《源码探秘 CPython》3. type 和 object 的恩怨纠葛

原创 古明地觉 古明地觉的编程教室 2022-01-04 09:30

楔子

type 和 object 两者的关系估计会让很多人感到困惑，我们说 type 站在**类型金字塔**的顶端，任何对象按照类型追根溯源，最终得到的都是 type；object站在**继承金字塔**的顶端，任何对象按照继承关系追根溯源，最终得到的都是 object。

因此我们可以得出以下结论：

- *type 的父类是 object*
- *object 的类型是 type*

```
print(type.__base__)  # <class 'object'>
print(object.__class__)  # <class 'type'>
```

打印的结果也说明了结论是正确的，但这就奇怪了，type 的父类是 object，可 object 类型又是 type，那么问题来了，是先有 type 还是先有 object 呢？

带着这些疑问，开始下面的内容。

类型对象的类型：PyType_Type

我们之前考察了 float 类型对象，知道它在 C 的层面是 PyFloat_Type 这个静态全局变量，它的类型是 type，包括我们自定义的类的类型也是 type。而 type 在 Python 中是一个至关重要的对象，它是所有类型对象的类型，我们称之为元类型(metaclass)，或者元类。借助元类型，我们可以实现很多神奇的高级操作。那么 type 在 C 的层面又长啥样呢？

在介绍 PyFloat_Type 的时候我们知道了 type 在底层对应 PyType_Type，而它在 "Object/typeobject.c" 中定义，因为我们说所有的类型对象加上元类都是要预先定义好的，所以在源码中就必须要以静态全局变量的形式出现。

```
PyTypeObject PyType_Type = {
    PyVarObject_HEAD_INIT(&PyType_Type, 0)
    "type",                                     /* tp_name */
    sizeof(PyHeapTypeObject),                   /* tp_basicsize */
    sizeof(PyMemberDef),                        /* tp_itemsize */
    (destructor)type_dealloc,                   /* tp_dealloc */

    // ...
    (reprfunc)type_repr,                        /* tp_repr */

    // ...
};
```

所有的类型对象加上元类都是 PyTypeObject 这个结构体实例化得到的，所以它们内部的成员都是一样的，只不过传入的值不同，实例化之后的结果也不同，可以是 PyLong_Type、可以是 PyFloat_Type，也可以是这里的 PyType_Type。

所以 PyType_Type 的内部成员和 PyFloat_Type 是一样的，但是我们还是要重点看一下里面的宏 PyVarObject_HEAD_INIT，我们看到它传递的是一个 &PyType_Type，说明它把自身的类型也设置成了 PyType_Type。换句话说，PyType_Type 里面的 ob_type 成员指向的还是 PyType_Type。

```
>>> type.__class__
<class 'type'>
>>> type.__class__.__class__.__class__.__class__.__class__ is type
True
>>> type(type(type(type(type(type())))) is type
True
>>>
```

显然不管我们套娃多少次，最终的结果都是True，显然这也是符合我们的预期的。

类型对象的基类：PyBaseObject_Type

我们说 Python 中有两个类型对象比较特殊，一个是站在类型金字塔顶端的 **type**，另一个是站在继承金字塔顶端的 **object**。说完了 type，我们再来说说 object。之前介绍类型对象的时候，我们说类型对象内部的 tp_base 表示继承的基类，那么对于PyType_Type来讲，它内部的 tp_base 肯定是**PyBaseObject_Type（object）**。

但令我们吃鲸的是，它的 tp_base 居然是个 0，如果为 0 的话则表示没有这个属性。

```
3637        Py_TPFLAGS_BASETYPE | Py_TPFLAGS_TYPE_SUBCLASS,         /* tp_flags */
3638    type_doc,                                   /* tp_doc */
3639    (traverseproc)type_traverse,                /* tp_traverse */
3640    (inquiry)type_clear,                        /* tp_clear */
3641    0,                                          /* tp_richcompare */
3642    offsetof(PyTypeObject, tp_weaklist),        /* tp_weaklistoffset */
3643    0,                                          /* tp_iter */
3644    0,                                          /* tp_iternext */
3645    type_methods,                               /* tp_methods */
3646    type_members,                               /* tp_members */
3647    type_getsets,                               /* tp_getset */
3648    0,                                          /* tp_base */
3649    0,                                          /* tp_dict */
3650    0,                                          /* tp_descr_get */
3651    0,                                          /* tp_descr_set */
```


