Cython的一点使用经验

赖勇浩@齐昌网络

2015.9.13.

内容提要

- 自我介绍
- Cython是什么？
- Cython有什么能力？
- 一点使用经验
- Q&A

自我介绍

- 赖勇浩，Python 的重度用户，PyCon 的老朋友。
- 从业10年，创立广州齐昌网络科技有限公司，主营技术咨询与开发外包。
- 合著有《编写高质量代码：改善Python程序的91个建议》一书。

Cython是什么？

- Cython is an optimising static compiler for both the Python programming language and the extended Cython programming language (based on Pyrex).

Cython是什么？

- It makes writing C extensions for Python as easy as Python itself.
  - Cython can compile (most) regular Python code
  - It aims to become a superset of the [Python] language which gives it high-level, object-oriented, functional, and dynamic programming.

Cython有什么能力？

- 无痛提升运行效率。
- Python程序防逆向工程。
- 一点工作，数（百）倍运行效率。
- 把好用的Python库给C/C++用。
- 更方便、更简单地编写Python扩展。
- 封装已有的C/C++库。

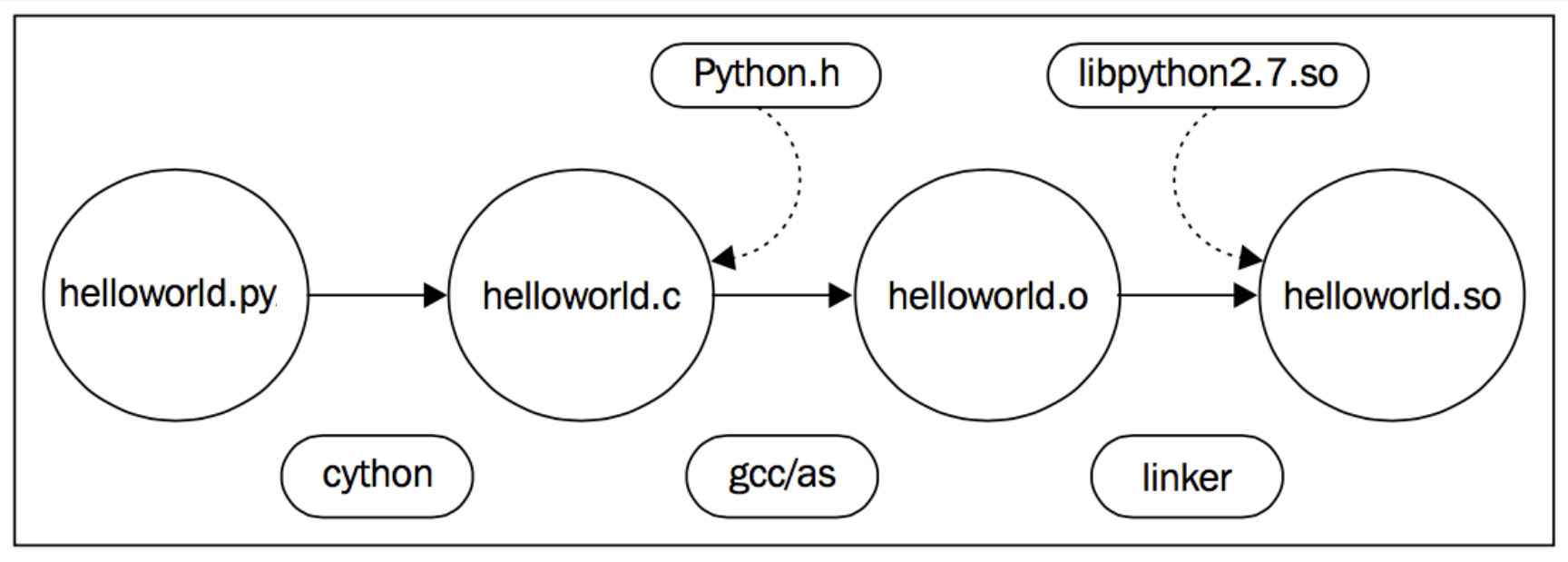
无痛提升运行效率。

```
def f(x):
    return x**2-x

def integrate_f(a,b,N):
    s = 0
    dx = (b-a)/N
    for i in range(N):
        s += f(a+i*dx)
    return s * dx
```

```
cython ex1.py

gcc -fPIC -shared -I/
usr/include/python2.7
ex1.c -o ex1.so
```

~35% times speedup

Python程序防逆向工程。



没有字节码，顺手防逆向。

一点工作，数（百）倍运行效率。

```
def f(double x):
    return x**2-x

def integrate_f(double a, double b, int N):
    cdef int i
    cdef double s, dx
    s = 0
    dx = (b-a)/N
    for i in range(N):
        s += f(a+i*dx)
    return s * dx
```

~4 times speedup

一点工作，数（百）倍运行效率。

```
def f(double x):
    return x**2-x

cdef double f(double x) except? -2:
    return x**2-x
```

