BAB IV

FURTHER EXPRESSION

Stiap kode yang dituliskan menggunakan bahasa yang dikompilasi seperti C, C++ atau Java dapat diintegrasikan ke skrip Python lainnya. Kode ini diagnggap sebagai ektensi.

Modul ekstensi Python tidak lebih dari sekedar perpustakaan C biasa. Pada mesin Unix, perpustakaan ini biasanya diakhiri dengan .so (untuk objek bersama). Pada mesin windows, biasanya melihat .dll (untuk perpusatkaan yang terhubung secara dinamis).

4.1 Pra-Persyaratan untuk Menulis Ekstensi

Untuk memulai ektensi, memerlukan file header Python. Pada mesin Unix, biasanya memerlukan instalasi paket khusus pengembang seperti python 2-5.

Pengguna window mendapatkan header ini sebagai bagian dari paket saat menggunakan pemasang Python biner.

Harus memiliki pengetahuan yang baik tentang C atau C++ untuk menulis ekstensi Python menggunakan pemrograman C.

Untuk melihat modul ekstensi Python, perlu mengelompokkan kode menjadi empat bagian :

- File header Python h
- Fungsi C yang ingin ditampilkan sebagai antarmuka dari modul
- Sebuah tabel memetakan nama-nama fungsi saat pengembang Python melihat ke fungsi C didalam modul ekstensi
- Fungsi inilisasi

Perlu menyertakan file header Python.h di file sumber C memberi akses ke API Python internal digunakan untuk menghitung modul ke penerjamah.

Menyertakan header Python.h sebelum header lain yang mungkin dibutuhkan. Mengikuti termasuk dengan fungsi yang ingin dipanggil dari Python.

Tanda tangan penerapan C fungsi selalu mengambil salah satu dari tiga bentuk berikut :

```
static PyObject *MyFunction( PyObject *self, PyObject *args );
 static PyObject *MyFunctionWithKeywords(PyObject *self,
                    PyObject *args,
                    PyObject *kw);
 static PyObject *MyFunctionWithNoArgs( PyObject *self );
```

Masing-masing deklarasi seelumnya mengembalikan objek Python. Tidak ada yang namanya fungsi void dengan Python seperti ada di C. Jika ingin fungsi mengembalikan nilai, Python. Header Python mendefinisikan makro. Py _Return _None yang melakukan ini.

Nama-nama fungsi C bisa menjadi apapun yang disuka karena tidak pernah diluar modul ekstensi mendefinisikan sebagai statis.

Fungi Cbiasanya diberi nama dengan menggabnungkan modul dan fungsi Python bersama-sama yang ditunjukan disini :

```
static PyObject *module _func(PyObject *self, PyObject *args) {
  /* Do your stuff here. */
  Py _RETURN _NONE;
}
```

Ini adalah fungsi Python yang disebut func didalam modul-modul. Memasukkan petunjuk ke fungsi C ke dalam tabel metode untuk modul yang biasanya muncul selanjutnya dikode sumber tael pemetaan metode.

Tabel metode ini adalah susunan sederhana dari struktur PyMethodSef. Struktur itu terlihat seperti ini :

```
struct PyMethodDef {
  char *ml _name;
  PyCFunction ml _meth;
  int ml _flags;
  char *ml _doc;
};
```

Inilai uraian anggota struktur ini :

- Ml _name

  Nama fungsi yang digunakan penafsir Python saat digunakan dalam program Python

- Ml _meth

  Menjadi alamaat ke fungsi yang memiliki salah satu tanda tangan yang dijelaskan dalam penelusuran sebelumnya

- Ml _flags

  Memberitahu penafsir yang mana dari tiga tanda tangan yang digunakan ml _meth. Bendera ini biasanya mmiliki nilai meth _varargs. Bendera ini dapat digandakan dengan or'ed dengan meth _keywords jika ingin memiarkan argumen kata kunci masuk ke fungsi. Ini juga bisa memiliki nilai meth _noargs yang menunjukan bahwa tidak ingin menerima argumen apa pun.

