Beijing Forest Studio
北京理工大学信息系统及安全对抗实验中心

Python对象探究

硕士研究生 柯懂湘

2018年07月08日

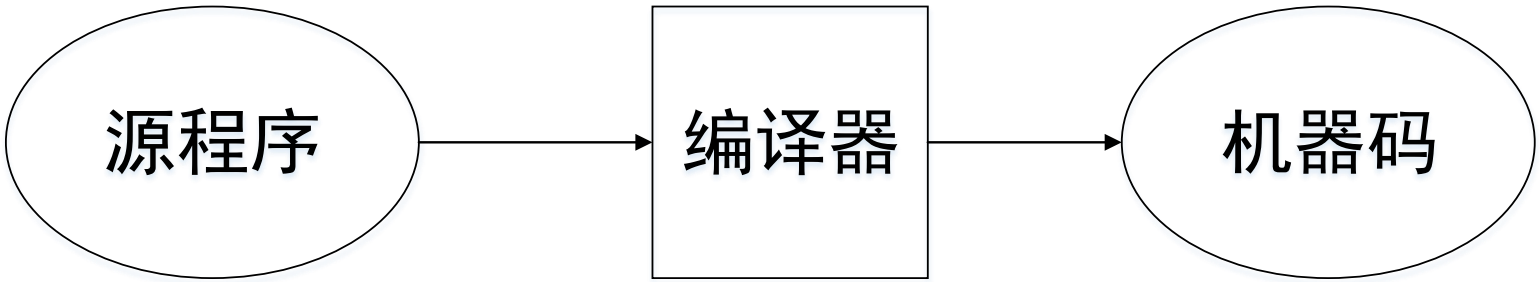
内容提要

- 背景知识
  - 编译型语言/解释型语言
  - 静态类型/动态类型
  - 类、对象、实例
- Python对象探究
  - Python对象分类
  - Python对象是什么样的
  - Python类型对象是什么样的
  - Python多态的实现
  - Python垃圾回收机制

背景知识

编译型语言

- 编译型语言
  - 编译型语言的首先将源代码编译生成**机器码**，再由**机器运行机器码**（二进制）。
  - 代表语言：C/C++



解释型语言

- 解释型语言
  - 由解释器来执行源代码
  - 代表语言：python、java
- 解释器执行源代码的方式
  - 解释器直接运行高级编程语言（shell）
  - 解释器首先将高级语言转换为字节码，然后解释器运行这些字节码(python)
  - 以解释器包含的编译器对高级语言编译，并指示处理器运行编译后的程序(JIT)

动态类型与静态类型

- 动态类型语言
  - 是指在运行期间才去做数据类型检查的语言。在用动态语言编程时，不用给变量指定数据类型，该语言会在你第一次赋值给变量时，在内部将数据类型记录下来。（python）

- 静态类型语言
  - 与动态类型语言刚好相反，它的数据类型检查发生在在编译阶段，也就是说在写程序时要声明变量的数据类型。（C/C++、JAVA）

类、对象、实例

- 类(class)
  - 表示某种类型
  - 包含属性和方法
- 对象(object)
  - 具有状态和行为的实体
- 实例(instance)
  - 对象相对于类的称呼
- 每个对象(Object)都是某个类(class)的一个实例(instance).

