PL/NoSQL
Программирование на не-SQL-образных процедурных языках

Иван Панченко

PROFESSIONAL Postgres

Зоопарк процедурных языков

1) Идущие в коробке

PL/PgSQL PL/Python

PL/Perl PL/TCL

2) Другие

PL/V8 PL/Ruby ☹

PL/Java PL/Go

PL/PHP ☹ PL/Lua

PL/R PL/Sh

Установка PL/языков

Ubuntu (репозиторий pgdg)

```
sudo apt -y install postgresql-plperl-11
sudo apt -y install postgresql-plpython-11
sudo apt -y install postgresql-plpython3-11

sudo apt -y install postgresql-plv8-10
```

- https://github.com/AoAnima/PLV8-Postgresql11-Ubuntu-18

Установка PL/языков

CentOS

```
yum -y install postgresql11-plperl
yum -y install postgresql11-plpython
~~yum -y install postgresql11-plpython3~~
~~yum -y install postgresql11-plv8~~
```

Установка PL/языков в БД

```
CREATE EXTENSION plperl;
CREATE EXTENSION plpythonu;
CREATE EXTENSION plpython3u;
CREATE EXTENSION plv8;
```

Собрать pl/perl,python руками?

```
./configure --with-perl --with-python
   [PYTHON=/usr/bin/python3]
```

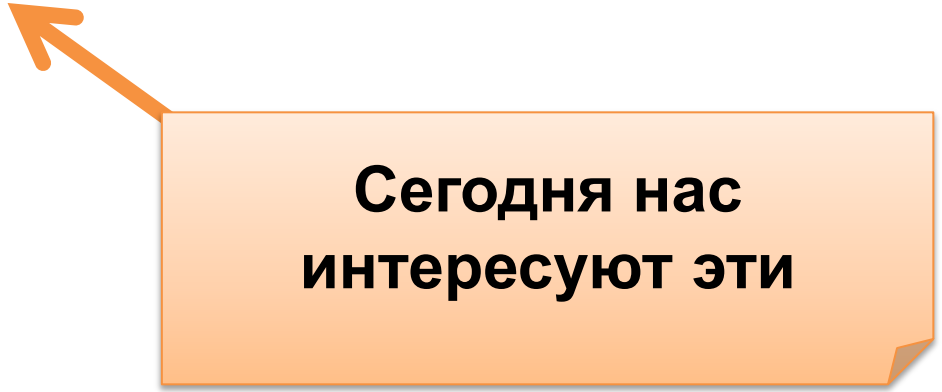
Собрать plv8 голыми руками?

Осторожно, Трафик!!! И другие неприятности.

```
yum -y install postgresql11-devel make git
wget
  https://github.com/plv8/plv8/archive/v2.3.9.tar.
  gz
tar xzf v2.3.9.tar.gz
cd plv8-2.3.9/
make PG_CONFIG=/usr/pgsql-11/bin/pg_config static
make PG_CONFIG=/usr/pgsql-11/bin/pg_config install
```

Альтернативная классификация зоопарка

1) «нормальные»
2) DSL
   PL/Proxy
3) PL/SQL
4) C



Эй, а как же стандарты?

SQL/PSM

SQL/JRT

В PostgreSQL: PL/PgPSM, PL/Java

А что, если языков не хватит ??

```
CREATE [ OR REPLACE ] [ TRUSTED ] [ PROCEDURAL ]
    LANGUAGE имя

HANDLER обработчик_вызова_функции

[ INLINE обработчик_инлайн_блока ]

[ VALIDATOR функция_проверки_кода_функции ]
```

- Эти три функции можно написать на C

https://postgrespro.ru/docs/postgresql/11/plhandler

https://postgrespro.ru/docs/postgresql/11/sql-createlanguage

Доверенные и недоверенные

TRUSTED	UNTRUSTED
Любой юзер может создать функцию. Нельзя работать с I/O, в т.ч. с сетью	Создает функцию только суперъюзер. Можно всё, что можно.
plpgsql plv8 plperl pljava	plperlu pljavau plpython2u plpython3u

PL/pgSQL vs PL/*

	PL/PgSQL	PL /*
Хорошо	Нативная работа с типами данных	Глобальный контекст интерпретатора Доступны сеть и диск (u)
Плохо	Только trusted Дорогой вызов функций Слабоват для JSON	Требуются преобразования типов

