深入 Python 列表的内部实现

2015/10/27 · 基础知识 · 2 评论 · 列表

本文由 伯乐在线 – caimaoy 翻译，jasper 校稿。未经许可，禁止转载！
英文出处：Laurent Luce。欢迎加入翻译组。

本文将介绍列表在 CPython中的实现，因为毕竟Cpython 又是 Python 最为常用的实现。

Python 中的列表非常强大，看看它的内部实现机制是怎么样的，一定非常有趣。

下面是一段 Python 脚本，在列表中添加几个整数，然后打印列表。

```Python
>>> l = []
>>> l.append(1)
>>> l.append(2)
>>> l.append(3)
>>> l
[1, 2, 3]
>>> for e in l:
...   print e
...
1
2
3
```

可以发现，列表是一个迭代器。

Cpython 中的列表实现类似于下面的 C 结构体。ob_item 是指向列表对象的指针数组。allocated 是申请内存的槽的个数。

```Python
typedef struct {
    PyObject_VAR_HEAD
    PyObject **ob_item;
    Py_ssize_t allocated;
} PyListObject;
```

列表初始化

看看初始化一个空列表的时候发生了什么，例如：l = []。

```Python
arguments: size of the list = 0
returns: list object = []
PyListNew:
    nbytes = size * size of global Python object = 0
    allocate new list object
    allocate list of pointers (ob_item) of size nbytes = 0
    clear ob_item
    set list's allocated var to 0 = 0 slots
    return list object
```

要分清列表大小和分配的槽大小，这很重要。列表的大小和 len(l) 的大小相同。分配槽的大小是指已经在内存中分配了的槽空间数。通常分配的槽的大小要大于列表大小，这是为了避免每次列表添加元素的时候都调用分配内存的函数。下面会具体介绍。

Append 操作

向列表添加一个整数：l.append(1) 时发生了什么？调用了底层的 C 函数 app1()。

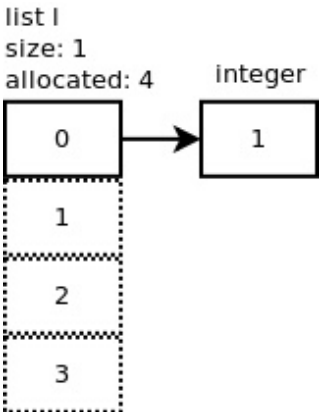
```Python
arguments: list object, new element
returns: 0 if OK, -1 if not
app1:
    n = size of list
    call list_resize() to resize the list to size n+1 = 0 + 1 = 1
    list[n] = list[0] = new element
    return 0
```

下面是 list_resize() 函数。它会多申请一些内存，避免频繁调用 list_resize() 函数。列表的增长模式为：0，4，8，16，25，35，46，58，72，88……

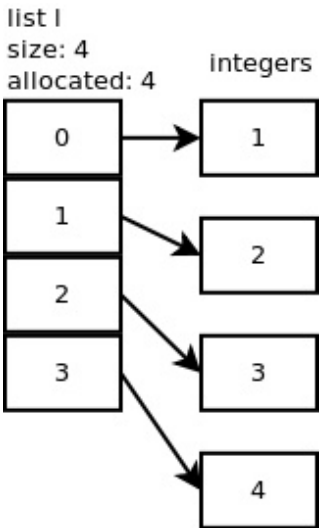
```Python
arguments: list object, new size
returns: 0 if OK, -1 if not
list_resize:
    new_allocated = (newsize >> 3) + (newsize < 9 ? 3 : 6) = 3
    new_allocated += newsize = 3 + 1 = 4
    resize ob_item (list of pointers) to size new_allocated
    return 0
```