Python对象探究

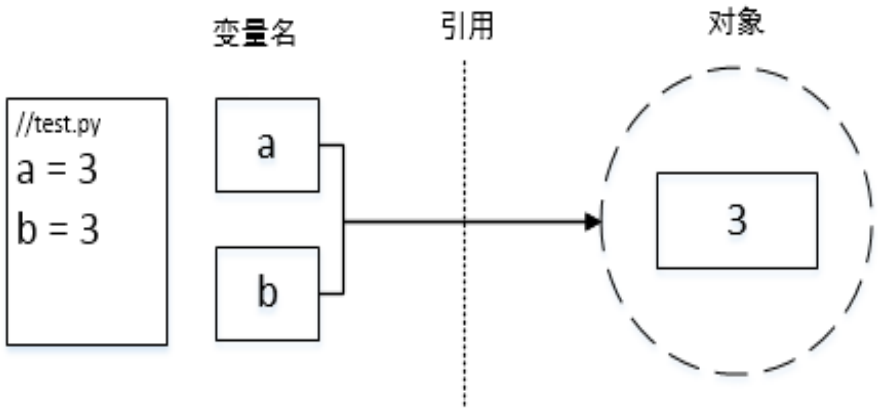
Python对象探究

- 提问：下面代码的输出结果是什么？

```
a = 3
b = 3
print(a is b)
print(a == b)
```

Python对象探究

- Python的动态类型机制
  - 变量名只是对象的一个引用
  - 变量名没有类型，只有对象才有类型

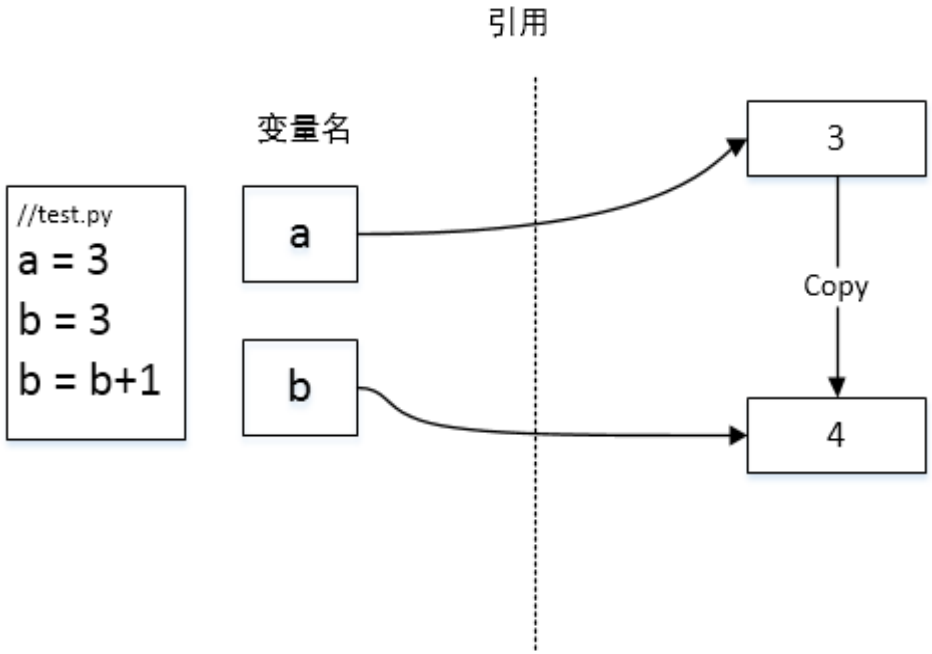


可变对象与不可变对象

- 不可变对象
  - 对象内容不可变
  - int、string
- 可变对象
  - 对象内容可变
  - dict、list

```
a = 3
b = 3
print(a is b)
print(a == b)
b += 1
print(a is b)
```

可变对象与不可变对象



- Python对象到底是什么样的？

Python实例对象

- PyObject
  - Python中所有东西都是对象，而所有对象都拥有一些相同的内容，这些内容在PyObject中定义。
  - PyObject是Python对象的核心。

```
typedef struct _object {
    int ob_refcnt;  //对象引用计数
    struct _typeobject *ob_type; //指向对象的类型对象
} PyObject;
```

Python实例对象

PyObject定义了每个python对象都必须有的内容，这些内容将出现在每个对象所占有的内存的最开始的字节中。这就意味着Python中的每一个对象都可以通过一个PyObject*指针引用。

```
//python int对象
typedef struct {
    int ob_refcnt;  //对象引用计数
    struct _typeobject *ob_type; //指向对象的类型对象
    long ob_ival;  //整形对象具体数值
} PyIntObject;
```

Python类型对象

- 一个PyTypeObject对象就是Python中对面向对象理论中“类”这个概念的实现。

```
typedef struct _typeobject {
    int ob_refcnt;
    struct _typeobject *ob_type;
    int ob_size;
    char tp_name; //类型名称
    int tp_basicsize, tp_itemsize;
    /*Method to implement standard operations*/
    destructor tp_dealloc;
    printfunc tp_print;
    /*more stanard operations*/
    hashfunc tp_hash;
    ternaryfunc tp_call;
    ......
} PyTypeObject
```

Python类型对象

- PyTypeObject中主要包含了四类信息
  - 类型名，主要是Python内部以及调试的时候使用
  - 创建该类型对象时分配的内存空间大小信息
  - 与该类型相关的操作信息
  - 类型的类型信息

对象行为

- PyTypeObject中定义了大量的函数指针，这些函数指针可视为类型对象所定义的操作，而这些操作直接决定着一个对象在运行时所表现的行为。
  - tp_new决定一个实例对象如何被创建
  - tp_init决定一个对象如何被初始化
  - tp_print决定一个对象如何被输出