Особенности всех PL/*

- Работа с БД напрямую через интерфейс SPI (хотя не запрещены и обычные способы)
- Своё управление памятью
- Свои структуры данных
- Своя обработка ошибок

Особенности PL/Perl

- Интерпретатор создается при первом использовании
- PL/PerlU и PL/Perl – разные экземпляры интерпретатора
- Параметры:

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

plperl.on_plperl_init = ' ... '

plperl.on_plperlu_init = ' ... '

plperl.use_strict  = on
```

Особенности PL/Python

- Только untrusted
- Интерпретатор создается при первом использовании
- PL/Python2 и PL/Python3 вместе использовать нельзя
- Указать, что делать при инициализации, нельзя
- Однострочники делать неудобно
- SD – статический словарь; GD - глобальный словарь

Особенности PL/v8

- Только trusted
- Автоматический маппинг JSON (JSONB)
- Возможность определять window functions
- Возможность делать подтранзакции
- Упрощенный вызов других функций PL/v8
- `plv8.execition_timeout=300` (по умолчанию не включено при сборке)
- Инициализация: `plv8.start_proc=my_start_func` //(имя PLv8-функции)

  - Подробнее см https://rymc.io/2016/03/22/a-deep-dive-into-plv8/

Особенности PL/Java

- Это компилируемый язык
- Работа с базой через интерфейс JDBC
- Нужно ограничивать память JVM

Hello world PL/Perl

```
DO $$
    elog(NOTICE,"Hello World");
$$ LANGUAGE plperl;
```

```
NOTICE:  Hello World
DO
```

Можно использовать обычные функции warn и die

Hello world PL/Python

```
DO $$
   plpy.notice('Hello World', hint="Будь
   здоров",  detail="В деталях")
$$ LANGUAGE plpythonu;
```

```
NOTICE:  Hello World
DETAIL:  В деталях
HINT:  Будь здоров
```

error, warning, debug, log, info, fatal

См. доку

Hello world PL/v8

```
DO $$
  plv8.elog(NOTICE, 'Hello World');
$$ LANGUAGE plv8;
NOTICE:  Hello World
DO
```

Можно использовать `throw 'Errmsg'`

Работа с БД PL/Perl

```
DO $$ warn Data::Dumper::Dumper(
   spi_exec_query('select 57 as x'))
   $$ LANGUAGE plperl;

WARNING: $VAR1 = {
     'status' => 'SPI_OK_SELECT',
     'processed' => 1,
     'rows' => [{'x' => '57'}]
   };
```

Работа с БД PL/Python

```
DO $$ plpy.notice(
   plpy.execute('select 57 as x'))
   $$ LANGUAGE plpythonu;

NOTICE:  <PLyResult status=5 nrows=1
   rows=[{'x': 57}]>
```

Работа с БД PL/Python

```
DO $$ plpy.notice(
   plpy.execute('select 57 as x')[0]['x'])
   $$ LANGUAGE plpythonu;

NOTICE: 57
```

Работа с БД PL/v8

```
DO $$ plv8.elog(NOTICE,
   JSON.stringify(
      plv8.execute('select 57 as x')));
   $$ LANGUAGE plv8 ;

NOTICE:  [{"x":57}]
```

PL/Perl : экранирование

Функции из SPI:

`quote_literal` – берет в апострофы и удваивает ‘ и \

`quote_nullable` – то же, но undef => NULL

`quote_ident` – берет в кавычки, если надо

```
DO $$
   warn "macy's";
   warn quote_literal("macy's");
   $$ LANGUAGE plperl;
```

PL/Perl : экранирование (2)

А также:

```
encode_bytea                decode_bytea
encode_array_literal        encode_typed_literal
encode_array_constructor
```

```
DO $$
warn quote_typed_literal(
    ["один", "двадцать один"], "text[]"  );
$$ LANGUAGE plperl;
```

PL/Python : экранирование

plpy.quote_literal

plpy.quote_nullable

plpy.quote_ident