添加的整数对象。虚线的方框表示已经分配但没有使用的槽空间。

列表追加元素操作的平均复杂度为 O(1)。



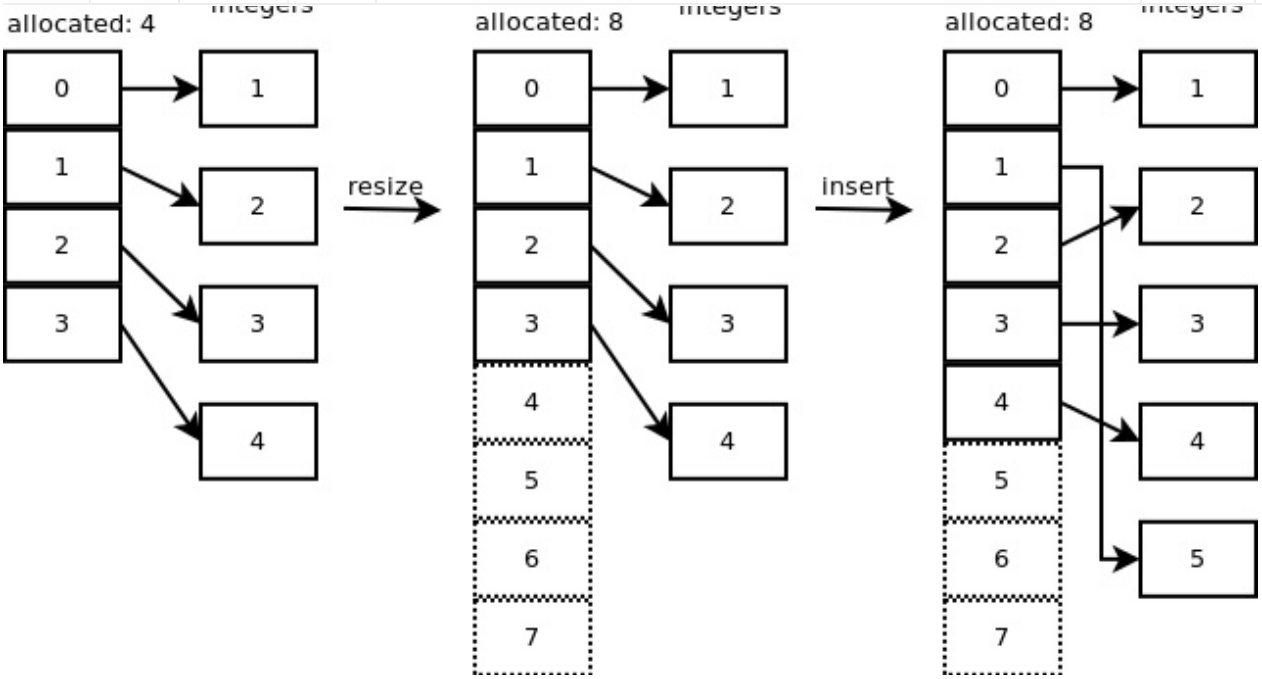
继续添加新的元素：l.append(2)。调用 list_resize 函数，参数为 n+1 = 2， 但是因为已经申请了 4 个槽空间，所以不需要再申请内存空间。再添加两个整数的情况也是一样的：l.append(3)，l.append(4)。下图显示了我们现在的情况。



Insert 操作

在列表偏移量 1 的位置插入新元素，整数 5：l.insert(1,5)，内部调用ins1() 函数。

```Python
arguments: list object, where, new element
returns: 0 if OK, -1 if not
ins1:
    resize list to size n+1 = 5 -> 4 more slots will be allocated
    starting at the last element up to the offset where, right shift each element
    set new element at offset where
    return 0
```



