Библиотеки
(часть 2)

Библиотека на Си, приложение на Python.

Предположим, что у нас есть динамическая библиотека со следующим набором функций:

- Простая (с точки зрения Python) функция add:

```
int add(int a, int b);
```

- Функция divide возвращает несколько значений, одно из которых возвращается с помощью указателя:

```
int divide(int a, int b, int *remainder);
```

Библиотека на Си, приложение на Python.

окончание

– Функции avg, fill_array, filter обрабатывают массив:

```
double avg(double *arr, int n);
void fill_array(double *arr, int n);
int filter(double *src, int src_len,
                        double *dst, int *dst_len);
```

Нам необходимо вызвать эти функции из программы на Python без написания какого-либо дополнительного кода на Си или использования каких-либо утилит.

ctypes (шаг 1)

Чтобы загрузить библиотеку необходимо создать объект класс CDLL:

```
import ctypes
lib = ctypes.CDLL('example.dll')
```

Классов для работы с библиотеками в модуле ctypes несколько:

- CDLL (cdecl и возвращаемое значение int);
- OleDLL (stdcall и возвращаемое значение HRESULT);
- WinDLL (stdcall и возвращаемое значение int).

Класс выбирается в зависимости от соглашения о вызовах, которое использует библиотека.

ctypes (шаг 2)

После загрузки библиотеки необходимо описать заголовки функций библиотеки, используя нотацию и типы известные Python.

```
# int add(int, int)
add = lib.add
add.argtypes = (ctypes.c_int, ctypes.c_int)
add.restype = ctypes.c_int
```

Чтобы интерпретатор Python смог правильно конвертировать аргументы, вызвать функцию add и вернуть результат ее работы, необходимо указать атрибуты argtypes и restype.

ctypes: не Python поведение (1)

В языке Си используются идиомы, которых нет в языке Python. Например, функция divided возвращает одно из значений через свой аргумент. Поэтому решение «в лоб» обречено на неудачу.

```
# int devide(int, int, int*)
_divide = lib.divide
_divide.argtypes = (ctypes.c_int, ctypes.c_int, \
                    ctypes.POINTER(ctypes.c_int))
_divide.restype = ctypes.c_int

x = 0
_divide(7, 5, x)
```

ctypes: не Python поведение (2)

Целые числа в Python «неизменяемые» объекты. Попытка их изменить вызовет исключение. Поэтому для аргументов, которые «используют» указатель, необходимо с помощью описанных в модуле ctypes совместимых типов создать объект и передать именно его.

```
def divide(x, y):
  rem = ctypes.c_int()
  quot = _divide(x, y, rem)
  return quot, rem.value
```

ctypes: массивы

Функция avg ожидает получить указатель на массив. Необходимо понять, какой тип данных Python будет использоваться (список, кортеж и т.п.) и как он преобразуется в массив.

```
# void avg(double*, int)
_avg = lib.avg
_avg.argtypes = (ctypes.POINTER(ctypes.c_double), \
                                        ctypes.c_int)
_avg.restype = ctypes.c_double

def avg(nums):
  src_len = len(nums)
  src = (ctypes.c_double * src_len)(*nums)
  return _avg(src, src_len)
```

ctypes: итоги

- Основная проблема использования этого модуля с большими библиотеками – написание большого количества сигнатур для функций и, в зависимости от сложности функций, функций-оберток.
- Необходимо детально представлять внутренне устройство типов Python и то, каким образом они могут быть преобразованы в типы Си.
- Альтернативные подходы – использование Swig или Cython.

Модуль расширения (1)

Полное и исчерпывающее описание алгоритма написания модуля расширения может быть найдено в документации Python:

“Extending and Embedding the Python Interpreter” (https://docs.python.org/3/extending/index.html).

Сейчас будут рассмотрены только наиболее важные моменты.

Модуль расширения (2)

Обычно функции модуля расширения имеют следующий вид

```
static PyObject* py_func(PyObject* self, PyObject* args)
{
    ...
}
```

- PyObject – это тип данных Си, представляющий любой объект Python.
- Функция модуля расширения получает кортеж таких объектов (args) и возвращает новый Python объект в качестве результата.
- Аргумент self не используется в простых функциях.

Модуль расширения (4)

- Функция PyArg_ParseTuple используется для конвертирования переменных из представления Python в представление Си.
- На вход эта функция принимает строку форматирования, которая описывает тип требуемого значения, и адреса переменных, в которые будут помещены значения.
- В ходе конвертации функция PyArg_ParseTuple выполняет различные проверки. Если что-то пошло не так, функция возвращает NULL.

```
int a, b;
if (!PyArg_ParseTuple(args,"ii", &a, &b))
    return NULL;
```

Модуль расширения (5)

- Функция Py_BuildValue используется для создания объектов Python из типов данных Си. Эта функция также получает строку форматирования с описанием желаемого типа.

```
int a, b, c;

if (!PyArg_ParseTuple(args,"ii", &a, &b))
    return NULL;

c = add(a, b);

return Py_BuildValue("i", c);
```

Модуль расширения (6)

- Ближе к концу модуля расширения располагаются таблица методов модуля PyMethodDef и структура PyModuleDef, которая описывает модуль в целом.
- В таблице PyMethodDef перечисляются
  - Си функции;
  - имена, используемые в Python;
  - флаги, используемые при вызове функции,
  - строки документации.
- Структура PyModuleDef используется для загрузки модуля.
- В самом конце модуля располагается функция инициализации модуля, которая практически всегда одинакова, за исключением своего имени.

Модуль расширения: компиляция

Для компиляции модуля используется Python-скрипт setup.py. Компиляция выполняется с помощью команды:

```
python setup.py build_ext --inplace
```

На Windows компиляция может сразу не заработать. Внимательно прочитать:

- https://github.com/valtron/llvm-stuff/wiki/Building-Python-3.4--extension-modules-with-MinGW
- https://stackoverflow.com/questions/34135280/valueerror-unknown-ms-compiler-version-1900