- Ml _doc

  Ini adalah docstring untuk fungsi yang bisa jadi NULL jika tidak ingin menulisnya.

  Tabel ini perlu diakhiri dengan sentinel yang terdiri dari NULL dan 0 untuk anggota yang sesuai.

  Contoh:

  ```
  static PyMethodDef module _methods[] = {
    { "func", (PyCFunction)module _func, METH _NOARGS, NULL },
  ```

```
  { NULL, NULL, 0, NULL }
};
```

Bagian terakhir dari modul ekstensi adalah fungsi inialisasi. Fungsi ini dipanggil oleh juru bahasa Python saat modul diisikan. Hal ini diperlukan agar fungsi diberi nama intiModule dimana modul adalah nama modul.

Fungsi inialisasi perlu diekspor dari perpustakaan yang akan dibangun. Header Python mendefinisikan PyMODINIT _Func untuk memasukkan mantra yang sesuai agar terjadi pada lingkungan tertentu tempat menyuusun. Yang harus dilakukan adalah mengunakan saat menentukan fungsinya.

Fungsi inialisasi C umumnya memiliki strktur keseluruhan berikut :

```
PyMODINIT _FUNC initModule() {
  Py _InitModule3(func, module _methods, "docstring...");
}
```

Berikut adalah penjelasan fugsi Py _IntiModule :

- Func

  Ini adalah fungsi yang akan diekspor

- Module

  Ini adalah nama tabel pemetaan yang didefinisikan diatas

- Docstring

Ini adalah komentar yang ingin diberikan diekstensi

Menempatkan ini semua bersama-sama terlihat sebagai berikut :

```
#include <Python.h>

static PyObject *module _func(PyObject *self, PyObject *args) {
  /* Do your stuff here. */
  Py _RETURN _NONE;
}

static PyMethodDef module _methods[] = {
  { "func", (PyCFunction)module _func, METH _NOARGS, NULL },
  { NULL, NULL, 0, NULL }
};

PyMODINIT _FUNC initModule() {
  Py _InitModule3(func, module _methods, "docstring...");
}
```

Contoh :

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

```
static PyObject* helloworld(PyObject* self)
{
   return Py _BuildValue("s", "Hello, Python extensions!!");
}

static char helloworld _docs[] =
   "helloworld( ): Any message you want to put here!! \n";

static PyMethodDef helloworld _funcs[] = {
   {"helloworld", (PyCFunction)helloworld,
    METH _NOARGS, helloworld _docs },
   {NULL }
};

void inithelloworld(void)
{
   Py _InitModule3("helloworld", helloworld _funcs,
                   "Extension module example!");
}
```

Disini fungsi Py _BuildValue digunakan untuk membangun nilai Python.

4.2 Membangun dan Menginstal Ekstensi

Distutils paket membuatnya sangat mudah mendistribusikan modul Python, baik Python murni dan modul ekstensi dengan cara standar. Modul didistribusikan dalam bentuk sumber dan dibangun dan diinstal melalui skrip setup yang iasa disebut seup.py sebagai berikut :

```
from distutils.core import setup, Extension
setup(name='helloworld', version='1.0',  \
    ext _modules=[Extension('helloworld', ['hello.c'])]).
```

Sekarang gunakan perintah berikut yang aka melalakukan semua kompilasi dan langkah penghubunh yang diperlukan dengan perintah dan bendera penyusun dan penghubung yang benar dan menyalin perpustakaan dinamis yang dihasilkan ke dalam direktori yang sesuai .

Contoh :

```
$ python setup.py install
```

Pada sistem berbasis Unix kemungkinan besar perlu menjalankan perintah ini sebagai root agar meminta izin untuk menulis ke direktori paket situs. Ini biasanya tidak menjadi masalah pada window.

Setelah menginstal ekstensi, akan dapat mengimpor dan memanggil ekstensi tersebut di skrip Python sebagai berikut :

```
#!/usr/bin/python
```

```
import helloworld

print helloworld.helloworld()
```

Ini akan menghasilkan hasil sebagai berikut :
Hello, Python extensions!!

