《源码探秘 CPython》13. 整数在底层是如何进行大小比较的?

原创 古明地觉 古明地觉的编程教室 2022-01-18 09:30

收录于合集
#CPython
97个 >

楔子

整数溢出是程序开发中的一大难题，由此引发的 BUG 更是不计其数，而且相当隐蔽。之前使用Golang刷LeetCode的时候，怎么也通不过，发现是因为LeetCode后台有一个测试用例比较特殊，导致相加之后的整数太大，Golang的int64存不下。

而Python选择从语言层面彻底解决这个痛点，殚心竭虑地设计了整数对象。我们也探索了整数对象，并初步掌握了整数对象的内部结构。

Python的整数是串联了多个C语言的**digit**(uint32_t)，在底层通过一个数组来实现整数的存储。这么做的好处就是Python的整数没有长度限制了，因此不会溢出(而浮点数使用C的double，所以它会溢出)。

整数之所以不会溢出，是因为数组是没有长度限制的，所以只要你的内存足够，就可以算任意大的数。Python表示：存不下？会溢出？这都不是事儿，直接继续往数组里面塞digit就ok了。

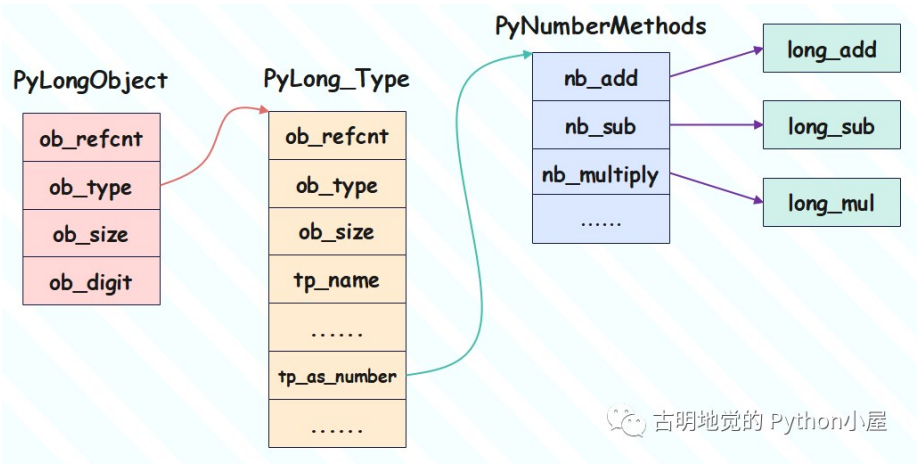
> 这里再重温一下PyLongObject的数据结构，我们说它是一个变长对象。ob_size指的是数组的长度，并且它除了表示长度之外，还能体现整数的正负，而ob_digit这个数组只用来存储其绝对值。

但是说实话，用数组实现大整数的思路其实平白无奇，并没有什么神秘的，就是将多个整数组合起来，模拟具有更高位的大整数。但这种实现方式的难点在于大整数**数学运算**的实现，它们才是重点，也是也比较考验编程功底的地方。

所以我们在分析浮点数的时候，一直说整数要比浮点数复杂，原因就在于此。比如加法运算，浮点数相加的话直接两个double相加即可，但是整数相加可就没有那么简单了，因为我们不能简单地将两个数组相加。

那么下面我们看一下整数在底层是怎么运算的，而运算逻辑在什么地方相信不用我说了。还是那句话，类型对象定义了哪些**操作**，决定了实例对象具有哪些**行为**。

根据浮点数的经验，我们知道PyLong_Type的tp_as_number也指向了PyNumberMethods结构体实例，里面的成员都是指向与整型运算相关的函数的指针。所以，我们直接去longobject.c中查看即可。



上图还是比较清晰的，如果想查看某个操作的底层实现，那么直接去对应的函数中查看即可。

整数的大小比较

在介绍运算之前，先来看看两个整数如何比较大小。

这里多说一句，本来是想把整数的**大小比较**以及**运算**写在一块的，但由于今天有点事情，回来的比较晚，一下子写不完，所以就分开写了。我们先来聊一聊整数之间的大小比较，至于运算我们就留到明天说吧。

首先整数在底层是通过数组存储的，ob_size 的绝对值维护了数组的长度。显然数组越长，整数的绝对值就越大，这是毫无疑问的。至于整数的正负，则由 ob_size 的正负来体现。

那么两个整数进行比较时：

- 如果ob_size均为正，显然 ob_size 越大，底层数组越长，对应的整数越大；
- 如果ob_size均为负，显然 ob_size 越大（越靠近0），底层数组越短，整数的绝对值越小。但因为是负数，还要乘上-1，因此整数反而会越大；
- 如果ob_size一正一负，这个应该就不必说了， ob_size 体现了整数的正负。正数肯定大于负数，所以 ob_size 大的那一方，对应的整数更大。

