

Программирование на C для PostgreSQL

Alexander Korotkov

Postgres Professional

LVEE, 2015

- ▶ Locale support
- ▶ Extendability (indexing)
 - ▶ GiST (KNN), GIN, SP-GiST
- ▶ Full Text Search (FTS)
- ▶ Jsonb, VODKA
- ▶ Extensions:
 - ▶ intarray
 - ▶ pg_trgm
 - ▶ ltree
 - ▶ hstore
 - ▶ plantuner



<https://www.facebook.com/oleg.bartunov>
obartunov@postgrespro.ru,
teodor@postgrespro.ru
<https://www.facebook.com/groups/postgresql/>

- ▶ Indexed regexp search
- ▶ GIN compression & fast scan
- ▶ Fast GiST build
- ▶ Range types indexing
- ▶ Split for GiST
- ▶ Indexing for jsonb
- ▶ jsquery
- ▶ Generic WAL + create am (WIP) a.korotkov@postgrespro.ru



```
#include "postgres.h"
#include "fmgr.h"

PG_MODULE_MAGIC;
PG_FUNCTION_INFO_V1(add);

Datum
add(PG_FUNCTION_ARGS)
{
    int32 arg1 = PG_GETARG_INT32(0),
          arg2 = PG_GETARG_INT32(1);
    PG_RETURN_INT32(arg1 + arg2);
}
```

- ▶ Datum – это unsigned integer достаточной длины, чтобы хранить в себе pointer.

```
typedef uintptr_t Datum;
```

- ▶ Любое значение PostgreSQL может быть приведено к типу Datum и наоборот. Для этого используются макросы DatumGet*(x) и *GetDatum(x).
- ▶ Значения, которые помещаются в Datum, могут быть переданы по значению, остальные передаются по указателю.
- ▶ Аргументы функции и возвращаемые значения передаются как Datum.

- ▶ Параметры передаются через специальную структуру

```
/* Standard parameter list for fmgr-compatible functions */  
#define PG_FUNCTION_ARGS FunctionCallInfo fcinfo  
typedef struct FunctionCallInfoData *FunctionCallInfo;
```

- ▶ Для доступа к параметрам заведены макросы PG_GET_ARG_*(n)

```
#define PG_GETARG_DATUM(n) (fcinfo->arg[n])  
#define PG_GETARG_INT32(n) DatumGetInt32(PG_GETARG_DATUM(n))
```

- ▶ Для возвращения результата используются макросы
PG_RETURN_*(x)

```
#define PG_RETURN_DATUM(x) return (x)  
#define PG_RETURN_INT32(x) return Int32GetDatum(x)
```

- ▶ PG_FUNCTION_INFO_V1(funcname) указывает на использование calling convention version 1, был ещё version 0...

Введён начиная с версии 8.2 и позволяет автоматически проверять совместимость модуля со сборкой PostgreSQL.

```
/* The actual data block contents */
#define PG_MODULE_MAGIC_DATA \
{ \
    sizeof(Pg_magic_struct), \
    PG_VERSION_NUM / 100, \
    FUNC_MAX_ARGS, \
    INDEX_MAX_KEYS, \
    NAMEDATALEN, \
    FLOAT4PASSBYVAL, \
    FLOAT8PASSBYVAL \
}
```

▲▲

Из чего состоит расширение pg_aa?

- ▶ `Makefile` – файл для сборки с помощью GNU make
- ▶ `pg_aa-1.0.sql` – SQL-скрипт, создающий объекты БД
- ▶ `pg_aa.c` – исходный текст на C
- ▶ `pg_aa.control` – control file с параметрами
- ▶ `sql/pg_aa.sql` – регрессионные тесты
- ▶ `data/pg_aa.sql` – данные для регрессионных тестов
- ▶ `expected/pg_aa.out` – ожидаемые результаты

```
# pg_aa/Makefile

MODULE_big = pg_aa      # Название собираемого .so файла
OBJS = pg_aa.o          # Список .о файлов, из которых собирается .so
EXTENSION = pg_aa       # Название самого extension'а
DATA = pg_aa--1.0.sql   # Файлы, которые нужно скопировать в
SHLIB_LINK = -lgd -laa -lcasa # Библиотеки, которые нужно подключить
REGRESS = pg_aa         # Список тестов

ifdef USE_PGXS # Если USE_PGXS установлено, то нужные для сборки файлы
                # PostgreSQL находятся с помощью утилиты pg_config
PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
else          # Если нет, то считается что расширение помещено в папку
              # contrib исходников PostgreSQL
subdir = contrib/pg_aa
top_builddir = ../..
include $(top_builddir)/src/Makefile.global
include $(top_srcdir)/contrib/contrib-global.mk
endif
```

