Процедурные языки для server-side программирования в PostgreSQL

Иван Панченко
i.panchenko@postgrespro.ru

Postgres PROFESSIONAL

Pro & Contra

Процедурные языки лочат Вас на конкретную СУБД	Но могут быть использованы, чтобы скрыть внутреннюю специфику СУБД
Интерпретатор, встроенный в процесс СУБД привносит туда все свои глюки, лики, баги и дырки	Но также и полезную функциональность
Еще одна сущность на спину админам	Но зато есть возможность одной транзакцией выкатывать изменения в структуре таблиц и в хотя бы части кода.
Разработчикам нужно знать еще и этот язык	Не такой уж это и язык. Но можно будет аккуратно расслоить логику. Иногда – упростить деплой.

Базовое предположение

Процедурные языки зачем-то нужны.

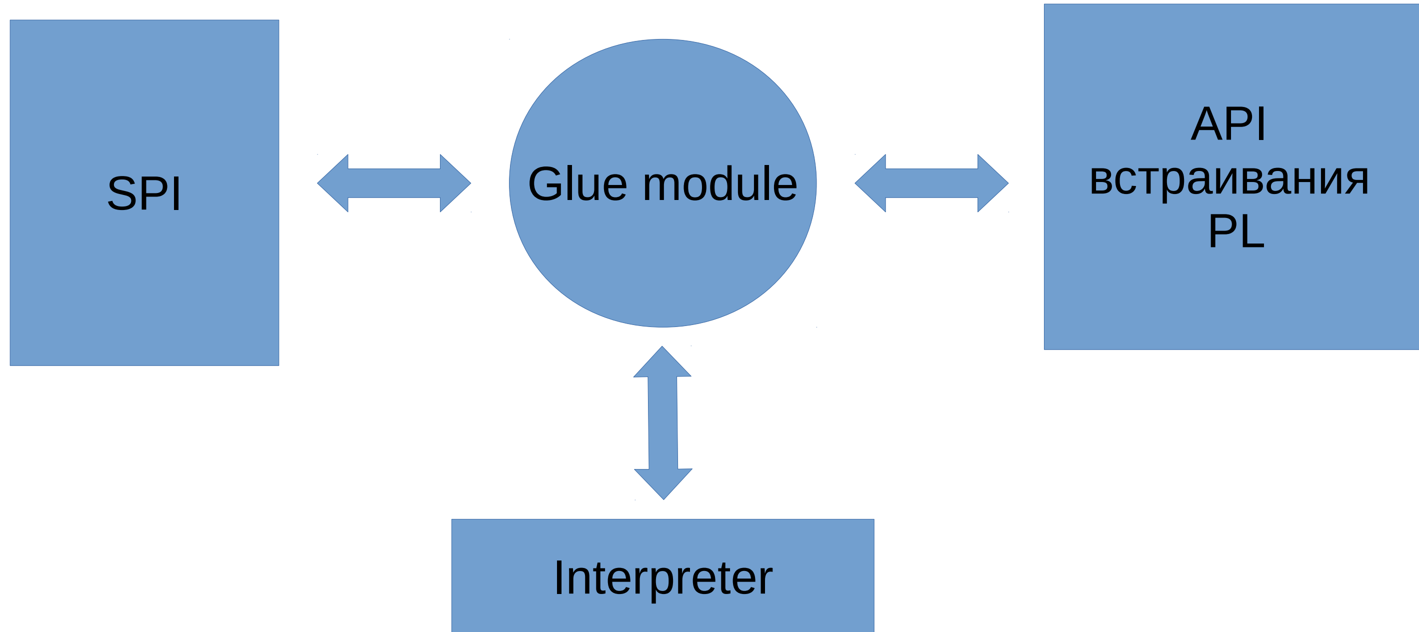
Что-то между

Промежуточное положение между SQL и C

PostgreSQL	C	PL/{Perl,Python,R,Lua,V8,.....}	PL/PgSQL
Oracle	C	Java	PL/SQL
DB/2	C		SQL PL, PL/SQL
MS SQL	C	.Net	T-SQL

PostgresPROFESSIONAL

Их место в PostgreSQL



Postgres PROFESSIONAL

HANDLER INSIDE

Задачи language handler:

- Инициализировать интерпретатор языка
- Получить/скомпилировать исходный код функции
- Закешировать результат компиляции
- Преобразовать параметры
- Предоставить доступ к SPI
- Запустить код
- Обработать исключения
- Преобразовать результат выполнения функции

Как создать PL/*

```
CREATE FUNCTION mylang()
    RETURNS language_handler
    AS 'mylang.so'
    LANGUAGE C;

CREATE [TRUSTED] PROCEDURAL LANGUAGE plmy
    HANDLER mylang
    INLINE mylang_inline
    VALIDATOR mylang_validator;
```

Эти функции *пока* можно писать только на C

HANDLER – функция-обработчик выполнения PL-функции
INLINE – функция-обработчик PL-однострочников
VALIDATOR – функция-валидатор

Хороший тон – запаковать PL в extension.