```
DO $$ plpy.notice(
   plpy.quote_literal("Macy's"))
$$ LANGUAGE plpythonu;
NOTICE:  'Macy''s'
```

PL/v8 : экранирование

plv8.quote_literal

plv8.quote_nullable

plv8.quote_ident

```
DO $$ plv8.elog(NOTICE,
plv8.quote_nullable("Macy's")); $$
LANGUAGE plv8 ;
NOTICE:  'Macy''s'
```

Производительность!

Сравним её на коленке.

Посмотрим, как влияет prepare запроса

Заодно посмотрим, как им пользуются

```
\timing
```

Производительность (1.1)

```
SELECT count(*) FROM pg_class;
```

0.5ms

```
DO $$ DECLARE a int; BEGIN SELECT count(*)
   INTO a FROM pg_class; END; $$ LANGUAGE
   plpgsql;
```

0.7ms

```
DO $$ my $x = spi_exec_query('SELECT
   count(*) FROM pg_class'); $$ LANGUAGE
   plperl;
```

0.7ms

Производительность (1.2)

```
DO $$ x = plpy.execute('SELECT count(*) FROM
   pg_class'); $$ LANGUAGE plpythonu;

DO $$ x = plpy.execute('SELECT count(*) FROM
   pg_class'); $$ LANGUAGE plpython3u;

DO $$ var x  = plv8.execute('SELECT count(*)
   FROM pg_class'); $$  LANGUAGE plv8 ;
```

0.8ms

0.9ms

0.9ms

Ноль – он и в Африке ноль.

Производительность (2.1)

```
DO $$ DECLARE a int; i int;
   BEGIN FOR i IN 0..999 LOOP
      SELECT count(*) INTO a FROM pg_class; END
   LOOP; END;
   $$ LANGUAGE plpgsql;
```

40ms

```
DO $$ for (0..999) {
   spi_exec_query('SELECT count(*) FROM
      pg_class');
   } $$ LANGUAGE plperl;
```

80ms

Производительность (2.2)

```
DO $$ for i in range (0,1000) :
   plpy.execute('SELECT count(*) FROM
   pg_class')

$$ LANGUAGE plpythonu;

DO $$ for(var i=0;i<1000;i++)
   plv8.execute('SELECT count(*) FROM
   pg_class'); $$  LANGUAGE plv8 ;
```

80ms

100ms

Производительность с prepare (3.1)

```
DO $$
   my $h = spi_prepare(
        'SELECT count(*) FROM pg_class'
   );
   for (0..999) { spi_exec_prepared($h); }
   spi_freeplan($h);

$$ LANGUAGE plperl;
```

45ms

Производительность с prepare (3.2)

```
DO $$
   h = plpy.prepare(
      'SELECT count(*) FROM pg_class'
   )
   for i in range (0,1000) :
      plpy.execute(h)

$$ LANGUAGE plpythonu;
```

45ms

50ms

Производительность с prepare (3.3)

```
DO $$
   var h=plv8.prepare(
      'SELECT count(*) FROM pg_class'
   );
   for(var i=0;i<1000;i++) h.execute();

$$  LANGUAGE plv8 ;
```

55ms

Производительность вычислений (4.1)

```
DO $$
   DECLARE i int; a bigint;
   BEGIN a=0;
    FOR i IN 0..1000000 LOOP
      a=a+i*i::bigint;
    END LOOP;
   END;

$$ LANGUAGE plpgsql;
```

280ms

Производительность вычислений (4.2)

```
DO $$
   my $a=0;
   for my $i (0..1000000) { $a+=$i*$i; };
$$ LANGUAGE plperl;

DO $$
   a=0
   for i in range(0,1000001):
      a=a+i*i
$$ LANGUAGE plpythonu;
```

80ms

80ms

100ms

Производительность вычислений (4.3)

```
DO $$
   var a=0;
   for(var i=0;i<=1000000;i++) a+=i*i;

$$ language plv8;
```

4ms !!

Доверяй, но проверяй

```
DO $$
   var a=0;
   for(var i=0;i<=1000000;i++) a+=i*i;
   plv8.elog(NOTICE, a);

$$ language plv8;


333333833333127550
```

4ms !!

Float-арифметика в JS

```
DO LANGUAGE plv8 $$
    plv8.elog(NOTICE, parseInt(33333383333312755033)) $$;