Этот файл является входной точкой для команд (CREATE|ALTER|DROP) EXTENSION.

```
# pg_aa extension
comment = 'ASCII art extension for PostgreSQL'
default_version = '1.0'           # Версия SQL-скрипта, которая
                                  # будет устанавливаться по-умолчанию
module_pathname = '$libdir/pg_aa' # Значение, которое будет
                                  # подставляться в MODULE_PATHNAME
                                  # SQL-скрипта
relocatable = true               # Можно ли перемещать extension в
                                  # другую схему
```

```
/* pg_aa/pg_aa--1.0.sql */

-- complain if script is sourced in psql, rather than via CREATE
EXTENSION
\echo Use "CREATE EXTENSION pg_aa" to load this file. \quit

--
-- PostgreSQL code for pg_aa.
--

CREATE FUNCTION aa_out(bytea, int4) -- Функция вывода с помощью libaa
RETURNS text
AS 'MODULE_PATHNAME'
LANGUAGE C IMMUTABLE STRICT;

CREATE FUNCTION caca_out(bytea, int4) -- Функция вывода с помощью libcaca
RETURNS text
AS 'MODULE_PATHNAME'
LANGUAGE C IMMUTABLE STRICT;
```

Связь между extension'ом и его объектами хранится в pg_depend.

Благодаря этому:

- ▶ DROP EXTENSION удаляет весь extension и ему не нужен для этого отдельный скрипт.
- ▶ Нельзя случайно удалить часть extension'а.
- ▶ В SQL dump не попадают объекты extension'а, а только команда CREATE EXTENSION.

```
# SELECT p.proname
FROM pg_depend d
      JOIN pg_extension e ON e.oid = d.refobjid
      JOIN pg_proc p ON p.oid = d.objid
WHERE e.extname = 'pg_aa';
prname
-----
aa_out
caca_out
(2 rows)
```

```
Datum
aa_out(PG_FUNCTION_ARGS)
{
    bytea *img = PG_GETARG_BYTEA_PP(0); /* Получаем значения аргументов */
    int     width = PG_GETARG_INT32(1);
    ...
    old_locale = pstrdup(setlocale(LC_CTYPE, NULL));
    setlocale(LC_CTYPE, "C"); /* libaa нормально работает только в
                               * локали C */
    im = gdImageCreateFromPngPtr(VARSIZE_ANY_EXHDR(img),
                                VARDATA_ANY(img));
    ...
    context = aa_init(&mem_d, &params, NULL);
    ...
    for (i = 0; i < gdImageSX(tb); i++) /* Отрисовываем */
    {
        for (j = 0; j < gdImageSY(tb); j++)
            aa_putpixel(context, i, j, get_intensity(tb, i, j));
    }
    ...
}
```

```
...
/* Получаем вывод и разбавляем его переводами строк */
s = (char *)aa_text(context);
result = (text *)palloc(VARHDRSZ +
    (width + 1) * height * MAX_MULTIBYTE_CHAR_LEN);
SET_VARSIZE(result, d - (char *)result);
d = VARDATA(result);
for (i = 0; i < height; i++)
{
    if (i > 0)
        *d++ = '\n';
    for (j = 0; j < width; j++)
        *d++ = *s++;
}
...
setlocale(LC_CTYPE, old_locale); /* Возвращаем локаль назад */
pfree(old_locale);
PG_RETURN_TEXT_P(result);
}
```

