*
     * (1) Perform parse analysis.
     */
    if (log_parser_stats)
        ResetUsage();

    pstate = make_parsestate(NULL);
    pstate->p_sourcetext = query_string;
    (*parserSetup) (pstate, parserSetupArg);

    query = transformTopLevelStmt(pstate, parsetree);

    if (post_parse_analyze_hook)
        (*post_parse_analyze_hook) (pstate, query);

    free_parsestate(pstate);
```

```
static void
pgss_post_parse_analyze(ParseState *pstate, Query *query)
{
    pgssJumbleState jstate;

    if (prev_post_parse_analyze_hook)
        prev_post_parse_analyze_hook(pstate, query);

    /* Assert we didn't do this already */
    Assert(query->queryId == 0);

    /* Safety check... */
    if (!pgss || !pgss_hash)
        return;

    /*
     * Utility statements get queryId zero.  We do this even in cases where
     * the statement contains an optimizable statement for which a queryId
     * could be derived (such as EXPLAIN or DECLARE CURSOR).  For such cases,
     * runtime control will first go through ProcessUtility and then the
     * executor, and we don't want the executor hooks to do anything, since we
     * are already measuring the statement's costs at the utility level.
     */
    if (query->utilityStmt)
    {
        query->queryId = 0;
        return;
    }

    /* Set up workspace for query jumbling */
    jstate.jumble = (unsigned char *) palloc(JUMBLE_SIZE);
    jstate.jumble_len = 0;
    jstate.clocations_buf_size = 32;
    jstate.clocations = (pgssLocationLen *)
        palloc(jstate.clocations_buf_size * sizeof(pgssLocationLen));
    jstate.clocations_count = 0;

    /* Compute query ID and mark the Query node with it */
    JumbleQuery(&jstate, query);
    query->queryId = hash_any(jstate.jumble, jstate.jumble_len);
```

PG_STAT_ACTIVITY_EXT插件--实现思路

- PostgreSQL数据库是一个多进程系统，每一个服务程序对应一个操作系统进程
- 可以将PostgreSQL服务进程的pid与该进程当前执行SQL的queryid作为key-value键值对保存
- 该键值对保存在共享内存的hashtable数据结构中
- 基于pg_stat_activity视图，新建一个新的视图pg_stat_activity_ext，增加字段queryid，该字段来源于函数GetQueryId()
- 函数GetQueryId的作用是根据pid获取queryid



PG_STAT_ACTIVITY_EXT插件--代码片段

1 实例启动时，加载插件，并安装hook

```
static shmem_startup_hook_type prev_shmem_startup_hook = NULL;
static post_parse_analyze_hook_type prev_post_parse_analyze_hook = NULL;

void _PG_init(void) {
      if (!process_shared_preload_libraries_in_progress)
            return;

      prev_shmem_startup_hook = shmem_startup_hook;
      shmem_startup_hook = pgsae_shmem_startup;

      prev_post_parse_analyze_hook = post_parse_analyze_hook;
      post_parse_analyze_hook = pgsae_post_parse_analyze;
}
```

```
typedef void (*shmem_startup_hook_type) (void);
shmem_startup_hook_type shmem_startup_hook = NULL;

void
CreateSharedMemoryAndSemaphores(bool makePrivate,
int port)
{
……
      if (shmem_startup_hook)
            shmem_startup_hook();
}

(源码位置：src/backend/storage/ipc/ipci.c)
```

```
typedef void (*post_parse_analyze_hook_type) (ParseState
*pstate,Query *query);
post_parse_analyze_hook_type post_parse_analyze_hook = NULL;

Query *
parse_analyze(Node *parseTree, const char *sourceText,
                    Oid *paramTypes, int numParams)
{
……
      if (post_parse_analyze_hook)
            (*post_parse_analyze_hook) (pstate, query);
……
      return query;
}
(源码位置：src/backend/parser/analyze.c)
```

PG_STAT_ACTIVITY_EXT插件--代码片段(续1)

2 实例启动时，初始化共享内存

```
static HTAB *pgsae_hash = NULL;

static void
pgsae_shmem_startup(void) {
	HASHCTL info;

	if (prev_shmem_startup_hook)
		prev_shmem_startup_hook();
	memset(&info, 0, sizeof(info));
	info.keysize = sizeof(pid_t);
	info.entrysize = sizeof(pgsaeEntry);
	info.match = pgsae_match_fn;
	pgsae_hash = ShmemInitHash("pg_stat_activity_ext
hash", 500, 10000, &info,HASH_ELEM | HASH_COMPARE);
}
```

```
static int
pgsae_match_fn(const void *key1, const void *key2, Size
keysize) {
	const pid_t *k1 = (const pid_t *) key1;
	const pid_t *k2 = (const pid_t *) key2;

	if (*k1 == *k2)
		return 0;
	else
		return 1;
}
```

3 SQL语句parse后保存pid-queryid信息到共享内存的HASHTABLE中