NOTICE:  33333383333312754000
```

- В Javascript целое представляется в форме float.
- Поэтому в предыдущих примерах результат получается быстро, но не точно:

33333383333312755O вместо 33333383333500000

$$\sum = n*(n+1)*(2n+1)/6$$

PL/Perl : Память

- Хорошо (не течёт):

```
CREATE OR REPLACE function cr()
   RETURNS int LANGUAGE plperl AS $$

   return spi_exec_query(
     'select count(*) from pg_class'
   )->{rows}->[0]->{count};

 $$;
```

PL/Perl : Память (2)

- Плохо (течёт):

```
CREATE OR REPLACE function cr()
   RETURNS int LANGUAGE plperl AS $$

   my $h = spi_prepare(
      'select count(*) from pg_class');

    return spi_exec_prepared($h)
        ->{rows}->[0]->{count};

 $$;
```

PL/Perl : Память (3)

- Хорошо (не течёт):

```
CREATE OR REPLACE function cr()
   RETURNS int LANGUAGE plperl AS $$
   my $h = spi_prepare(
      'select count(*) from pg_class');
   my $res = spi_exec_prepared($h)
       ->{rows}->[0]->{count};
   spi_freeplan($h);
   return $res;

 $$;
```

PL/Python : Память

- Хорошо (не течёт):

```
CREATE OR REPLACE function cr3() RETURNS int
   LANGUAGE plpythonu as $$

return plpy.execute(
   'select count(*) from pg_class'
   )[0]['count']

$$;
```

PL/v8 : Память

- Хорошо (не течёт):

```
CREATE OR REPLACE FUNCTION crq() RETURNS int
   LANGUAGE plv8 AS $$

return plv8.execute(
   'select count(*) from pg_class'

)[0].count;

$$;
```

PL/v8 : Память (2)

- Плохо (течёт):

```
CREATE OR REPLACE FUNCTION crq() RETURNS int
   LANGUAGE plv8 AS $$
   var h = plv8.prepare(
      'select count(*) from pg_class');

   return h.execute()[0].count;

$$;
```

PL/v8 : Память (3)

- Хорошо (не течёт):

```
CREATE OR REPLACE FUNCTION crq() RETURNS int
   LANGUAGE plv8 AS $$
   var h = plv8.prepare(
       'select count(*) from pg_class');
   var r = h.execute()[0].count;
   h.free();
   return r;
   $$;
```

PL/Perl : Параметры

- В каком виде они попадают в Perl?

```
CREATE OR REPLACE FUNCTION crq(a int, b
   bytea, c int[], d jsonb ) RETURNS void
   LANGUAGE plperl AS

$$ warn Dumper(@_) $$;

SELECT crq(1,'abcd',
   ARRAY[1,2,3],'{"a":2,"b":3}');
```

TRANSFORM

- Возможность определять функции преобразования типов (с 9.6): CREATE TRANSFORM

```
CREATE TRANSFORM FOR hstore LANUGAGE
   plperl (

 FROM SQL WITH FUNCTION
   hstore_to_plperl(internal),

 TO SQL WITH FUNCTION
   plperl_to_hstore(internal)

);
```

TRANSFORM для JSONB (с Pg11)

```
CREATE OR REPLACE FUNCTION crq2(a int, b bytea, c
   int[], d jsonb )
   RETURNS void LANGUAGE plperl
   TRANSFORM FOR TYPE jsonb AS $$
      warn Dumper(@_);

$$;
```

- Тогда JSONB попадет в функцию в виде структуры, а не строки.

PL/Python : Параметры

- В каком виде они попадают в Python?

```
CREATE OR REPLACE FUNCTION pdump(a int, b
   bytea, c int[], d jsonb ) RETURNS void
   LANGUAGE plpythonu AS

 $$ plpy.warning(a,b,c,d) $$;

SELECT pdump(1,'abcd',
   ARRAY[1,2,3],'{"a":2,"b":3}');
```

PL/Python : Параметры (2)

- Результат

```
(1,
   '\x124V',      # это лучше чем в PL/Perl
   [1, 2, 3],     # это тоже
   '{"a":"b"}')   # то же самое
```

- Transform тоже работает
- Начиная с Pg10, работают многомерные массивы (в PL/Perl – давно)

PL/v8 : Параметры

- В каком виде они попадают в JS?

