

《源码探秘 CPython》17. bytes 对象的行为（下）

原创 古明地觉 古明地觉的编程教室 2022-01-26 09:30



微信扫一扫
关注该公众号

收录于合集

#CPython

97个 >

承接上篇文章，我们来介绍**bytes 对象的行为**下半部分。

将序列重复多次：

bytes对象可以通过类似于整数的乘法运算，来将自身重复多次，举个栗子：

```
1 >>> a = b"abc"
2 >>> a * 3
3 b'abcbcbcb'
4 >>> a * -1
5 b''
6 >>>
```

如果乘上一个负数，等于乘上0，那么会得到一个空的字节序列。但是这个函数对应的不是PyNumberMethods里面的np_multiply，而是PySequenceMethods里面的sq_repeat。

```
1 static PyObject *
2 bytes_repeat(PyBytesObject *a, Py_ssize_t n)
3 {
4     Py_ssize_t i;
5     Py_ssize_t j;
6     Py_ssize_t size;
7     PyBytesObject *op;
8     size_t nbytes;
9     //如果n小于0，那么等于0
10    if (n < 0)
11        n = 0;
12    //这里条件写成Py_SIZE(a) * n > PY_SSIZE_T_MAX更容易理解
13    //但我们说，这种方式可以避免溢出
14    if (n > 0 && Py_SIZE(a) > PY_SSIZE_T_MAX / n) {
15        //先计算相乘之后字节序列的长度是否超过最大限制
16        //如果超过了，直接报错
17        PyErr_SetString(PyExc_OverflowError,
18            "repeated bytes are too long");
19        return NULL;
20    }
21    //计算Py_SIZE(a) * n得到size
22    size = Py_SIZE(a) * n;
23    //快分支
24    if (size == Py_SIZE(a) && PyBytes_CheckExact(a)) {
25        //如果两者相等，那么证明n = 1
26        //本质上和原来没有变化，因此直接返回本身即可
27        //但是多了一个引用，所以返回之前引用计数加1
28        Py_INCREF(a);
29        return (PyObject *)a;
30    }
31    //类型转化，此时是size_t类型，相当于无符号64位整型
32    //nbytes就是ob_sval中有效字符占的总大小，不包括 \0
33    nbytes = (size_t)size;
34    //PyBytesObject_SIZE是一个宏
35    //等价于(offsetof(PyBytesObject, ob_sval) + 1)，
36    //计算的是PyBytesObject的ob_sval之前的所有成员的总大小、再加 1
37    //或者理解为从第一个成员到ob_sval成员之间的偏移量、再加 1
38    //所以nbytes + PyBytesObject_SIZE就是bytes对象所需要的空间
```

```

39 //如果nbytes + PyBytesObject_SIZE还小于等于nbytes
40 //证明长度发生溢出, 当然, 这个判断条件可能乍一看有点怪异
41 if (nbytes + PyBytesObject_SIZE <= nbytes) {
42     PyErr_SetString(PyExc_OverflowError,
43         "repeated bytes are too long");
44     return NULL;
45 }
46 //申请空间, 大小为PyBytesObject_SIZE + nbytes
47 op = (PyBytesObject *)PyObject_MALLOC(PyBytesObject_SIZE + nbytes);
48 if (op == NULL)
49     //返回NULL, 表示申请失败
50     return PyErr_NoMemory();
51 //PyObject_INIT_VAR是一个宏, 设置ob_type和ob_size
52 (void)PyObject_INIT_VAR(op, &PyBytes_Type, size);
53 //设置ob_shash为-1
54 op->ob_shash = -1;
55 //将ob_sval最后一位设置为'\0'
56 op->ob_sval[size] = '\0';
57 if (Py_SIZE(a) == 1 && n > 0) {
58     //显然这里是在a对应的bytes对象长度为1时, 所走的逻辑
59     //也就是重复的bytes对象的长度为1时, 会走这个快分支
60     //直接将op->ob_sval里面n个元素设置为a->ob_sval[0]即可
61     memset(op->ob_sval, a->ob_sval[0], n);
62     return (PyObject *) op;
63 }
64 i = 0;
65 //否则将a -> ob_sval拷贝到op -> ob_sval中
66 //拷贝n次, 因为size = Py_SIZE(a) * n;
67 //这里是先拷贝了一次
68 if (i < size) {
69     memcpy(op->ob_sval, a->ob_sval, Py_SIZE(a));
70     i = Py_SIZE(a);
71 }
72 //然后拷贝n - 1次
73 while (i < size) {
74     j = (i <= size-i) ? i : size-i;
75     memcpy(op->ob_sval+i, op->ob_sval, j);
76     i += j;
77 }
78 return (PyObject *) op;
79 }

```

以上就是重复一段序列对应的逻辑, 假设 **b1=b"abc"**, **b2=b1*3**, 那么会根据b1的长度创建对应ob_sval长度为10的bytes对象, 然后将b1的ob_sval拷贝三次。