- ▶ Значения с не фиксированной длиной хранятся как заголовок + данные.
- ▶ Заголовок занимает 4 байта (VARHDRSZ), хранит флаги и длину. Начиная с 8.3 для коротких значений есть 1-байтовая версия заголовка.
- ▶ В заголовке также содержится признак сжатого и TOAST значений.
- ▶ `VARSIZE*(x)` – длина, `VARDATA*(x)` – получение указателя на данные, `SET_VARSIZE*(x)` – установить длину.


```
psql test
# CREATE EXTENSION pg_aa;
CREATE EXTENSION
# CREATE TABLE logo (name text NOT NULL, data bytea NOT NULL);
CREATE TABLE
# \q

echo -e 'PostgreSQL\t\\\\\\x' 'xxd -p pg_logo.png | tr -d '\r\n' ' |
psql test -c 'COPY logo FROM stdin;''

psql test
# SELECT aa_out(data, 10) FROM logo;
aa_out
-----
 _mixWQIXm,+
)WQenWx3i(+
)%QoxWwnr +
]"N?%W7"'+
  WiQ'
(1 row)
```

Как такое возможно?

~/ .psqlrc

```
-- Pager всегда включен
\pset pager always
-- Убираем экранирование с символа escape
\setenv PAGER 'sed "s/\\\\\\\\\\\\\\\\x1B/'echo "\033"/g" | less'
-- Настройки для less: отображение длинных линий, цветов т.д.
\setenv LESS '-iMSx4R -FX'
```

sql/pg_aa.sql

```
CREATE EXTENSION pg_aa;  
  
CREATE TABLE logo (name text NOT NULL, data bytea NOT NULL);  
\copy logo FROM 'data/logo.data'  
  
SELECT aa_out(data, 20) FROM logo ORDER BY name;  
SELECT aa_out(data, 40) FROM logo ORDER BY name;  
  
SELECT caca_out(data, 20) FROM logo ORDER BY name;  
SELECT caca_out(data, 40) FROM logo ORDER BY name;
```

expected/pg_aa.out

```
CREATE EXTENSION pg_aa;
CREATE TABLE logo (name text NOT NULL, data bytea NOT NULL);
\copy logo FROM 'data/logo.data'
SELECT aa_out(data, 20) FROM logo ORDER BY name;
      aa_out
-----
      qyggwp+      +
      jwQQU$Qr      +
      ]p%Y]nQk      +
      .L,.uJQQ,      +
      qZ'W'= )QQc      +
      j$'           ?m$g, +
      qh'           $SQ[ +
      -.T1p         ,dmmt +
      ](' )$c       yf]!')_,+
      ]'           /\ggaaQf ..'+
      W^-^y../""""""s...-+
...

```

```
typedef struct Complex
{
    double      x;
    double      y;
} Complex;
```

```
CREATE TYPE complex (
    internallength = 16, -- тип фиксированной длины
    input = complex_in, -- преобразование в текст
    output = complex_out, -- преобразование из текста
    receive = complex_recv, -- пересылка по сети
    send = complex_send, -- получение по сети
    alignment = double -- выравнивание
);
```

Преобразование cstring => complex. Если что-то пошло не так, то кидает ereport с уровнем >= ERROR.

```
Datum
complex_in(PG_FUNCTION_ARGS)
{
    char        *str = PG_GETARG_CSTRING(0);
    double      x,
               y;
    Complex     *result;

    if (sscanf(str, " ( %lf , %lf )", &x, &y) != 2)
        ereport(ERROR,
                (errcode(ERRCODE_INVALID_TEXT_REPRESENTATION),
                 errmsg("invalid input syntax for complex: \"%s\"",
                        str)));

    result = (Complex *) palloc(sizeof(Complex));
    result->x = x;
    result->y = y;
    PG_RETURN_POINTER(result);
}
```

Преобразование complex => cstring.

```
Datum
complex_out(PG_FUNCTION_ARGS)
{
    Complex    *complex = (Complex *) PG_GETARG_POINTER(0);
    char       *result;

    result = psprintf("(%g,%g)", complex->x, complex->y);
    PG_RETURN_CSTRING(result);
}
```


Специальные макросы `pq_send*(buf, value)` для кроссплатформенной отправки значений.

```
Datum
complex_send(PG_FUNCTION_ARGS)
{
    Complex    *complex = (Complex *) PG_GETARG_POINTER(0);
    StringInfoData buf;

    pq_begintypsend(&buf);
    pq_sendfloat8(&buf, complex->x);
    pq_sendfloat8(&buf, complex->y);
    PG_RETURN_BYTEA_P(pq_endtypsend(&buf));
}
```