```
CREATE OR REPLACE FUNCTION jdump(a int, b
   bytea, c int[], d jsonb ) RETURNS void
   LANGUAGE plv8 AS

$$ plv8.elog(WARNING,a,b,c,d) $$;

SELECT jdump(1,'abcd',
   ARRAY[1,2,3],'{"a":2,"b":3}');
```

PL/v8 : Параметры (2)

- Результат

```
(1
   18,52,86 // массив байтов
   1,2,3    // массив чисел
   [object Object] // JSON !! супер
```

- Transform не нужен, он реализован сразу в plv8
- Даты тоже преобразуются

PL/v8 : Версия

- PL/Perl, PL/Python – нет своей версии
- PL/v8:

`plv8.version` (не функция)

```
DO LANGUAGE plv8 $$
   plv8.elog(NOTICE,plv8.version);
   $$;
```

PL/v8 : Быстрый доступ к функциям

- Возвращает функцию по ее имени (полиморфизм – ошибка)

```
plv8.find_function(name);
```

```
DO LANGUAGE plv8 $$
   plv8.find_function('jdump')(1, 'abc');
   $$;
```

PL/v8 : Инициализация

- Стартовая функция определяется GUC my_init

```
CREATE OR REPLACE FUNCTION my_init()
   RETURNS void LANGUAGE plv8 AS $$
   this.xxx = function() { return 57; };
   this.qqq = 157;  $$;

SET plv8.start_proc = 'my_init';

DO LANGUAGE plv8 $$
   plv8.elog(NOTICE, qqq, xxx(3) );
   $$;
```

PL/Perl : Запросы с параметром

```
DO LANGUAGE plperl $$
my $h= spi_prepare('SELECT * FROM pg_class WHERE
relname ~ $1', 'text' );

warn Dumper(spi_query_prepared($h, 'pg'));
spi_freeplan($h);
$$;
```

PL/Python : Запросы с параметром

```
DO LANGUAGE plpythonu $$
h= plpy.prepare('SELECT * FROM pg_class WHERE
   relname ~ $1', ['text'] )

plpy.notice(plpy.execute (h, ['pg']))
$$;
```

PL/v8 : Запросы с параметром

```
DO LANGUAGE plv8 $$
var h= plv8.prepare('SELECT * FROM pg_class WHERE
   relname ~ $1', ['text'] );

plv8.elog(NOTICE, h.execute (['pg']));
h.free();
$$;
```

PL/Perl : Работа с курсором

```
DO LANGUAGE plperl $$
my $cursor = spi_query('SELECT * FROM pg_class');
my $row;
while(defined($row = spi_fetchrow($cursor))) {
        warn $row->{relname};
}
$$;
```

PL/Python : Работа с курсором

```
h = plpy.prepare('SELECT …');
cursor = plpy.cursor(h);
for row in cursor:
    …

cursor.close() // не обязательно
```

PL/v8 : Работа с курсором

```
var h = plv.prepare('SELECT …');
var cursor = h.cursor();
var row;
while(row = cursor.fetch()) {
…
}
cursor.close();
h.free();
```

PL/Perl : Транзакции

```
CREATE PROCEDURE . . .


spi_commit();

spi_rollback();
```

PL/Python : Транзакции

```
CREATE PROCEDURE . . .


plpy.commit();

plpy.rollback();
```

PL/Python : Подтранзакции

```
try:
    with plpy.subtransaction():
        plpy.execute("…")
        plpy.execute("…")
except plpy.SPIError, e:
   . . .
else:
    . . .
```

PL/v8 : Подтранзакции

```
try {
    plv8.subtransaction(function() {
        plv8.execute('UPDATE…');
        plv8.execute('UPDATE…');
    });

}

catch(e) {
    …

}
```

Зачем все это нужно

- Работа со сложными структурами данных и алгоритмами
- Формирование динамических SQL (ORM, отчеты)
- Большой набор библиотек из Perl, Python и т.п.
- Работа с внешними данными
- Прежде чем писать на C, попробуй на ***

THE END

- Вопросы:

  i.panchenko@postgrespro.ru

- Ещё есть, что улучшить!

  Участвуйте ☺