Seperti kemungkinan besar ingin mendefinisikan fungsi yang menerima argumen, dapat menggunakan salah satu tanda tangan lain untuk fungsi C. Sebagai contoh, fungsi berikut yang menerima beberapa parameter akan didefinisikan seperti ini :

```
static PyObject *module _func(PyObject *self, PyObject *args) {
  /* Parse args and do something interesting here. */
  Py _RETURN _NONE;
}
```

Tabel metode yang berisi entri untuk fungsi baru akan terlihat seperti ini :

```
static PyMethodDef module _methods[] = {
  { "func", (PyCFunction)module _func, METH _NOARGS, NULL },
  { "func", module _func, METH _VARARGS, NULL },
  { NULL, NULL, 0, NULL }
};
```

Menggunakan fungsi API PyArg _ParseTuple untuk mengekstrak argumen dari satu pointer PyObject yang dikirimkan ke fungsi C. Argumen pertama untuk PyArg _ParseTuple adalah args argumen. Ini adalah objek yang akan parsing. Argumen kedua adalah string format yang menggambarkan argumen saat mengharapkannya muncul. Setiap argumen diwakili oleh satu atau lebih karakter dalam format string sebagai berikut :

```
static PyObject *module _func(PyObject *self, PyObject *args) {
  int i;
  double d;
  char *s;

  if (!PyArg _ParseTuple(args, "ids", &i, &d, &s)) {
    return NULL;
  }

  /* Do something interesting here. */
  Py _RETURN _NONE;
}
```

Mengkompilasi versi baru dari modul dan mengimpornya memungkinkan untuk memanggil fungsi baru dengan sejumlah argumen dari jenis apa pun :

```
module.func(1, s="three", d=2.0)
module.func(i=1, d=2.0, s="three")
module.func(s="three", d=2.0, i=1)
```

Berikut adalah tanda tangan standar untuk fungsi PyArg _ParseTuple:
int PyArg _ParseTuple(PyObject* tuple,char* format,...)

Fungsi ini mengembalikan 0 untuk kesalahan, dan nilai tidak sama dengan 0 untuk kesuksesan. Tuple adalah PyObject * yang merupakan argumen kedua dari fungsi C. Format berikut adalah string C yang menggambarkan argumen wajib dan opsional. Berikut

adalah daftar kode format untuk fungsi PyArg _ParseTuple:

Code	C type	Meaning
c	char	String Python dengan panjang 1 menjadi huruf C.
d	double	Pelampung Python menjadi C ganda.
f	float	Pelampung Python menjadi pelampung C.
i	int	Int Python menjadi int int
l	long	Sebuah int Python menjadi panjang C.
L	long long	Sebuah int Python menjadi C panjang panjang
O	PyObject*	Gets non-NULL meminjam referensi ke argumen Python.
s	char*	Python string tanpa nulls tertanam ke C char *.
s #	char*+int	Setiap string Python ke alamat dan panjang C.
t #	char*+int	Read-only penyangga segmen tunggal ke alamat C dan panjangnya.
u	Py _UNICODE*	Python Unicode tanpa nulls tertanam ke C.
u #	Py _UNICODE*+int	Setiap alamat dan panjang Python Unicode C.
w #	char*+int	Membaca / menulis penyangga segmen tunggal ke alamat dan panjang C.
z	char*	Seperti s, juga menerima None (set C char * ke NULL).
z #	char*+int	Seperti s #, juga menerima None (set C char * ke NULL).
(...)	as per ...	Urutan Python diperlakukan sebagai satu argumen per item.
|		Argumen berikut bersifat opsional.
:		Format akhir, diikuti dengan nama fungsi untuk pesan error.
;		Format akhir, diikuti oleh seluruh

Py _BuildValue mengambil format string seperti PyArg _ParseTuple. Alih-alih menyampaikan alamat nilai yang sedang bangun, melewati nilai sebenarnya. Berikut adalah contoh yang menunjukkan bagaimana menerapkan fungsi tambah :

```
static PyObject *foo _add(PyObject *self, PyObject *args) {
  int a;
  int b;

  if (!PyArg _ParseTuple(args, "ii", &a, &b)) {
    return NULL;
  }
  return Py _BuildValue("i", a + b);
}
```