虚线的方框依旧表示已经分配但没有使用的槽空间。现在分配了 8 个槽空间，但是列表的大小却只是 5。

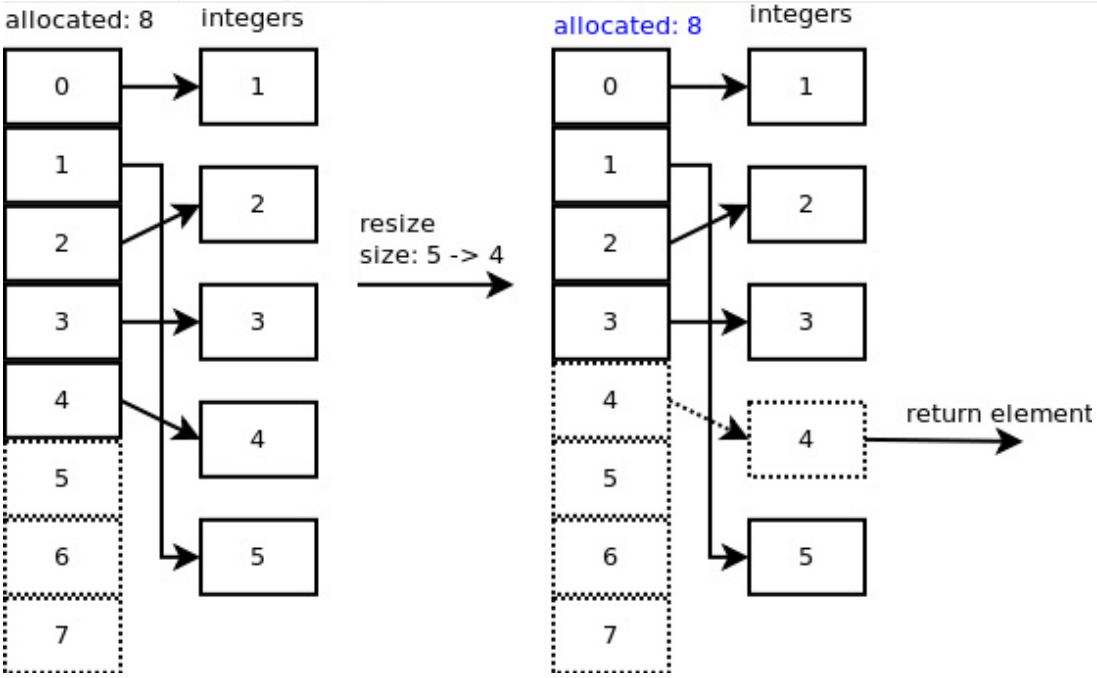
列表插入操作的平均复杂度为 O(n)。

Pop 操作

取出列表最后一个元素 即l.pop()，调用了 listpop() 函数。在 listpop() 函数中会调用 list_resize 函数，如果取出元素后列表的大小小于分配的槽空间数的一半，将会缩减列表的大小。

```Python
arguments: list object
returns: element popped
listpop:
    if list empty:
        return null
    resize list with size 5 - 1 = 4. 4 is not less than 8/2 so no shrinkage
    set list object size to 4
    return last element
```