对象行为

- 操作信息中有三组重要的操作族
  - PyNumberMethods定义一个数值对象支持的操作
  - PySequenceMethods定义一个序列对象支持的操作
  - PyMappingMethods定义了一个关联对象支持的操作
  - 对于一个类型来说，完全可以同时定义三类操作

```
a = "hello"
b = "world"
print(a + " " + b )
print(a[0:3])
```

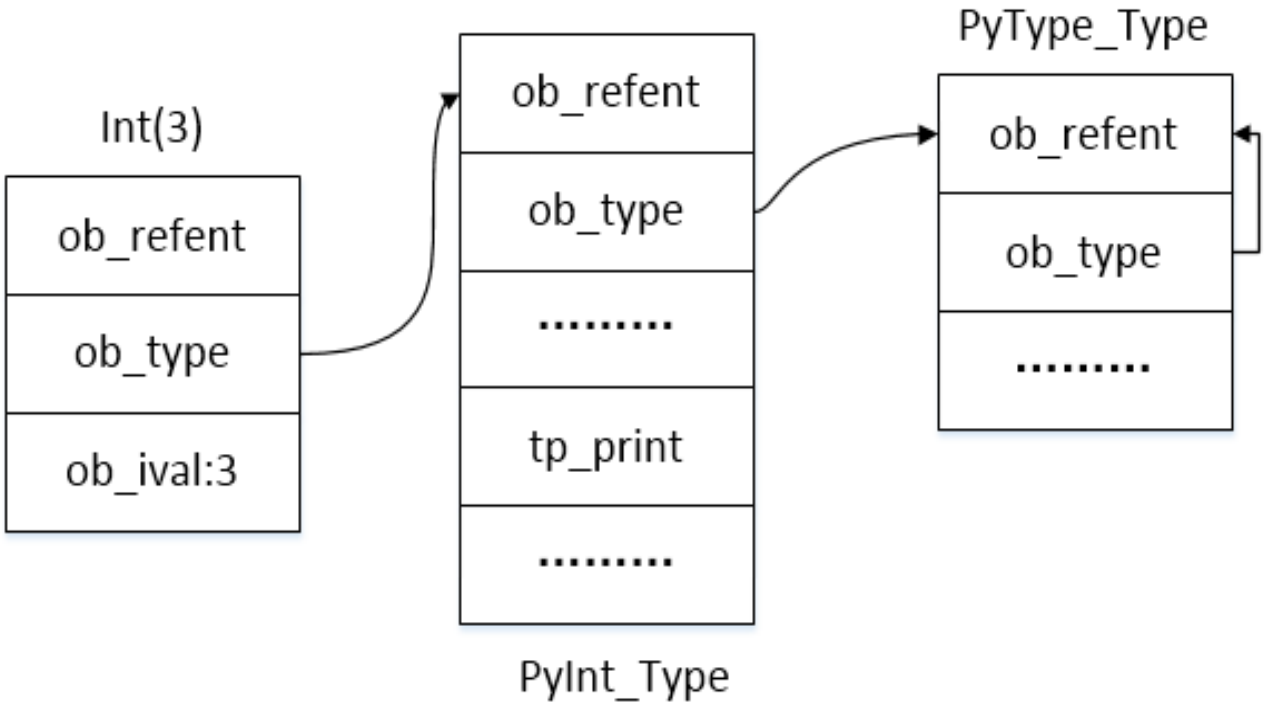
类型对象的类型

- 类型对象的类型是什么?
  - PyType_Type（python中的metaclass）
- PyType_Type的类型是什么?
  - PyType_Type

```
Type "help", "copyright", "credits" or "license" for more information.
>>> int.__class__
<class 'type'>
>>> type.__class__
<class 'type'>
>>>
```

int对象在内存中的表示

- Int(3)在内存中的表示



对象的多态

- 什么是多态
  - 为不同数据类型的实体提供统一的接口
- python中一切皆对象
  - 在python中创建一个对象，比如PyIntObject对象时，会分配内存，进行初始化，然后Python内部会用PyObject*变量而不是PyIntObject*来维护和保存这个对象。
  - 具体类型信息可以通过ob_type域获得
  - 类似于c++多态函数的参数类型为基类