Специальные макросы `pq_getmsg*(buf)` для кроссплатформенного получения значений.

```
Datum
complex_recv(PG_FUNCTION_ARGS)
{
    StringInfo  buf = (StringInfo) PG_GETARG_POINTER(0);
    Complex     *result;

    result = (Complex *) palloc(sizeof(Complex));
    result->x = pq_getmsgfloat8(buf);
    result->y = pq_getmsgfloat8(buf);
    PG_RETURN_POINTER(result);
}
```

```
Datum -- Код оператора на C
complex_add(PG_FUNCTION_ARGS)
{
    Complex    *a = (Complex *) PG_GETARG_POINTER(0);
    Complex    *b = (Complex *) PG_GETARG_POINTER(1), *result;
    result = (Complex *) palloc(sizeof(Complex));
    result->x = a->x + b->x;
    result->y = a->y + b->y;
    PG_RETURN_POINTER(result);
}
```

```
CREATE FUNCTION complex_add(complex, complex)
    RETURNS complex
    AS 'MODULE_PATHNAME'
    LANGUAGE C IMMUTABLE STRICT;

CREATE OPERATOR + ( -- Определение оператора
    leftarg = complex, rightarg = complex, -- Типы аргументов
    procedure = complex_add, -- Функция, которая выполняет оператор
    commutator = + -- Коммутатор есть( ещё negator)
);
```

```
Datum -- Код оператора на C
complex_mult(PG_FUNCTION_ARGS)
{
    Complex    *a = (Complex *) PG_GETARG_POINTER(0);
    Complex    *b = (Complex *) PG_GETARG_POINTER(1);
    Complex    *result;
    result = (Complex *) palloc(sizeof(Complex));
    result->x = a->x * b->x - a->y * b->y;
    result->y = a->x * b->y + a->y * b->x;
    PG_RETURN_POINTER(result);
}
```

```
CREATE FUNCTION complex_mult(complex, complex)
    RETURNS complex
    AS 'MODULE_PATHNAME'
    LANGUAGE C IMMUTABLE STRICT;

CREATE OPERATOR * ( -- Определение оператора
    leftarg = complex, rightarg = complex, -- Типы аргументов
    procedure = complex_mult, -- Функция, которая выполняет оператор
    commutator = * -- Коммутатор есть( ещё negator)
);
```

Теперь мы можем преобразовывать комплексные числа из строки и в строку, а также использовать операторы + и *.

```
# SELECT '(1, 1)::complex + '(-1,1)::complex;
?column?
-----
(0,2)
(1 row)

# SELECT '(1, 1)::complex * '(-1,1)::complex;
?column?
-----
(-2,0)
(1 row)
```

Сравнение по комплексного числа по модулю.

```
Datum
complex_abs_lt(PG_FUNCTION_ARGS)
{
    Complex    *a = (Complex *) PG_GETARG_POINTER(0);
    Complex    *b = (Complex *) PG_GETARG_POINTER(1);
    PG_RETURN_BOOL(complex_abs_cmp_internal(a, b) < 0);
}
```

```
CREATE FUNCTION complex_abs_lt(complex, complex) RETURNS bool
AS 'MODULE_PATHNAME' LANGUAGE C IMMUTABLE STRICT;

CREATE OPERATOR < (
    leftarg = complex, rightarg = complex, procedure = complex_abs_lt,
    commutator = > , negator = >= ,
    restrict = scalarltsel, join = scalarltjoinsel
);
```

Аналогичным образом вводятся <, <=, =, >=, >.

Operator class (opclass)

