《源码探秘 CPython》28. 列表支持的操作（中）

原创 古明地觉 古明地觉的编程教室 2022-02-14 09:00

index查询元素的索引

index方法可以接收一个元素，然后返回该元素首次出现的位置。当然还可以额外指定一个start和end，表示查询的范围。

```
static PyObject *
list_index_impl(PyListObject *self, PyObject *value, Py_ssize_t start,
                Py_ssize_t stop)
{
    Py_ssize_t i;

    //如果start小于0，加上长度
    //还小于0，那么等于0
    if (start < 0) {
        start += Py_SIZE(self);
        if (start < 0)
            start = 0;
    }
    if (stop < 0) {
        //如果stop小于0，加上长度
        //还小于0，那么等于0
        stop += Py_SIZE(self);
        if (stop < 0)
            stop = 0;
    }
    //从start开始循环
    for (i = start; i < stop && i < Py_SIZE(self); i++) {
        //获取相应元素
        PyObject *obj = self->ob_item[i];
        //增加引用计数，因为有指针指向它
        Py_INCREF(obj);
        //进行比较，PyObject_RichCompareBool是比较函数
        //接收三个参数：元素1、元素2、操作(这里显然是Py_EQ)
        //相等返回1，不相等返回0
        int cmp = PyObject_RichCompareBool(obj, value, Py_EQ);
        //比较完之后，减少引用计数
        Py_DECREF(obj);
        if (cmp > 0)
            //如果相等，返回其索引
            return PyLong_FromSsize_t(i);
        else if (cmp < 0)
            return NULL;
    }
    //循环走完一圈，发现都没有相等的
    //那么报错，提示元素不再列表中
    PyErr_Format(PyExc_ValueError, "%R is not in list", value);
    return NULL;
}
```

所以列表index方法的时间复杂度为O(n)，因为它在底层要循环整个列表，如果运气好，可能第一个元素就是；运气不好，就只能循环整个列表了。

同理后面要说的**if value in lst**这种方式也是一样的，因为都要循环整个列表，只不过后者返回的是一个布尔值。

count查询指定元素出现的次数

列表有一个 count 方法，可以计算出某个元素出现的次数。

```
static PyObject *
list_count(PyListObject *self, PyObject *value)
{
    //初始为0
    Py_ssize_t count = 0;
    Py_ssize_t i;

    //遍历每一个元素
    for (i = 0; i < Py_SIZE(self); i++) {
        //获取元素，和传入的value相比较
        PyObject *obj = self->ob_item[i];
        //如果相等，那么count自增1，继续下一次循环
        //注意这里的相等，它判断的是对象的地址
        //如果地址一样，那么看做是相等
        if (obj == value) {
           count++;
           continue;
        }
        Py_INCREF(obj);
        //走到这里说明地址不一样
        //但是地址不一样只能说明a is b不成立
        //并不代表a == b不成立
        //所以调用PyObject_RichCompareBool判断对象维护的值是否相等
        int cmp = PyObject_RichCompareBool(obj, value, Py_EQ);
        Py_DECREF(obj);
        //大于0，说明相等，count++
        if (cmp > 0)
            count++;
        else if (cmp < 0)
            return NULL;
    }
    //返回count
    return PyLong_FromSsize_t(count);
}
```

毫无疑问，count方法无论在什么情况下，它都是一个时间复杂度为O(n)的操作，因为列表必须要从头遍历到尾。

但是我们需要注意的是，里面判断相等的方式。因为变量只是一个指针，所以C里面的==相当于Python里面的**is**，Python 里面的==则对应`PyObject_RichCompareBool`这个函数。而源码里面在比较的时候先调用==，所以会先判断两者是不是同一个对象。

可能有人好奇我为什么说这些，举个栗子就明白了。

```
class A:

    def __eq__(self, other):
        return False


a = A()
lst = [a, a, a]
print(lst[0] == a)  # False
print(lst[1] == a)  # False
print(lst[2] == a)  # False

```

```
print(lst.count(a))  # 3
```

我们看到列表里面的三个元素和 a 都不相等，但是计算数量的时候，结果是 3。原因就是比较的时候是先比较地址，如果地址一样，那么认为元素相同。

而且上面的index方法也是如此，但问题是我们没有在里面看到**if (obj==value)**这行代码啊。事实上在`PyObject_RichCompareBool`这个函数里面已经包含了比较地址的逻辑，该函数会先比较地址是否一样，如果一样则认为相等，不一样再比较对象维护的值是否相等。

