

微信扫一扫  
关注该公众号

收录于合集

#CPython

97个 &gt;

## 列表的创建

创建列表，Python底层只提供了唯一一个Python/C API，也就是PyList\_New。这个函数接收一个size参数，允许我们在创建一个PyListObject对象时指定底层的PyObject \* 数组的长度。

```
1 PyObject *
2 PyList_New(Py_ssize_t size)
3 {
4     //声明一个PyListObject *对象
5     PyListObject *op;
6 #ifdef SHOW_ALLOC_COUNT
7     static int initialized = 0;
8     if (!initialized) {
9         Py_AtExit(show_alloc);
10        initialized = 1;
11    }
12 #endif
13
14     //如果size小于0, 直接抛异常
15     if (size < 0) {
16         PyErr_BadInternalCall();
17         return NULL;
18     }
19     //缓存池是否可用, 如果可用
20     if (numfree) {
21         //将缓存池内对象个数减1
22         numfree--;
23         //从缓存池中获取
24         op = free_list[numfree];
25         //设置引用计数
26         _Py_NewReference((PyObject *)op);
27 #ifdef SHOW_ALLOC_COUNT
28         count_reuse++;
29 #endif
30     } else {
31         //不可用的时候, 申请内存
32         op = PyObject_GC_New(PyListObject, &PyList_Type);
33         if (op == NULL)
34             return NULL;
35 #ifdef SHOW_ALLOC_COUNT
36         count_alloc++;
37 #endif
38     }
39     //如果size等于0, ob_item设置为NULL
40     if (size <= 0)
41         op->ob_item = NULL;
42     else {
43         //否则的话, 创建一个指定容量的指针数组, 然后让ob_item指向它
44         //所以是先创建PyListObject对象, 然后创建指针数组
45         //最后通过ob_item建立联系
46         op->ob_item = (PyObject **) PyMem_Calloc(size, sizeof(PyObject *)
47 );
48         if (op->ob_item == NULL) {
49             Py_DECREF(op);
```

```

50         return PyErr_NoMemory();
51     }
52 }
53 //设置ob_size和allocated, 然后返回op
54 Py_SIZE(op) = size;
55 op->allocated = size;
56 _PyObject_GC_TRACK(op);
57 return (PyObject *) op;
58 }

```

我们注意到源码里面有一个缓存池，是的，Python大部分对象都有自己的缓存池，只不过实现的方式不同。

## 列表的销毁

创建PyListObject对象时，会先检测缓存池free\_list里面是否有可用的对象，有的话直接拿来用，否则通过malloc在系统堆上申请。列表的缓存池是使用数组实现的，里面最多维护80个PyListObject对象。

```

1  #ifndef PyList_MAXFREELIST
2  #define PyList_MAXFREELIST 80
3  #endif
4  static PyListObject *free_list[PyList_MAXFREELIST];

```

根据之前的经验我们知道，既然创建的时候能从缓存池中获取，那么在执行析构函数的时候也要把列表放到缓存池里面。

```

1  static void
2  list_dealloc(PyListObject *op)
3  {
4      Py_ssize_t i;
5      PyObject_GC_UnTrack(op);
6      Py_TRASHCAN_SAFE_BEGIN(op)
7      //先释放底层数组
8      if (op->ob_item != NULL) {
9          i = Py_SIZE(op);
10         //但是释放之前, 还有一件重要的事情
11         //要将底层数组中每个指针指向的对象的引用计数都减去1
12         //因为它们不再持有对"对象"的引用
13         while (--i >= 0) {
14             Py_XDECREF(op->ob_item[i]);
15         }
16         //然后释放底层数组所占的内存
17         PyMem_FREE(op->ob_item);
18     }
19     //判断缓存池里面PyListObject对象的个数, 如果没满, 就添加到缓存池
20     //注意: 我们看到执行到这一步的时候, 底层数组已经被释放掉了
21     if (numfree < PyList_MAXFREELIST && PyList_CheckExact(op))
22         //添加到缓存池的时候, 是添加到尾部
23         //获取的时候也是从尾部获取
24         free_list[numfree++] = op;
25     else
26         //否则的话就释放掉PyListObject对象所占的内存
27         Py_TYPE(op)->tp_free((PyObject *)op);
28     Py_TRASHCAN_SAFE_END(op)
29 }

```

我们知道在创建一个新的PyListObject对象时，实际上是分为两步的，先创建PyListObject对象，然后创建底层数组，最后让PyListObject对象中的ob\_item成员指向这个底层数组。

同理，在销毁一个PyListObject对象时，先销毁ob\_item维护的底层数组，然后再释放PyListObject对象自身(如果缓存池已满)。

现在可以很清晰地明白了，原本空荡荡的缓存池其实是被已经死去的PyListObject对象填充了。在以后创建新的PyListObject对象时，Python会首先唤醒这些死去的PyListObject对象，给它们一个洗心革面、重新做人的机会。但需要注意的是，这里缓存的仅仅是PyListObject对象，对于底层数组，其ob\_item已经不再指向了。

从list\_dealloc中我们看到，PyListObject对象在放进缓存池之前，ob\_item指向的数组就已经被释放掉了，同时数组中指针指向的对象的引用计数会减1。所以最终数组中这些指针指向的对象也大难临头各自飞了，或生存、或毁灭，总之此时和PyListObject之间已经没有任何联系了。

但是为什么要这么做呢？为什么不连底层数组也一起维护呢？可以想一下，如果继续维护的话，数组中指针指向的对象永远不会被释放，那么很可能会产生悬空指针的问题，所以这些指针指向的对象所占的空间必须交还给系统(前提是没有其它指针指向了)。

但是实际上，是可以将PyListObject对象维护的底层数组进行保留的，即：只将数组中指针指向的对象的引用计数减1，然后将数组中的指针都设置为NULL，不再指向之前的对象了，但是并不释放底层数组本身所占用的内存空间。

因此这样一来，释放的内存不会交给系统堆，那么再次分配的时候，速度会快很多。但是这样带来一个问题，就是这些内存没人用也会一直占着，并且只能供PyListObject对象的ob\_item指向的底层数组使用。因此Python还是为避免消耗过多内存，采取将底层数组所占的内存交还给了系统堆这样的做法，在时间和空间上选择了空间。

```
1 lst1 = [1, 2, 3]
2 print(id(lst1)) # 1243303086208
3 # 扔到缓存池中, 放在数组的尾部
4 del lst1
5
6 # 从缓存池中获取, 也会从数组的尾部开始拿
7 lst2 = [1, 2, 3]
8 print(id(lst2)) # 1243303086208
9
10 # 因此打印的地址是一样的
```

## 小结

关于列表，我们这里就介绍完了，内容还是蛮多的。作为一个功能强大的数据结构，多花些时间是有必要的。

收录于合集 #CPython 97

< 上一篇

《源码探秘 CPython》31. 元组是怎么实现的？

下一篇 >

《源码探秘 CPython》29. 列表支持的操作（下）

喜欢此内容的人还喜欢

MySQL全面瓦解28：分库分表架构与思维



undo log与red log以及mysql的锁



HelloNull



