Python.

Модули расширения.

Модули расширения

- Зачем писать свой модуль расширения?

Модули расширения.

- Как же можно написать свой собственный модуль расширения на C/C++? Существует множество способов:
  - Программный интерфейс C API
  - Библиотека boost::python
  - Стандартная библиотека ctypes
  - SWIG (Simplified Wrapper and Interface Generator)
  - …

Модули расширения.

- Swig – является терминальной программой, которая создает оберточный Си файл для нашего файла на C/C++ при помощи специального интерфейсного файла, в котором мы указываем функции, необходимые для использования в Python

Модули расширения. Пример.

- Напишем модуль, суммирующий 2 матрицы большого размера, заполненные случайными числами. Он будет содержать всего одну ф-цию:
  - void do_example(size_t input_size);

Модули расширения. Пример.

- Интерфейсный файл:

```
/* File : example.i */
%module example
%{
extern void do_example(size_t input_size);
%}
```

Модули расширения.

- При помощи программы swig создаем модуль на python example.py и файл обертки example_wrap.c
- **~$ swig -python ./example.i**

Модули расширения. Пример.

~$ g++ -c -fPIC ./example.c ./example_wrap.c -I "/usr/include/python2.6"
~$ g++ -shared ./example.o ./example_wrap.o -o _example.so

- Получаем библиотеку _example.so

Модули расширения.

```
>>> import example
>>> dir(example)
['__builtins__', '__doc__', '__file__', '__name__',
'__package__', '_example', '_newclass', '_object',
'_swig_getattr', '_swig_property', '_swig_repr',
'_swig_setattr', '_swig_setattr_nondynamic',
'do_example', 'new', 'new_instancemethod']
>>> example.do_example(100)
>>>
```

Время работы

```
>>> from timeit import Timer  # in Python
>>> Timer("do(3000)","from example_py import do_example as do").timeit(1)
8.6974020004272461

>>> Timer("do(3000)","from example import do_example as do").timeit(1)
0.43890213966369629
```

Pylint

- Pylint — это анализатор кода на python, а именно синтаксиса, возможных ошибок и соответствие кода стандарту
- **~$ pylint ./example_py.py**
- После этого в терминале выведется информация об ошибках и статистика о файле

Сообщения pylint

~$ pylint ./example_py.py --reports=n

No config file found, using default configuration
********** Module example_py
W: 4: Found indentation with tabs instead of spaces
...
W: 20: Found indentation with tabs instead of spaces
C: 1: Missing docstring
C: 3:do_example: Missing docstring
C: 4:do_example: Invalid name "A" (should match [a-z_][a-z0-9_]{2,30}$)
C: 5:do_example: Invalid name "B" (should match [a-z_][a-z0-9_]{2,30}$)
C: 7:do_example: Invalid name "rowA" (should match [a-z_][a-z0-9_]
{2,30}$) C: 8:do_example: Invalid name "rowB" (should match [a-z_][a-
z0-9_]

Отладка

- >>> import pdb
  >>> dir(pdb)
  ['Pdb', 'Repr', 'Restart', 'TESTCMD', '__all__', '__builtins__', '__doc__', '__file__', '__name__', '__package__', '_repr', '_saferepr', 'bdb', 'cmd', 'find_function', 'help', 'line_prefix', 'linecache', 'main', 'os', 'pm', 'post_mortem', 'pprint', 're', 'run', 'runcall', 'runctx', 'runeval', 'set_trace', 'sys', 'test', 'traceback']

Запуск отладчика

```
>>> import pdb
>>> from example_py import do_example as
do_ex
>>> pdb.runcall(do_ex,2)
> /home/ktisha/python_try/
example_py.py(4)do_example()
-> A = []
(Pdb)
```

Методы PDB

- pdb.run(statement[, globals[, locals]]) – запуск цели под контролем отладчика
  - Statement – цель для отладки (в виде строки)
  - Globals, locals – параметры вызова
- pdb.runeval(expression[, globals[, locals]]) – аналогично прошлому, но возвращает значение цели
- pdb.runcall(function[, argument, ...]) – вызов функции. Функция передается не в виде строки

Команды PDB

- h(elp) [command] – Список команд либо информация о конкретной
- w(here) – Выводит часть содержимого стека.
- u(p) – Перемещает текущий кадр стека вверх
- d(own) – Перемещает текущий кадр стека вниз

Breakpoints

- b(reak)
  [[filename:]lineno | function[, condition]]
  – Точка останова в файле или функции

- tbreak
  [[filename:]lineno | function[, condition]]
  – То же, но точка удаляется после прохода.

- cl(ear) [bpnumber [bpnumber ...]]