Postgres PROFESSIONAL

SPI

SPI	libpq
int SPI_connect(void)	PGconn *PQconnectdb(const char *conninfo)
int SPI_finish(void)	void PQfinish(PGconn *conn)
int SPI_exec(const char * command, long count)	PGresult *PQexec(PGconn *conn, const char *command)
int SPI_execute_with_args(const char *command, int nargs, Oid *argtypes, Datum *values, const char *nulls, bool read_only, long count)	PGresult *PQexecParams(PGconn *conn, const char *command, int nParams, const Oid *paramTypes, const char * const *paramValues, const int *paramLengths, const int *paramFormats, int resultFormat)

SPI

SPI	libpq
SPIPlanPtr SPI_prepare(const char * command, int nargs, Oid * argtypes)	PGresult *PQprepare(PGconn *conn, const char *stmtName, const char *query, int nParams, const Oid *paramTypes)
int SPI_execute_plan_with_paramlist(SPIPlanPtr plan,ParamListInfo params, bool read_only, long count)	PGresult *PQexecPrepared(PGconn *conn, const char *stmtName,int nParams, const char * const *paramValues, const int *paramLengths, const int *paramFormats, int resultFormat)

Postgres PROFESSIONAL

SPI

SPI	libpq
uint32 SPI_processed; oid SPI_lastoid; SPITupleTable *SPI_tuptable; int SPI_result;	
void * SPI_palloc(Size) void * SPI_repalloc(void * , Size) void SPI_pfree(void * pointer)	

Postgres PROFESSIONAL

Hello World

```
do ' warn "КОЛБАСА"; ' language plperl;
WARNING:  КОЛБАСА at line 1.
CONTEXT:  PL/Perl anonymous code block
DO
```

```
do ' print "КОЛБАСА"; '
language plperl;
DO
```

Use elog !!

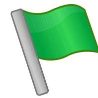
Trusted vs untrusted

plperlu	plperl
\c – httpd do ' ' language plperlu; ERROR: permission denied for language plperlu	\c - httpd do ' ' language plperl; DO
\c – postgres do ' ' language plperlu;	
do 'use Digest::MD5; ' language plperlu; DO	do 'use Digest::MD5; ' language plperl; ERROR: Unable to load Digest/MD5.pm into plperl at line 1.
do 'open(X,">/tmp/q"); print X 0; close(X); ' language plperlu; DO	do 'open(X,">/tmp/q"); print X 0; close(X); ' language plperl; ERROR: 'open' trapped by operation mask at line 1.

PostgresPro PROFESSIONAL

Persistent vars

```
do 'our $x= 2' language plperl;
DO
do 'warn $x;' language plperl;
WARNING:  2 at line 1.
```


Caching

Not transaction-aware

Конфигурация

Параметры в файле postgresql.conf

```
plperl.use_strict = on
plperl.on_init = 'use Data::Dumper; '
```

И тогда:

```
do 'my $x=2; elog(NOTICE, Dumper($x));'
         language plperl;
```

NOTICE: $VAR1 = 2;

PROFESSIONAL Postgres

Работа с данными

Главная функция

```
$result = spi_exec_query($query, $limit);

do language plperl 'warn Dumper(spi_exec_query(
    "select 1 as w"));';

$VAR1 = {
          'processed' => 1,
          'status' => 'SPI_OK_SELECT',
          'rows' => [
                      {
                        'w' => '1'
                      }
                    ]
        };
```

Postgres PROFESSIONAL

С данными аккуратнее

--######### Готовим план и спасаемся от SQL Injections ######

```
do language plperl $$  ## подготовим запрос
   our $plan = spi_prepare(
      'select count(*) from pg_tables
           where relname ~ $1 ',
      'text'
   );
$$;
do language plperl '  ## выполним запрос
   warn Dumper spi_exec_prepared(our $plan, "tab")
';
do language plperl '  ## подчистим
   spi_freeplan(our $plan);
';
```