Ini adalah apa yang akan terlihat seperti jika diimplementasikan dengan Python :

```
def add(a, b):
  return (a + b)
```

Mengembalikan dua nilai dari fungsi sebagai berikut, ini akan dipicu menggunakan daftar dengan Python :

```
static PyObject *foo _add _subtract(PyObject *self, PyObject *args) {
  int a;
  int b;

  if (!PyArg _ParseTuple(args, "ii", &a, &b)) {
    return NULL;
  }
  return Py _BuildValue("ii", a + b, a - b);
}
```

Ini adalah apa yang akan terlihat seperti jika diimplementasikan dengan Python :

```
def add _subtract(a, b):
  return (a + b, a - b)
```

Berikut adalah tanda tangan standar untuk fungsi Py _BuildValue :

```
PyObject* Py _BuildValue(char* format,...)
```

Format berikut adalah string C yang menggambarkan objek Python untuk dibangun. Argumentasi berikut Py _BuildValue adalah nilai C dari mana hasilnya dibuat. Hasil PyObject * adalah referensi baru.

Berikut daftar tabel string kode yang umum digunakan, yang nol atau lebihnya digabungkan ke dalam format string :

Code	Tipe C	Meaning
c	char	Sebuah char C menjadi string Python dengan panjang 1.
d	double	C ganda menjadi float Python.
f	float	Pelampung C menjadi float Python.
i	int	C int menjadi int Python.
l	long	Sebuah C panjang menjadi int Python.
N	PyObject*	Melewati objek Python dan mencuri referensi.
O	PyObject*	Melewati objek Python dan MENINGKATKANNYA seperti biasa.
O &	convert+void*	Konversi sewenang-wenang
s	char*	C 0-diakhiri char * ke string Python, atau NULL to None.
s #	char*+int	C char * dan panjang ke string Python, atau NULL to None.
u	Py _UNICODE*	String C-wide, null-terminated menjadi Python Unicode, atau NULL to None.
u #	Py _UNICODE*+int	String dan panjang lebar C ke Unicode Python, atau NULL to None.
w #	char*+int	Membaca / menulis penyangga segmen tunggal ke alamat dan panjang C.
z	char*	Seperti s, juga menerima None (set C char * ke

Kode {... } membangun kamus dari sejumlah nilai C, kunci dan nilai bergantian. Misalnya, Py _BuildValue (" {issi }", 23, "zig", "zag", 42) mengembalikan kamus seperti {23: 'zig', 'zag': 42 } Python.

Setiap blok memori yang dialokasikan dengan malloc () pada akhirnya harus dikembalikan ke genangan memori yang tersedia dengan satu panggilan untuk membebaskan (). Penting untuk menelepon gratis () pada waktu yang tepat. Jika alamat blok dilupakan tapi gratis () tidak dipanggil untuk itu, memori yang ditempatinya tidak dapat digunakan kembali sampai program berakhir. Ini disebut kebocoran memori. Di sisi lain, jika sebuah program memanggil gratis () untuk satu blok dan kemudian terus menggunakan blok tersebut, itu menciptakan konflik dengan penggunaan ulang blok melalui panggilan malloc () yang lain. Ini disebut dengan menggunakan memori yang dibebaskan. Ini memiliki konsekuensi buruk yang sama seperti merujuk pada data yang tidak diinisiasi - dump inti, hasil yang salah, crash misterius.

Karena Python membuat penggunaan malloc () dan gratis (), dibutuhkan strategi untuk menghindari kebocoran memori dan juga penggunaan memori yang bebas. Metode yang dipilih disebut penghitungan referensi. Prinsipnya sederhana: setiap objek berisi sebuah counter, yang bertambah saat referensi ke objek disimpan di suatu tempat, dan yang dikurangi saat referensi itu dihapus. Saat counter mencapai nol, referensi terakhir ke objek telah dihapus dan objeknya dibebaskan.