```
static void pgsae_post_parse_analyze(ParseState *pstate, Query *query) {
	bool found;
	pid_t key;
	pgsaeEntry *entry;

	if (prev_post_parse_analyze_hook)
		prev_post_parse_analyze_hook(pstate, query);

	key = getpid();
	entry = (pgsaeEntry *) hash_search(pgsae_hash, &key, HASH_ENTER, &found);
	entry->pid = key;
	entry->queryId = query->queryId;
}
```

PG_STAT_ACTIVITY_EXT插件--代码片段(续2)

4 新增一个Function（C函数），用来根据pid从共享内存HASHTABLE中获取queryid

```
Datum GetQueryId(PG_FUNCTION_ARGS) {
      pid_t key = (pid_t) PG_GETARG_UINT32(0);
      bool found;
      pgsaeEntry *entry;

      if (!pgsae_hash)
            ereport(ERROR,
                        (errcode(ERRCODE_OBJECT_NOT_IN_PREREQUISITE_STATE), errmsg("pg_stat_activity_ext must be
loaded via 'load command' or shared_preload_libraries")));

      entry = (pgsaeEntry *) hash_search(pgsae_hash, &key, HASH_FIND, &found);
      if (found) {
            return entry->queryId;
      } else {
            return 0;
      }
}
```

5 视图PG_STAT_ACTIVITY_EXT和GetQueryId Function的脚本

```
CREATE FUNCTION pg_catalog.GetQueryId(bigint)
RETURNS bigint
AS 'MODULE_PATHNAME' ,'GetQueryId'
LANGUAGE C VOLATILE;
```

```
CREATE VIEW pg_stat_activity_ext AS
SELECT a.*,GetQueryId(a.pid) as queryid
FROM pg_stat_activity a;
```

PG_STAT_ACTIVITY_EXT插件--相关文件说明

文件名	文件说明	备注
pg_stat_activity_ext.c	C源码文件	
pg_stat_activity_ext--1.0.sql	SQL脚本文件	用来创建PG_STAT_ACTIVITY_EXT视图和GetQueryId函数
pg_stat_activity_ext.control	控制文件	
Makefile	编译安装所需文件	

注：将以上文件放置于contrib/pg_stat_activity_ext 目录下

PG_STAT_ACTIVITY_EXT插件--安装部署

1 编译安装

```
cd contrib/pg_stat_activity_ext
make && make install
```

2 配置参数(postgresql.conf)

```
shared_preload_libraries = 'pg_stat_statements,pg_stat_activity_ext'    ##注意先后顺序
```

3 创建对象

```
postgres=# create extension pg_stat_activity_ext;
```

4 重启数据库实例

```
pg_ctl restart
```

5 验证

```
postgres=# \dx
                                    List of installed extensions
         Name          | Version   |   Schema   |                          Description
-----------------------+-----------+------------+--------------------------------------------------------------
 pg_stat_activity_ext | 1.0       | public     | extension of pg_stat_activity view
 pg_stat_statements    | 1.4       | public     | track execution statistics of all SQL statements executed
```

PG_STAT_ACTIVITY_EXT插件--最终效果

```
postgres=# select pid,usename,state,queryid,query from pg_stat_activity_ext;
  pid  | usename  | state  |  queryid   |                                  query
-------+----------+--------+------------+-----------------------------------------------------------------
 20981 | postgres | active | 1517127817 | select pid,usename,state,queryid,query from pg_stat_activity_ext;
 20857 | postgres | idle   | 2044555847 | select now();
 20899 | postgres | idle   | 2236211933 | select count(*) from pg_class;
 20939 | postgres | idle   | 3589441560 | select 1;
```

```
postgres=# select sae.usename,ss.calls,ss.total_time,ss.min_time,
ss.max_time,sae.queryid,sae.query
from pg_stat_activity_ext sae,pg_stat_statements ss
where sae.datid=ss.dbid and sae.usesysid=ss.userid
and sae.queryid=ss.queryid;
 usename  | calls | total_time | min_time | max_time |  queryid   |                          query
----------+-------+------------+----------+----------+------------+-----------------------------------------------------------
 postgres |     4 |      1.542 |    0.324 |    0.523 | 2223967786 | select sae.usename,ss.calls,ss.total_time,ss.min_time,+
          |       |            |          |          |            | ss.max_time,sae.queryid,sae.query                         +
          |       |            |          |          |            | from pg_stat_activity_ext sae,pg_stat_statements ss       +
          |       |            |          |          |            | where sae.datid=ss.dbid and sae.usesysid=ss.userid        +
          |       |            |          |          |            | and sae.queryid=ss.queryid;
 postgres |     1 |      0.032 |    0.032 |    0.032 | 3589441560 | select 1;
 postgres |     1 |      0.056 |    0.056 |    0.056 | 2044555847 | select now();
 postgres |     1 |      0.108 |    0.108 |    0.108 | 2236211933 | select count(*) from pg_class;
```

```
postgres=# select Getqueryid(20899);
 getqueryid
------------
 2236211933
(1 row)
```

总结

倡导

希望各位PG爱好者：

从搭建开发调试环境开始入门，

尽快找到main函数，

单步debug逐步深入内核，

尽早加入到源码研究的队伍中来，

早日成为PG数据库源码的贡献者！

Thanks!

Q & A