但是上面有一点需要注意的是, 就是当重复之后的序列和原序列相同时, 那么不会创建新的bytes对象, 只需给原来的bytes对象的引用计数增加1即可。

```

1 >>> b1 = b"abc"
2 >>> b2 = b1 * 1
3 >>> id(b1) == id(b2)
4 True

```

根据索引或切片获取指定元素：

根据索引获取元素会得到一个整数, 根据切片获取元素得到的还是bytes对象。

```

1 >>> val = b"abcdef"
2 >>> val[1], type(val[1])
3 (98, <class 'int'>)
4 >>>
5 >>> val[1:4], type(val[1:4])

```

```
6 (b'bcd', <class 'bytes'>)
7 >>>
```

我们说PySequenceMethods里面的sq_item是根据索引获取指定元素，但底层真正执行的时候调用的却不是这个函数。调用的函数在PyMappingMethods里面，我们看一下bytes对象都实现了这个方法簇里面的哪些成员。

```
1 static PyMappingMethods bytes_as_mapping = {
2     (lenfunc)bytes_length,
3     (binaryfunc)bytes_subscript,
4     0,
5 };
```

我们看到映射操作，bytes对象中只有两个，一个bytes_length获取长度，这个在bytes_as_sequence中已经实现了，还有一个就是bytes_subscript进行切片操作。

因为映射操作只有两个，所以我们就放在这里介绍了。

```
1 static PyObject*
2 bytes_subscript(PyBytesObject* self, PyObject* item)
3 {
4     //参数是self和item, 那么在Python的层面上就类似于self[item]
5     //检测item, 看它是不是一个整型
6     if (PyIndex_Check(item)) {
7         //如果是, 转成 ssize_t
8         Py_ssize_t i = PyNumber_AsSsize_t(item, PyExc_IndexError);
9         if (i == -1 && PyErr_Occurred())
10             return NULL;
11         //如果i小于0, 那么将i加上序列的长度, 得到正数索引
12         if (i < 0)
13             i += PyBytes_GET_SIZE(self);
14         //如果i还是小于0, 或者本身大于等于序列的长度
15         //那么抛出IndexError
16         if (i < 0 || i >= PyBytes_GET_SIZE(self)) {
17             PyErr_SetString(PyExc_IndexError,
18                             "index out of range");
19             return NULL;
20         }
21         //获取之后, 将其转成整数返回
22         return PyLong_FromLong((unsigned char)self->ob_sval[i]);
23     }
24     //检测是否是一个切片
25     else if (PySlice_Check(item)) {
26         //起始位置、终止位置、步长、拷贝的字节个数、循环变量
27         Py_ssize_t start, stop, step, slicelength, i;
28         size_t cur; //拷贝的字节所在的位置
29         //两个缓存
30         char* source_buf;
31         char* result_buf;
32         //返回的结果
33         PyObject* result;
34         //这里是会将item解包, 获取起始位置、终止位置、以及步长
35         if (PySlice_Unpack(item, &start, &stop, &step) < 0) {
36             return NULL;
37         }
38         //根据start、stop、step计算拷贝的字节个数
39         slicelength = PySlice_AdjustIndices(PyBytes_GET_SIZE(self), &start,
40 rt,
41                                     &stop, step);
42
43         //slicelength小于等于0的话, 直接返回空的字节序列, 比如val[3: 2]
44         //显然此时是不循环的, 因为start对应的位置在end之后, 而且步长为正
45         if (slicelength <= 0) {
46             return PyBytes_FromStringAndSize("", 0);
```

```

47     }
48     //如果起始位置为0, 步长为1, 且拷贝的字节个数等于字节序列的长度
49     //说明前后没有变化
50     else if (start == 0 && step == 1 &&
51             slicelength == PyBytes_GET_SIZE(self) &&
52             PyBytes_CheckExact(self)) {
53         //那么增加引用计数, 直接返回
54         //显然这又是一个快分支
55         Py_INCREF(self);
56         return (PyObject *)self;
57     }
58     else if (step == 1) {
59         //如果步长是1, 那么从start开始拷贝
60         //拷贝slicelength个字节
61         return PyBytes_FromStringAndSize(
62             PyBytes_AS_STRING(self) + start,
63             slicelength);
64     }
65     else {
66         //走到这里, 说明步长不是1, 只能一个一个拷贝了
67         source_buf = PyBytes_AS_STRING(self);
68         //创建PyBytesObject对象, 空间为slicelength
69         result = PyBytes_FromStringAndSize(NULL, slicelength);
70         if (result == NULL)
71             return NULL;
72
73         //拿到内部的ob_sval
74         result_buf = PyBytes_AS_STRING(result);
75         //从start开始然后逐个字节拷贝过去
76         //依旧循环slicelength次
77         //通过cur记录拷贝的位置, 然后每次循环都加上步长step
78         for (cur = start, i = 0; i < slicelength;
79             cur += step, i++) {
80             result_buf[i] = source_buf[cur];
81         }
82         //返回
83         return result;
84     }
85 }
86 //item要么是整数、要么是切片, 走到这里说明不满足条件
87 else {
88     //比如:item我们传递了一个字符串
89     //显然此时是非法的
90     //所以抛出TypeError异常
91     PyErr_Format(PyExc_TypeError,
92                 "byte indices must be integers or slices, not %.200s"
93                 ,
94                 Py_TYPE(item)->tp_name);
95     //返回空, 因为出现异常了
96     return NULL;
97 }
98 }