但是在 count 方法里面，将比较地址的逻辑又单独拿了出来，可以理解为快分支。但即使没有也无所谓，因为在`PyObject_RichCompareBool`里面还是会先对地址进行比较。

remove删除指定元素

除了根据索引删除元素之外，也可以根据值来删除元素，会删除第一个出现的元素。

```
static PyObject *
list_remove(PyListObject *self, PyObject *value)
{
    Py_ssize_t i;

    for (i = 0; i < Py_SIZE(self); i++) {
        //从头开始遍历，获取元素
        PyObject *obj = self->ob_item[i];
        Py_INCREF(obj);
        //比较是否相等，如果地址相同也算相等
        int cmp = PyObject_RichCompareBool(obj, value, Py_EQ);
        Py_DECREF(obj);
        //如果相等，那么进行删除
        if (cmp > 0) {
            //可以看到在删除元素的时候
            //还是调用了list_ass_slice
            if (list_ass_slice(self, i, i+1,
                               (PyObject *)NULL) == 0)
                //返回None
                Py_RETURN_NONE;
            return NULL;
        }
        else if (cmp < 0)
            return NULL;
    }
    //否则说明元素不在列表中
    PyErr_SetString(PyExc_ValueError, "list.remove(x): x not in list");
    return NULL;
}
```

以上就是remove函数的底层实现，说白了就是一层 for 循环，依次比较列表的每个元素和待删除元素是否相等。如果出现了相等的元素，则删除，然后直接返回，因为只删除一个；当整个循环遍历结束也没有发现满足条件的元素，那么报错，待删除元素不存在。

所以背后的逻辑并没有我们想象中的那么神秘。

reverse翻转列表

如果是你的话，你会怎么对列表进行翻转呢？显然是采用双指针，头指针指向列表的第一个元素，尾指针指向列表的最后一个元素，然后两两交换。

交换完毕之后，头指针后移一位、尾指针前移一位，继续交换。当两个指针相遇时，停止交换，而 Python 底层也是这么做的。

```c
static PyObject *
list_reverse_impl(PyListObject *self)
{
    //如果列表长度不大于1的话
    //那么什么也不做，直接返回None即可
    //Py_RETURN_NONE等价于return Py_None
    if (Py_SIZE(self) > 1)
        //大于1的话，执行reverse_slice， 传递了两个参数
        //第一个参数显然是底层数组首元素的地址
        //而第二个参数则是底层数组中索引为ob_size的元素的地址
        //但是很明显能访问的最大索引应该是ob_size - 1才对
        //别急我们继续往下看， 看一下reverse_slice函数的实现
        reverse_slice(self->ob_item, self->ob_item + Py_SIZE(self));
    Py_RETURN_NONE;
}


static void
reverse_slice(PyObject **lo, PyObject **hi)
{
    assert(lo && hi);

    //我们看到又执行了一次--hi，将hi移动到了ob_size - 1位置
    //也就是说此时二级指针hi指向的还是索引为ob_size - 1的元素
    //所以个人觉得有点纳闷
    //直接reverse_slice(self->ob_item, self->ob_item + Py_SIZE(self) - 1)
;不行吗
    --hi;
    //当lo小于hi的时候
    while (lo < hi) {
        PyObject *t = *lo;
        *lo = *hi;
        *hi = t;
        //上面三步就等价于 *lo, *hi = *hi, *lo
        //但是C不支持这么写
        //所以就是将索引为0的元素和索引为ob_size-1的元素进行了交换
        //然后两个指针继续靠近，指向的元素继续交换，直到两个指针相遇
        ++lo;
        --hi;
    }
}
```

所以到现在，你还认为Python的列表神秘吗？虽然我们很难自己写出一个Python解释器，但是底层的一些思想其实并没有那么难，作为一名程序猿很容易想的到。

小结

列表支持的操作还没有结束，我们还需要一篇才能介绍完。

收录于合集 #CPython 97

‹ 上一篇
《源码探秘 CPython》29. 列表支持的操作（下）

下一篇 ›
《源码探秘 CPython》27. 列表支持的操作（上）

文章已于2022-04-07修改

喜欢此内容的人还喜欢

一文剖析MySQL主从复制异常错误代码13114

TtrOpsStack

力扣 428. 序列化和反序列化 N 叉树 DFS

钰娘娘知识汇总

MySQL · 参数故事 · timed_mutexes

夜雨成诗