PROFESSIONAL Postgres

С данными еще аккуратнее

--   Фетчим по одной строке

```
do language plperl $xx$
  my $plan = spi_prepare(
    'select oid, relname from pg_class where relname ~ $1 ',
    'text'
  );

  my $result = spi_query_prepared($plan, 'index');

  while (defined (my $row = spi_fetchrow($result))) {
      warn Dumper $row;  # по одной строке
  }
  warn ((split(/\s/,read_file("/proc/$$/statm")))[1]);
  spi_freeplan($plan);  ### иначе память !!
$xx$;
```

Функции на всех PL создаются примерно одинаково

```
create or replace function my1 ( X int )
      returns int4[]
      language plperl
      as $$
     my ($id) = @_;
     if ($id) {
      warn Dumper(spi_exec_query(
                  "select * from pg_class limit 1"
      ));
     }
     return [2,3,4,5];
$$ ;
```

Польза!

Функции, написанные на PL/*, открывают умопомрачительные возможности для:

- Доступа к библиотекам
- Программирования сложных алгоритмов
- Доступа к данным, лежащим вне БД (включая информацию о здоровье сервера)
- Управлению чем-либо, кроме данных БД
- И т.п.

Функция может возвращать таблицу

```
CREATE TYPE song AS (id int, lyrics text);

CREATE OR REPLACE FUNCTION get_songs (INTEGER)
    RETURNS SETOF song LANGUAGE 'plperl' AS $$
 for(1..shift) {
  return_next {
    id =>  int rand(12345),
    lyrics=> join '',  (' ','.','/', 0..9, 'A'..'Z', 'a'..'z')
            [map {rand 65} (0..56)]
  };
 }
 return;
$$ ;

select * from get_songs(122);
   id   |                          lyrics
-------+-------------------------------------------------------
  4572 | jRfqCtPHFtI l.FIKUtp/p0 ESSefqbJ ayHyk5eRifYj CYloV
….
```

PROFESSIONAL Postgres

Еще польза!

Функции, возвращающие таблицы, открывают умопомрачительные возможности для:

- Преобразования / генерации SQL-запросов
- Генерации данных
- Непростой обработки/агрегации данных
- Извлечения (проксирования) данных откуда угодно
- Создания ORM
- И т.п.

Postgres PROFESSIONAL

Типы данных

Массивы – “из коробки”, остальное – в текстовом виде.

```
$VAR1 = '"key"=>"\'val\'"';
```

9.5

```
CREATE extension hstore_plperl;
CREATE function test_data95 (hstore) RETURNS text
    TRANSFORM FOR TYPE hstore as
   'return Dumper @_' language plperl;
```

```
$VAR1 = {                         +
          'key' => '\'val\''+
        };
```

Postgres PROFESSIONAL

И внутри!

Преобразование типа распространяется и на внутреннюю работу функции !

```
CREATE or replace function test_s95f() returns text as $$
   return Dumper(spi_exec_query("select 'key=>''val'''::hstore"));
$$ TRANSFORM FOR TYPE hstore  language plperl;
SELECT test_s95f();

$VAR1 = {                                                 +
          'processed' => 1,                               +
          'rows' => [                                     +
                      {                                   +
                        'hstore' => {                     +
                                      'key' => '\'val\''+
                                    }                     +
                      }                                   +
                    ],                                    +
          'status' => 'SPI_OK_SELECT'                     +
        };
```

PROFESSIONAL Postgres

9.5!

В PostgreSQL 9.5 появится возможность, создавая новые extension's, добавлять в процедурные языки поддержку любых типов данных.

Это удобно, но раньше то же самое можно было бы делать руками, внутри функций или модифицируя handler'ы языков.