不是说 type 的基类是 object 吗？
为啥 tp_base 是 0 呢，事实上如果你去看 PyFloat_Type 的话，会发现它内部的 tp_base 也是 0。
为 0 的原因就在于我们目前看到的类型对象是一个半成品，因为 Python 的动态性，显然不可能在定义的时候就将所有成员属性都设置好、然后解释器一启动就会得到我们平时使用的类型对象。
目前看到的类型对象是一个半成品，有一部分成员属性是在解释器启动之后再进行动态完善的。

> 至于是怎么完善的，都有哪些成员需要解释器启动之后才能完善，我们后续系列会说。

而 PyBaseObject_Type 位于 Object/object.c 中，我们来一睹其芳容。

```
PyTypeObject PyBaseObject_Type = {
    PyVarObject_HEAD_INIT(&PyType_Type, 0)
    "object",                                   /* tp_name */
    sizeof(PyObject),                           /* tp_basicsize */
    0,                                          /* tp_itemsize */
    object_dealloc,                             /* tp_dealloc */

    // ...
    object_repr,                                /* tp_repr */
    // ...
};
```

我们看到**PyBaseObject_Type**的类型也被设置成了**PyType_Type**，而**PyType_Type**类型对象在被完善之后，它的 tp_base 也会指向**PyBaseObject_Type**。所以之前我们说 Python 中的 type 和 object 是同时出现的，它们的定义是需要依赖彼此的。

```
>>> object.__class__
<class 'type'>
>>>
```

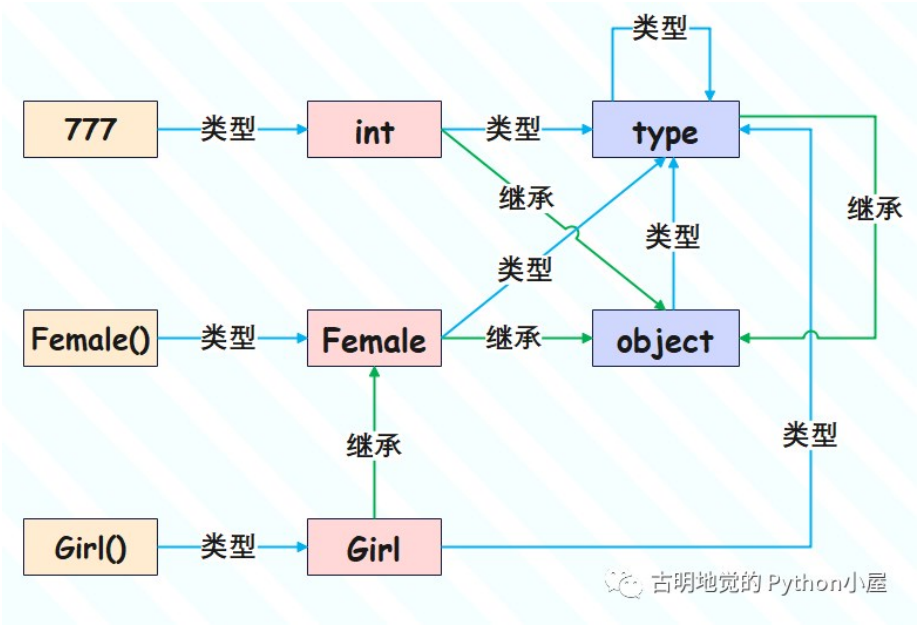
注意：解释器在完善 **PyBaseObject_Type** 的时候，是不会设置其 tp_base 成员的，因为继承链必须有一个终点，否则对象沿着继承链进行属性查找的时候就会陷入死循环，而 object 已经是继承链的顶点了。

```
>>> print(object.__base__)
None
>>>
```

- object -> PyBaseObject_Type
- object() -> PyBaseObject

小结

至此，我们算是从解释器的角度完全理清了 Python 中对象体系，其实我们之前画的图已经将 Python 对象体系表达的很清晰了，如下：



我们之前花了很大一部分笔墨来从 Python 的角度介绍其对象体系，之所以这么做就是为了能够更好地理解后续内容。如果能在 Python 层面上充分理解的话，那么在 CPython 层面上理解也就不难了。

当然啦，目前还远远没有结束，我们后续还会针对内建对象对象进行专门的剖析。

收录于合集 #CPython 97

‹ 上一篇
《源码探秘 CPython》4. 对象是怎么被创建的?

下一篇 ›
《源码探秘 CPython》2. 解密 PyObject、PyVarObject、PyTypeObject