Breakpoints

- disable [bpnumber [bpnumber ...]]
  - Отключение точек останова
- enable [bpnumber [bpnumber ...]]
  - Включение точек останова
- ignore bpnumber [count]
  - Минимальное число проходов
- condition bpnumber [condition]
  - Активация точки по выражению

Переходы

- commands [bpnumber]
  – Выполнение команд на точке остановки

(Pdb) commands 1
(com) print some_variable
(com) end
(Pdb)

- s(tep)
  – Выполнение строки с вхождением
- n(ext)
  – Выполнение строки без вхождения
- unt(il)
  - Выполнение до перехода за номер строки

Шаги отладчиком

(Pdb) s
> /home/ktisha/python_try/example_py.py(5)do_example()
-> B = []
(Pdb)
(Pdb) n
> /home/victor/python_try/example_py.py(6)do_example()
-> for i in xrange(size):
(Pdb)

Переходы

- r(eturn)
  – Выполнение до выхода из функции
- c(ont(inue))
  – Выполнение до точки останова
- j(ump) lineno
  – Переход на строку. Возможен только внизу стека.
- a(rgs)
  – Выводит аргументы функции
- p expression
  – Вычисляет выражение в текущем контексте
- q(uit)
  – Завершает работу отладчика

Текущее положение

```
(Pdb) l
8     rowB = []
9     for j in xrange(size):
10        rowA.append(rnd.random())
11        rowB.append(rnd.random())
12        A.append(rowA)
13  ->    B.append(rowB)
14    C = []
15    for i in xrange(size):
16        rowC = []
17    for j in xrange(size):
(Pdb)
```

Оптимизация

- Как можно понять, какая именно функция или метод занимают больше всего времени для своего исполнения?

Оптимизация

```
>>> import profile
>>> dir(profile)
['OptionParser', 'Profile', 'Stats', '__all__',
'__builtins__', '__doc__', '__file__', '__name__',
'__package__', '_get_time_resource',
'_get_time_times', '_has_res', 'help', 'main',
'marshal', 'os', 'resgetrusage', 'resource', 'run',
'runctx', 'sys', 'time']
```

```
>>> from example_py import do_example as do_ex
>>> profile.run("do_ex(300)")
450904 function calls in 2.520 CPU seconds
Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
270900    0.700    0.000    0.700    0.000 :0(append)
180000    0.440    0.000    0.440    0.000 :0(random)
     1    0.000    0.000    0.000    0.000 :0(random)
     1    0.000    0.000    2.520    2.520 <string>:1(<module>)
     1    1.380    1.380    2.520    2.520 example.py:3(do_example)
     1    0.000    0.000    2.520    2.520 profile:0(do_ex(300))
```

- ncalls - количество вызовов функции или метода,
- tottime - полное время выполнения кода функции (без времени нахождения в вызываемых функциях),
- percall - тоже, в пересчете на один вызов,
- cumtime - аккумулированное время нахождения в функции, вместе со всеми вызываемыми функциями.
- В последнем столбце приведено имя файла, номер строки с функцией или методов и его имя.

Правила оптимизации

- Не нужно оптимизировать программу, если скорость её выполнения достаточна.
- Используемый алгоритм имеет определённую временную сложность, поэтому перед оптимизацией стоит сначала пересмотреть алгоритм.
- Стоит использовать готовые и отлаженные функции и модули, даже если для этого нужно немного обработать данные. Например, в Питоне есть встроенная функция **sort()**.
- Профилирование поможет выяснить узкие места. Оптимизацию нужно начинать с них.

Правила оптимизации

- Вызов функций является достаточно дорогостоящей операцией, поэтому внутри вложенных циклов нужно стараться избегать вызова функций или, например, переносить цикл в функции.
- Функция, обрабатывающая последовательность, эффективнее, чем обработка той же последовательности в цикле вызовом функции.
- Старайтесь вынести из глубоко вложенного цикла всё, что можно вычислить во внешних циклах. Доступ к локальным переменным более быстрый, чем к глобальным, или чем доступ к полям.
- В случае, если модуль проводит массированную обработку данных и оптимизация алгоритма и кода не помогает, можно переписать критические участки на языке Си

Pipelines

- перенаправление вывода одного программного процесса на ввод другого процесса.

- Модули
  - Pipes (встроенный)
  - Grapevine

Pipelines

```
>>> import pipes
>>> t=pipes.Template()   # abstraction
>>> t.append('tr a-z A-Z', '--')
>>> f=t.open('/tmp/1', 'w')
>>> f.write('hello world')
>>> f.close()
>>> open('/tmp/1').read()
'HELLO WORLD'
```

Grapevine

```
>>> from grapevine import *
>>> xrange(-10, 10) | grep(lambda x: x % 3 == 2)
| (x * (x + 1) for x in STDIN) | list
[90, 42, 12, 0, 6, 30, 72]
```