~150 times speedup

把好用的Python库给C/C++用。

```
# pyurllib.pyx

import urllib

cdef public char * urlopen(char *
url):
    content=\
        urllib.urlopen(url).read()
    return content
```

把好用的Python库给C/C++用。

```
#include <Python.h>
#include "pyurllib.h"

int main (int argc, char **
argv)
{
  /* Boiler plate init Python
*/
  Py_SetProgramName (argv
[0]);
  Py_Initialize ();

  /* Init our url module into
Python memory */
  initpyurllib();
 if (argc >= 2)
    {
      /* call directly into
our cython module */

printf("%s",urlopen(argv[1]));
    }
  else
    printf ("require url...
\n");

  /* cleanup python before
exit ... */
  Py_Finalize ();

  return 0;
}
```

更方便、更简单地编写Python扩展。

```
// fic.c
#include “Python.h”
static PyObject* fib(...) {
// 解释参数
// C 类型的实参转变为Python对象
// 业务逻辑代码
// 构建返回值的Python对象
    return ret;
}
static PyMethodDef fib_methods[] = {
    {“fib”, fib, ...},
    {NULL,NULL,0,NULL},
};
void initfib(void) {
    (void) Py_InitModule(...);
}
```

```
#setup.py
module=Extension(‘fib’,
sources=[‘fib.c’])
setup(name=‘fib’,  version=‘1.0,
ext_modules=[module])

$ python setup.py build_ext –inplace

>>> import fib
>>> fib.fib(2000)
1 1 2 3 5 8 13 21 34 55 89 144 233
377 610 987 1597
```

before

更方便、更简单地编写Python扩展。

```
# fib.pyx
def fib(n):
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a + b

from distutils.core import
setup
from Cython.Build import
cythonize

setup(

ext_modules=cythonize("fib.pyx
"),
)
```

```
$ python setup.py build_ext –
inplace

>>> import fib
>>> fib.fib(2000)
1 1 2 3 5 8 13 21 34 55 89 144
233 377 610 987 1597
```

after

封装已有的C/C++库。

- 选择：ctypes/CFFI/boost.python/SWIG/cython
- 优势：
  - 易学，Python+C=Cython，重用旧知识
  - 好用，pxd，重用声明文件
  - C++支持全面，可从C++中回调Python函数，为Python class重载C++ class行为提供可能
  - ……

一点使用经验

- 经历了一个项目：封装C&C++编写的数据库驱动为Python模块。
  - 导出类型定义、宏与结构体
  - 导出函数、函数类型与复杂的类
  - 在Python类中使用C++类成员变量
  - 错误码与异常的转换
  - 编写一层C++代码简化接口

导出类型定义、宏与结构体

```
cdef extern from "ossTypes.h":
    ctypedef char CHAR

    ctypedef unsigned long long UINT64
    ctypedef long long SINT64
    ctypedef long long INT64

cdef extern from "ossErr.h":
    enum:SDB_OK
    enum:SDB_DMS_EOC
```

```
cdef extern from "msg.h":
    ctypedef struct MsgSysInfoRequest:
        pass
    ctypedef struct MsgSysInfoReply:
        pass
    ctypedef struct MsgOpReply:
        INT32 field_u_need
```

导出函数、函数类型与复杂的类

```
cdef extern from "_sdbapi.h" namespace "qc":
    SINT32 steal_size(const CHAR*)
    ctypedef void (*network_func)(CHAR* buff, size_t
size)

    cdef cppclass ConnectionContext:
        ConnectionContext() except +

        CHAR* send_buff
        INT32 send_buff_size
        network_func sender
        network_func recver

        string str()
```

在Python类中使用C++类成员变量

```
import socket
cdef class Connection(object):
    cdef ConnectionContext* _ctx
    cdef object _sock
    def __cinit__(self):
        self._ctx = new ConnectionContext()
        self._sock = socket.socket()
    def __dealloc__(self):
        del self._ctx
```

错误码与异常的转换

- 需要有统一的错误处理机制，C/C++ 的错误码方式需要转换为Python异常。

```
class Error(StandardError):
    def __init__(self, msg, err=None):
        self.err = err
        self.msg = msg

    def __str__(self):
        return 'Error(err=%s, msg="%s")' %
(self.err, self.msg)
```

编写一层C++代码简化接口

- 避免需要把复杂的类用Cython声明一遍
- 避免C++代码风格传导到Python
- 更一致的生命周期管理
- 编译器、平台相关的需求（如#pragma pack）
- 胶合层提供更高的灵活性

关于齐昌

客户


TONGCHIDAO


CODING
CLOUD DEVELOPMENT


wecom


猜


机智云
Gizwits


SequoiaDB

技术


python


ython


THE
C
PROGRAMMING
LANGUAGE


THE
C++
PROGRAMMING LANGUAGE


django


MySQL


ubuntu


Bootstrap

Q&A