

Python 源码学习

KenshinLiu

Last Update: 2022 年 4 月 10 日

编译 Python

Listing 1: 编译使用的操作系统和 Python 版本信息

```
$ sw_vers
ProductName:      macOS
ProductVersion:  12.3
BuildVersion:     21E230

$ git branch
* main

$ git log -n 1
commit df9f7597559b6256924fcd3a1c3dc24cd5c5edaf (HEAD -> main, origin/main, origin/HEAD)
Author: Inada Naoki <songofacandy@gmail.com>
Date:   Tue Mar 1 10:27:20 2022 +0900
    compiler: Merge except_table and cnotab (GH-31614)
```

Listing 2: 编译命令

```
# debug 模式编译
$ ./configure --with-pydebug
$ make
```

Listing 3: doxygen 生成调用关系

```
$ doxygen -g doxygen.config
$ doxygen doxygen.config
```

Object

Python 所有对象的基础都是 Object. Object 在堆上进行分配。每个 Object 上都有一个表示引用计数的变量，当引用计数变为 0 之后，Object 会被回收。Object 上有一个表示类型的变量，每个 Object 的类型在 Object 被创建出来的时候就被固定，之后不会在改变。这个类型也是使用一个对象来进行表示，对象内部包含一个指针，指针指向被称为类型对象。这个类型对象包含一个表示类型的指针，指向自身。

Object 一旦分配之后，分配的内存大小以及内存地址都不会再发生改变。如果 Object 需要保存可变大小的数据，那么 Object 可以持有指向可变大小数据的一个指针。

Python 中的对象总是通过 PyObject* 类型的指针进行访问。PyObject 结构体只包含引用计数和一个类型指针。Python 中的对象的内存地址的前部都是一个 PyObject 结构体，通过这个 PyObject 结构体，就可以知道一个对象的引用计数和真实的类型。

Listing 4: PyObject 定义

```
struct _object {
    PyObject_HEAD_EXTRA // 这个字段用来编译一个 debug 版本，不用关注
    Py_ssize_t ob_refcnt;
    PyTypeObject *ob_type;
};
typedef struct _object PyObject;
```

PyObject 只能表示单个的对象，如果需要表示含有多个对象的结构，需要用到另一个结构体 PyVarObject。其中的 ob_size 字段表示对象的数量。

Listing 5: PyVarObject 定义

```
typedef struct {
    PyObject ob_base;
    Py_ssize_t ob_size; /* Number of items in variable part */
} PyVarObject;
```

上面提到了 Object 上有一个表示类型的变量。这个变量就是 PyObject 中的 ob_type 字段，类型为 PyTypeObject。Object 真正的类型就存储在这个结构体中。

Listing 6: PyTypeObject 的定义

```
struct _typeobject {
    PyObject_VAR_HEAD
    const char *tp_name; /* For printing, in format "<module>.<name>" */
    // ... ..
};

typedef struct _typeobject PyTypeObject;
```

定义中可以看到有一个 tp_name 字段，这个字段以字符串形式存储了类型名。

在定义的开始位置还可以看到一个 PyObject_VAR_HEAD，也就是上面提到的 PyVarObject，这说明 PyTypeObject 对象自身也有一个指针指向一个 PyTypeObject 类型的对象。那么这里就会有一个疑问：其他的 Object 的类型名存储在 PyTypeObject 的实例中，那么 PyTypeObject 实例自身的类型又是什么？存储在什么位置？要解答这个问题需要看一下 PyTypeObject 的实例化过程，下面以 Python 中最基础的类型类 class type 来举例子。

Listing 7: PyTypeObject 实例化

```
PyTypeObject PyType_Type = {
    PyVarObject_HEAD_INIT(&PyType_Type, 0) // 对象的指针指向自身
    "type",                                /* tp_name */
    // ... ..
};
```

PyType_Type 就是 class type 的实例，在实例化的代码中可以看到 PyType_Type 对象的 PyObject_VAR_HEAD 字段设置为了它自身，tp_name 字段设置为"type"。这样就成功的让类的继承体系中的每个类型都有了一个类型定义。

使用如下代码可以进行验证。

Listing 8: 修改 typeobject.c

```
PyTypeObject PyType_Type = {
    PyVarObject_HEAD_INIT(&PyType_Type, 0) // 对象的指针指向自身
    "PyTypeObject", // tp_name 从 "type" 修改为 "PyTypeObject"
    // ... ..
};
```

Listing 9: 创建 test.py

```
class A(object):
    pass

class B(A):
    pass

a = A()
b = B()

print(A.__class__)
print(B.__class__)

print(a.__class__)
print(b.__class__)
```

重新编译修改后的 Python，并使用生成的 Python 解释器执行 test.py，可以看到如下输出。

Listing 10: test.py 运行输出

```
# 修改 Python 源码前, 命令 python test.py 的输出
<class 'type'>
<class 'type'>
<class '__main__.A'>
<class '__main__.B'>

# 修改 Python 源码后, 命令 python test.py 的输出
<class 'PyTypeObject'>
<class 'PyTypeObject'>
<class '__main__.A'>
<class '__main__.B'>
```

从输出中可以看出, 类型 A 和类型 B 的类型都变成了修改后的 “PyTypeObject”, 但是对象 a 和对象 b 的类型不变。

再看一下 Python 中 int 类型的实例化。

Listing 11: int 类型类的实例化

```
PyTypeObject PyLong_Type = {
    PyVarObject_HEAD_INIT(&PyType_Type, 0) // class int 的类型也是 class type
    "int",                                     /* tp_name */
    offsetof(PyLongObject, ob_digit),         /* tp_basicsize */
    sizeof(digit),
}
```