```

所以从底层我们可以看到, Python为我们做的事情是真的不少, 一个简单的切片, 在底层要这么多行代码。不过在我们分析完逻辑之后, 会发现其实也不过如此, 毕竟逻辑很好理解。

不过在Python中, 索引操作和切片操作, 我们都可以通过[__getitem__](#)实现。

```

1 class A:
2
3     def __getitem__(self, item):
4         return item

```

```

5
6
7 a = A()
8 print(a[123]) # 123
9 print(a["name"]) # name
10 print(a[1: 5]) # slice(1, 5, None)
11 print(a[1: 5: 2]) # slice(1, 5, 2)
12 print(a["yo": "ha": "哼哼"]) # slice('yo', 'ha', '哼哼')

```

通过 `__getitem__`，我们可以同时实现切片、索引获取，但是当item为字符串时，我们还可以实现字典操作。当然这部分内容，我们会在后面系列中分析类的时候介绍。

判断一个序列是否在指定的序列中：

```

1 >>> val = b"abcdef"
2 >>> b"abc" in val
3 True
4 >>> b"cbd" in val
5 False
6 >>>

```

如果让我们来实现的话，显然是两层for循环，那么Python是怎么做的呢？

```

1 static int
2 bytes_contains(PyObject *self, PyObject *arg)
3 {
4     //比如: b"abc" in b"abcde"会调用这里的bytes_contains
5     //self就是b"abcde"对应的PyBytesObject的指针
6     //arg是b"abc"对应的PyBytesObject的指针
7
8     //显然这里调用了_Py_bytes_contains
9     //传入了self -> ob_sval, self -> ob_size, arg
10    return _Py_bytes_contains(PyBytes_AS_STRING(self), PyBytes_GET_SIZE(
11 self), arg);
12 }
13 //上面的源码没有说明, 显然是在bytesobject.c中
14 //但是_Py_bytes_contains位于bytes_methods.c中
15 _Py_bytes_contains(const char *str, Py_ssize_t len, PyObject *arg)
16 {
17     //将arg转成整型
18     //但是显然只有当arg -> ob_sval的有效字节为1时才可以这么做
19     Py_ssize_t ival = PyNumber_AsSsize_t(arg, NULL);
20     //如果没有异常发生, 那么PyErr_Occurred()会返回NULL
21     //有异常发生, 那么PyErr_Occurred()会返回非NULL指针
22     //注意这个异常会表现在Python层面, 但在C里面程序依旧是正常执行的
23     //并且PyNumber_AsSsize_t在转换失败时会返回-1
24     //如果ival不等于-1证明没有异常发生, 等于-1的话可能恰好转换的结果也是-1
25     //那么会继续判断PyErr_Occurred()
26     if (ival == -1 && PyErr_Occurred()) {
27         //当这个条件满足时, 证明转化失败了,
28         //意味着arg -> ob_sval的有效字节数大于1
29         Py_buffer varg; //缓冲区
30         Py_ssize_t pos; //遍历位置
31         PyErr_Clear(); //这里将异常清空
32         //拿到缓冲区
33         if (PyObject_GetBuffer(arg, &varg, PyBUF_SIMPLE) != 0)
34             return -1;
35         //调用stringlib_find找到其位置, 里面也是使用了循环
36         pos = stringlib_find(str, len,
37                             varg.buf, varg.len, 0);
38         PyBuffer_Release(&varg); //释放缓冲区
39         //如果pos大于0确实找到了

```

```

40         return pos >= 0;
41     }
42     //否则说明只有一个字节, 但是字节不合法
43     if (ival < 0 || ival >= 256) {
44         PyErr_SetString(PyExc_ValueError, "byte must be in range(0, 256)"
45     );
46         return -1;
47     }
48     //走到这里说明是单个字节, 并且合法
49     //那么直接调用C中的memchr去寻找即可
    return memchr(str, (int) ival, len) != NULL;
}

```

以上就是bytes对象的行为相关的内容，虽然源代码读起来比较枯燥，但通过阅读源代码绝对是水平提升最快的一种方式。

收录于合集 [#CPython 97](#)

[< 上一篇](#)

《源码探秘 CPython》18. bytes 对象的缓存池

[下一篇 >](#)

《源码探秘 CPython》16. bytes 对象的行为 (上)

喜欢此内容的人还喜欢

Python第一阶段第3课——与计算机的沟通
慧心学社



linux系统利用awk统计nginx日志
SZMSJSZ



python第一阶段第二课《与世界打招呼》
慧心学社