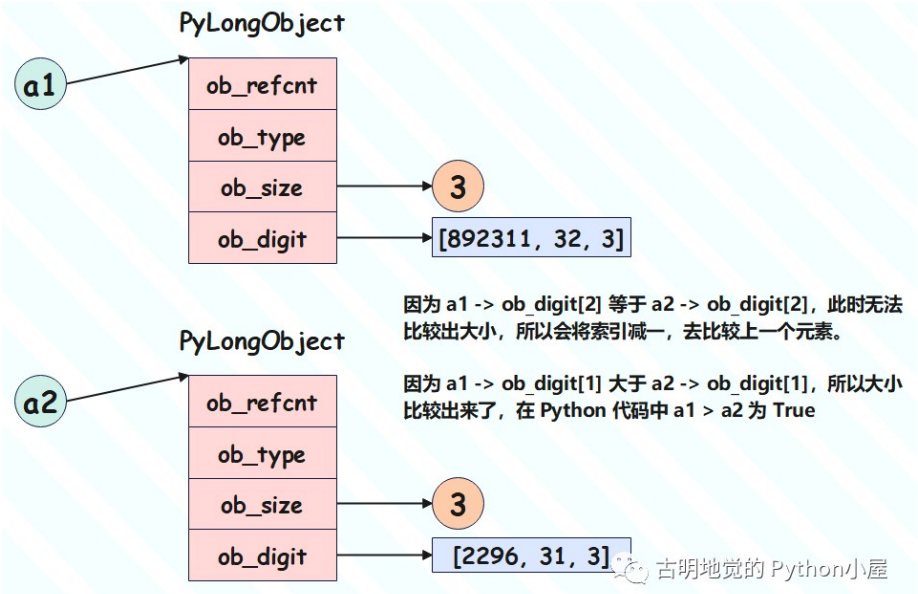
因此我们可以得出一个结论，当两个整数在比大小时，可以先比较各自的ob_size。如果ob_size不一样，那么可以直接比较出整数的大小，并且是ob_size 越大，整数越大。不管ob_size的正负如何，这一结论都是成立的，上面已经进行了验证。

但如果两个整数的ob_size一样，那么就从数组ob_digit的尾部元素开始，不断向前进行比较。只要两个整数的ob_digit中有一个对应元素不相同，那么就可以比较出大小。

之所以从数组的尾部开始，是因为数组元素的索引越大，那么充当的位数就越高，而在比较的时候显然要从高位开始比。

```
# ob_digit = [892311, 32, 3]
a1 = 3458764548181171607
# ob_digit = [2296, 31, 3]
a2 = 3458764547106539768
```

我们以 a1 和 a2 为例，显然 a1 大于 a2，那么在底层，它们是如何比较的呢？



当然啦，我们图中还少了一步，因为数组反映的是绝对值的大小。所以图中比较的是两个整数的绝对值，只不过正数和它的绝对值相同；

但如果是负数，那么绝对值越大，对应的整数反而越小，因此比较之后的结果还要再乘上-1。

最后，如果数组都遍历完了，发现相同索引对应的元素都是相同的，那么两个整数就是相等的。

下面我们就来看看源码是如何实现的？

```c
static int
long_compare(PyLongObject *a, PyLongObject *b)
{
    //sign是一个8字节的long，用来表示a和b之间的比较结果
    //如果a == b，那么sign = 0;
    //如果a > b，那么sign > 0;
    //如果a < b，那么sign < 0;
    Py_ssize_t sign;

    //Py_SIZE是一个宏:获取对象的ob_size
    //当然了，除了Py_SIZE，还有Py_REFCNT和Py_TYPE
    //用来获取对象的引用计数和类型指针
    if (Py_SIZE(a) != Py_SIZE(b)) {
    //这里是先比较两个整数的ob_size
    //如果ob_size不同，那么可以直接比较出大小
        sign = Py_SIZE(a) - Py_SIZE(b);
    //所以sign > 0的话，a > b
    //sign < 0的话，a < b
    //因为ob_size不一样， 所以sign不可能等于0
    }
    else {
        //如果相等， 那么说明a和b的符号相同， 数组长度也是一样的
        //那么接下来就只能挨个比较数组中的digit了
        //这里是获取数组的长度， 赋值给变量i
        //Py_ABS是一个宏，获取整数的绝对值
        Py_ssize_t i = Py_ABS(Py_SIZE(a));
        //进行while循环， i是数组的长度， 因此数组的最大索引是i-1，
        //所以这里的--i会先将i自减1,再判断自减1之后的i是否>=0
        while (--i >= 0 && a->ob_digit[i] == b->ob_digit[i])
        //然后比较a->ob_digit[i]和b->ob_digit[i]
        //如果数组内元素全部一样， 那么循环结束之后i肯定是-1
        //只要有一个不一样， 那么i一定>=0
            ;
        if (i < 0)
        //所以如果i < 0,说明两个整数的数组全部一样
        //因此两个整数是一样的，此时sign = 0
            sign = 0;
        else {
        //否则的话， 说明数组中索引为i的元素不一样
        //那么直接相减就可以了
            sign = (sdigit)a->ob_digit[i] - (sdigit)b->ob_digit[i];
        //如果sign大于0， 显然a对应的绝对值比b大
        //否则a对应的绝对值比b小
            if (Py_SIZE(a) < 0)
        //但我们说这比较的是绝对值
        //如果ob_size小于0，绝对值越大，整数反而越小
        //那么sign还要乘上-1
                sign = -sign;
        }
    }
    //因此最终: a > b则sign > 0, a < b则sign < 0, a == b则sign == 0
    //然后这里是一个嵌套的三元表达式
    //sign大于0则直接返回1，表示a > b
    //sign小于0返回-1，表示a < b
    //sign等于0则返回0，表示a == b
    return sign < 0 ? -1 : sign > 0 ? 1 : 0;
}
```

所以我们看到Python的整数就是按照上面这种方式比较的，总的来说就是先比较ob_size，ob_size不一样则可以直接比较出大小。如果ob_size一样的话，那么会从后往前挨个比较数组中的元素，最终确定大小关系。

小结

整数的实现并不简单，尽管设计的思路很好想，但难点在于它的数学运算，这是比较复杂并且考验编程功底的地方。

那么这次就先介绍到这里，下一次我们再来说一说整数的加减法运算。

收录于合集 #CPython 97

‹ 上一篇
《源码探秘 CPython》14. 整数在底层是如何运算的?

下一篇 ›
《源码探秘 CPython》12. 小整数对象池