print函数多态实现

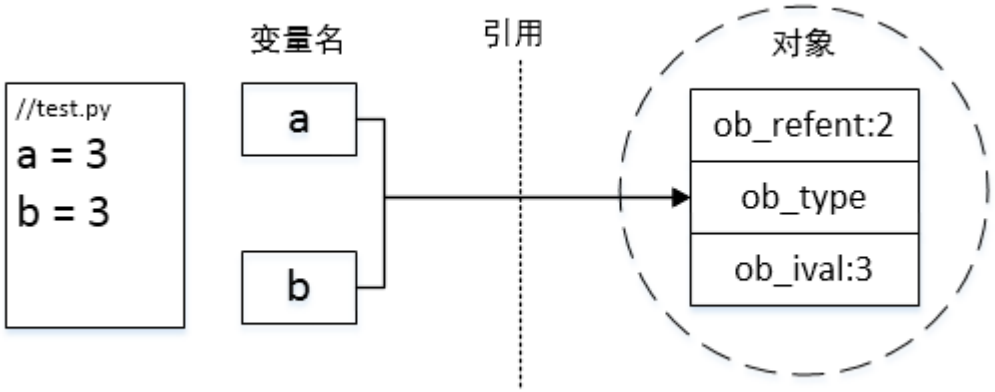
```
void print(PyObject *object)
{
    object->ob_type->tp_print(object);
}
```

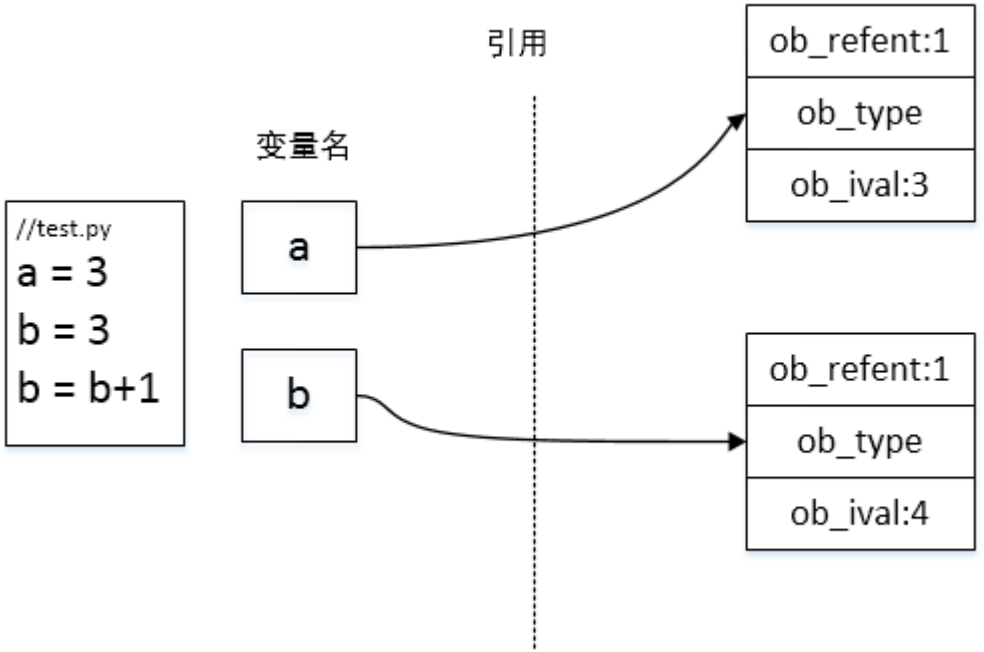
- 如果传递给print函数的指针为PyIntObject*，那么它会调用PyIntObject对象对应类型的输出操作。
- 如果传递给print函数的指针为PyStringObject*，那么它会调用PyStringObject对象对应类型的输出操作。

引用计数

- 垃圾回收机制
  - C/C++必须自己负责释放申请的内存，容易造成内存泄露等问题。
  - Pyton/Java，语言本身依靠垃圾回收机制负责内存的管理和维护。
  - Python通过对一个对象的引用计数管理来维护对象在内存的存在与否。
  - 当一个对象的引用计数为零，将调用tp_dealloc所指函数释放对象所占内存与系统资源

引用计数


变量名
引用
对象
//test.py
a = 3
b = 3
a
b
ob_refent:2
ob_type
ob_ival:3


引用
变量名
//test.py
a = 3
b = 3
b = b+1
a
b
ob_refent:1
ob_type
ob_ival:3
ob_refent:1
ob_type
ob_ival:4

Python面试题目

- 问题：解析下面的代码慢在哪？

```
def strtest1(num):
    str = "frist"
    for i in range(num):
        str += "X"
    return str
```

谢谢！

大成若缺，其用不弊。大盈若冲，其用不穷。大直若屈。大巧若拙。大辩若讷。静胜躁，寒胜热。清静为天下正。