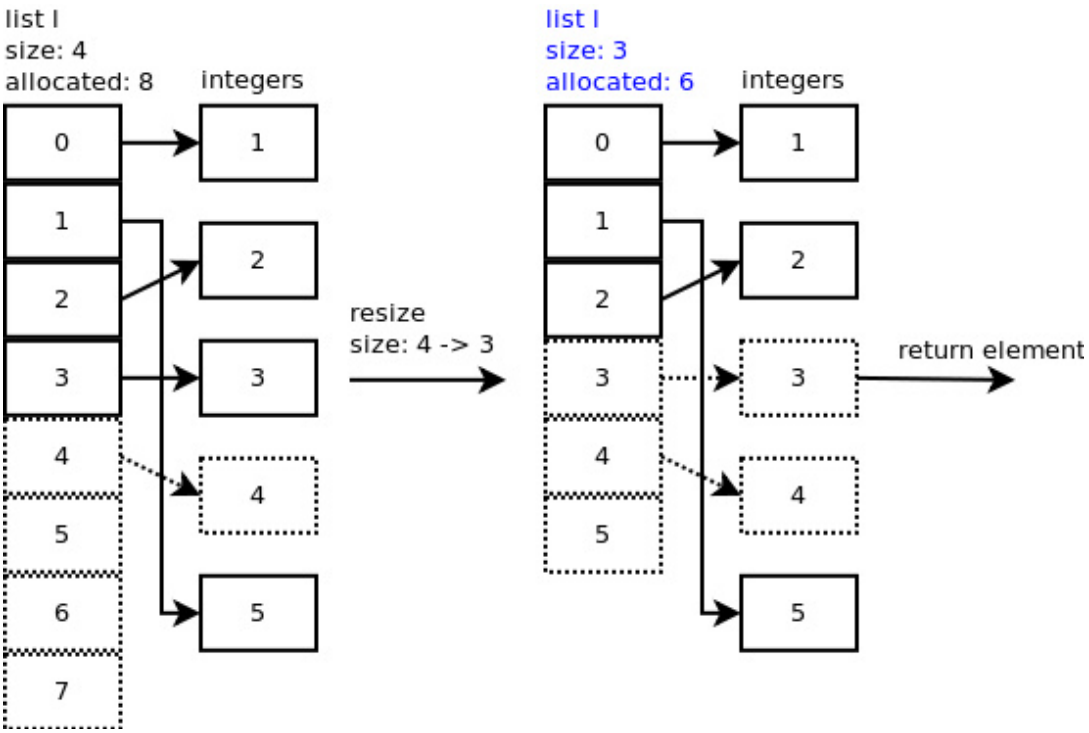
列表 pop 操作的平均复杂度为 O(1)。



可以看到 pop 操作后槽空间 4 依然指向原先的整数对象，但是最为关键的是现在列表的大小已经变为 4。

继续 pop 一个元素。在 list_resize() 函数中，size − 1 = 4 − 1 = 3 已经小于所分配的槽空间大小的一半，所以缩减分配的槽空间为 6，同时现在列表的大小为 3。

可以看到槽空间 3 和 4 依然指向原先的整数，但是现在列表的大小已经变为 3。



Remove 操作

Python 的列表对象有个方法，删除指定的元素： l.remove(5)。底层调用 listremove() 函数。

```
Python
```

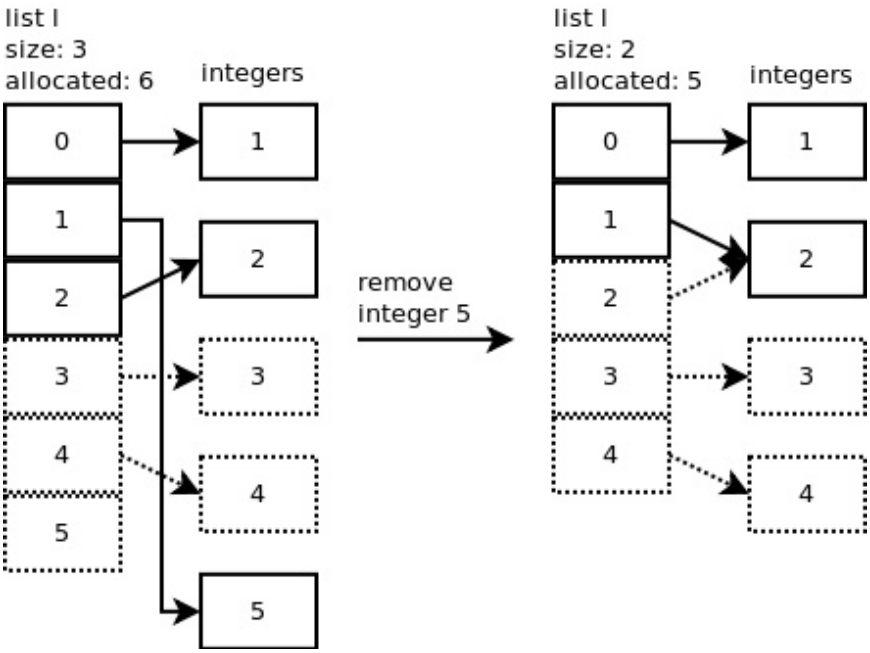
```
listremove:
    loop through each list element:
        if correct element:
            slice list between element's slot and element's slot + 1
            return none
    return null
```

为了做列表的切片并且删除元素，调用了 list_ass_slice() 函数，它的实现方法比较有趣。我们在删除列表位置 1 的元素 5 的时候，低位的偏移量为 1 同时高位的偏移量为 2.

Python

```
arguments: list object, low offset, high offset
returns: 0 if OK
list_ass_slice:
    copy integer 5 to recycle list to dereference it
    shift elements from slot 2 to slot 1
    resize list to 5 slots
    return 0
```

列表 remove 操作的复杂度为 O(n)。



打赏支持我翻译更多好文章，谢谢！

¥ 打赏译者

1 赞 5 收藏 2 评论

关于作者：caimaoy

简介还没来得及写 :)

个人主页 · 我的文章 · 13 ·