- ▶ Opclass – мостик между типом индекса (access method) и типом данных.
- ▶ Любой индекс строится с использованием opclass'а.
- ▶ Если конкретный opclass не задан, то используется opclass по-умолчанию.
- ▶ Кроме этого btree и hash opclass'ы используются при исполнении запросов для сортировки и хэширования.

```
CREATE OPERATOR CLASS complex_abs_ops
  DEFAULT FOR TYPE complex USING btree AS
  -- Операторы, которые поддерживает opclass
  OPERATOR          1          < ,
  OPERATOR          2          <= ,
  OPERATOR          3          = ,
  OPERATOR          4          >= ,
  OPERATOR          5          > ,
  -- Функция сравнения
  FUNCTION           1          complex_abs_cmp(complex, complex);
```

```
# CREATE INDEX complex_test_v_idx ON complex_test(v complex_abs_ops);
# SELECT * FROM complex_test WHERE v > '(0.5, 0.5)' ORDER BY v LIMIT 5;
      v
-----
(0.344993,0.617238)
(0.0314643,0.706409)
(0.472858,0.525748)
(0.649746,0.278991)
(0.282167,0.648376)
(5 rows)

Limit  (cost=0.42..1.11 rows=10 width=16) (actual time=0.054..0.068
      rows=10 loops=1)
  ->  Index Only Scan using complex_test_v_idx on complex_test
        (cost=0.42..41539.82 rows=605000 width=16)
        (actual time=0.054..0.066 rows=10 loops=1)
        Index Cond: (v > '(0.5,0.5)'::complex)
        Heap Fetches: 10
Planning time: 0.198 ms
Execution time: 0.099 ms
```


- ▶ Возвращают несколько значений (строк).
- ▶ Функция вызывается отдельно на каждую возвращаемую строку.
- ▶ Память, выделенную для возврата одной строки, не обязательно освобождать. Контекст памяти очищается между вызовами. Для памяти, которая должна сохраняться между вызовами, есть отдельный контекст.

```
/* Контекст, который будет сохраняться между вызовами функции */
typedef struct
{
    Complex current; /* Текущее значение*/
    Complex step;    /* Шаг */
} gsc_fctx;

Datum
generate_series_complex(PG_FUNCTION_ARGS)
{
    FuncCallContext *funcctx;
    gsc_fctx *fctx;

    if (SRF_IS_FIRSTCALL()) /* Первоначальная инициализация */
    {
        /* Достаём значения аргументов */
        Complex *start = (Complex *)PG_GETARG_POINTER(0);
        Complex *finish = (Complex *)PG_GETARG_POINTER(1);
        int64 count = PG_GETARG_INT64(2);
        MemoryContext oldcontext;

        .....
```

```
.....  
    funcctx = SRF_FIRSTCALL_INIT();  
    /* Переключаемся в постоянный контекст памяти */  
    oldcontext = MemoryContextSwitchTo(  
        funcctx->multi_call_memory_ctx);  
  
    /* Заполняем данные контекста */  
    fctx = (gsc_fctx *)palloc(sizeof(gsc_fctx));  
    fctx->current = *start;  
    fctx->step.x = (finish->x - start->x) / (double)(count - 1);  
    fctx->step.y = (finish->y - start->y) / (double)(count - 1);  
    funcctx->user_fctx = fctx;  
  
    /* Переключаем контекст памяти обратно */  
    MemoryContextSwitchTo(oldcontext);  
    /* Заранее знаем, сколько значений вернём */  
    funcctx->max_calls = count;;  
}  
.....
```

```

.....
funcctx = SRF_PERCALL_SETUP();
fctx = funcctx->user_fctx;
/* Нужно ещё возвращать значения? */
if (funcctx->call_cntr < funcctx->max_calls)
{
    /* Копируем текущее значение в текущий контекст памяти */
    Complex *result = (Complex *)palloc(sizeof(Complex));
    memcpy(result, &fctx->current, sizeof(Complex));
    /* Увеличиваем текущее значение */
    fctx->current.x += fctx->step.x;
    fctx->current.y += fctx->step.y;
    /* Возвращаем приготовленную копию */
    SRF_RETURN_NEXT(funcctx, PointerGetDatum(result));
}
else
{
    SRF_RETURN_DONE(funcctx);
}
}

```

```
-- Определения на уровне SQL
CREATE FUNCTION generate_series_complex(complex, complex, bigint)
  RETURNS SETOF complex AS 'MODULE_PATHNAME'
  LANGUAGE C IMMUTABLE STRICT;
```

```
# SELECT * FROM generate_series_complex('(0, 0)', '(1, 1)', 5);
 generate_series_complex
-----
(0,0)
(0.25,0.25)
(0.5,0.5)
(0.75,0.75)
(1,1)
(5 rows)
```

- ▶ Писать на С под PostgreSQL не так уж страшно.
- ▶ Можно начинать с расширений, постепенно преодолевая порог вхождения.
- ▶ Welcome to community!

Спасибо за внимание!