Выигрыш в производительности эта возможность ДАЕТ НЕ ВСЕГДА

	Execution time
TRANSFORM json	100%
JSON::XS	92%

PROFESSIONAL Postgres

Разные типы (есть чем заняться)

```
do $$ warn Dumper spi_exec_query(
    q!select 'Моська'    ::text     as a,
    'Слон'               ::bytea    as b,
    ARRAY['лает', 'на']             as c,
    '{"вес":25}'         ::json     as d,
    '''рост''=>''v37'''  ::hstore   as e!,
);  $$  language plperl;

        'a' => "\x{41c}\x{43e}\x{441}\x{44c}\x{43a}\x{430}",
        'b' => '\\xd0a1d0bbd0bed0bd',
        'c' => bless( {
                'array' => [
                    "\x{43b}\x{430}\x{435}\x{442}",
                    "\x{43d}\x{430}"
                ],
                'typeoid' => 1009
         }, 'PostgreSQL::InServer::ARRAY' ),
         'd' => '{"a":25}',
         'e' => '"k"=>"\'v\'"',
```

PROFESSIONAL Postgres

JSONB на PL/Perl

```
CREATE FUNCTION jsonb_to_plperl(val internal) RETURNS internal
LANGUAGE C STRICT IMMUTABLE
AS 'MODULE_PATHNAME';

CREATE FUNCTION plperl_to_jsonb(val internal) RETURNS jsonb
LANGUAGE C STRICT IMMUTABLE
AS 'MODULE_PATHNAME';

CREATE TRANSFORM FOR jsonb LANGUAGE plperl (
    FROM SQL WITH FUNCTION jsonb_to_plperl(internal),
    TO SQL WITH FUNCTION plperl_to_jsonb(internal)
);
```

Postgres PROFESSIONAL

Функции трансформации

```
PG_FUNCTION_INFO_V1(jsonb_to_plperl);

Datum
jsonb_to_plperl(PG_FUNCTION_ARGS)
{
	Jsonb  *in = PG_GETARG_JSONB(0);
	……
	return PointerGetDatum(newRV((SV *) v));
}

PG_FUNCTION_INFO_V1(plperl_to_jsonb);

Datum
plperl_to_jsonb(PG_FUNCTION_ARGS)
{
	RV *ptr = SvRV((SV *) PG_GETARG_POINTER(0));
	……
	PG_RETURN_POINTER(out);
}
```

Postgres PROFESSIONAL

Языков много

Из коробки:
PL/PGSQL
PL/Perl
PL/Tcl
PL/Python

PL/PerlU и т.п.

Из тумбочки:
PL/v8
PL/R
PL/php
PL/sh
PL/lua
PL/Java

select * from pg_pltemplate;

Postgres PROFESSIONAL

Feature matrix

https://wiki.postgresql.org/wiki/PL_Matrix

	PL/PgSQL	PL/Perl	PL/Python	PL/V8
Собственные функции	нет	есть	есть	есть
Глобальный контекст	нет	есть	есть	есть
Однострочники	есть*	есть	есть	есть
Нативный JSON	Совсем нет	почти	почти	Да!
Untrusted	нет	есть	есть	нет
Window functions	нет	нет	нет	есть
Cursor	есть	нет	нет	есть
Prepared statements	нет	есть	есть	есть
Отладка	нет	есть	?	есть
subtransactions	savepoint	savepoint	savepoint	есть

PROFESSIONAL
Postgres

Performance

В таблице – время выполнения тестов (больше - хуже)

	PL/PgSQL	PL/Perl	PL/v8
Overhead	0.05ms	0.05ms	0.05ms
CPU intensive	462s	92s	801ms
JSON conversion	-	580ms (21s PP)	770ms
Database	12s	18s	14s
City Mix	To be discussed		

Как ускорить PL/PgSQL ?

Реализовать его по другому. Надо ли ? Как?

Основная идея: перенести в compile-time все что можно.

PL/v8 – подходящая платформа для executor

PL/Perl – подходящая платформа для компилятора

v8_plpgsql_compile(code text) returns text;

PROFESSIONAL Postgres

Что ускорить в PL/PgSQL ?

```
for i IN 1..n LOOP
            s :=(arithmetics);
END LOOP;
```

Набор регулярных выражений

```
$text =~ s/FOR\s+(\w+)\s+IN\s+(\w+)\.\.(\w+)\s+LOOP/
 for(var $1 = $2;  $1 < $3; $1++) {
/gsi
;
```

ОНО!   УСКОРЯЕТ В СОТНИ РАЗ!

Postgres PROFESSIONAL

Почти тот же подход

Ускорители PostgreSQL:

Vitesse DB
Vitesse X

LLVM-компиляция запросов

select * from t where i = 10 or i = 20 or i = 30;

select count(*), sum(i*i), avg(i) from t;

Postgres PROFESSIONAL

Спасибо за внимание!

Иван Панченко

i.panchenko@postgrespro.ru
ООО “Постгрес Профессиональный”
http://postgrespro.ru/