《源码探秘 CPython》18. bytes 对象的缓存池

原创 古明地觉 古明地觉的编程教室 2022-01-27 09:30

效率问题

先来探究一下bytes对象的效率，我们知道Python的不可变对象在运算时，处理方式是再创建一个新的。所以三个bytes对象a、b、c在相加时，那么会先根据a + b创建**临时对象**，然后再根据**临时对象+c**创建新的对象，最后返回指针。所以：

```
result = b""
for _ in bytes_list:
    result += _
```

这是一种效率非常低下的做法，因为涉及大量临时对象的创建和销毁，不仅是这里字节序列，后面要分析的字符串也是同样的道理。

官方推荐的做法是，使用join，字符串和字节序列都可以对一个列表进行join，将列表里面的多个字符串或者字节序列join在一起。

举个Python中的例子，我们以字符串为例，字节序列同样如此：

```
def bad():
    s = ""
    for _ in range(1, 10):
        s += str(_)
    return s

def good():
    l = []
    for _ in range(1, 10):
        l.append(str(_))
    return "".join(l)

def better():
    return "".join(str(_) for _ in range(1, 10))

def best():
    return "".join(map(str, range(1, 10)))
```

字节序列缓存池

为了优化单字节bytes对象的创建效率，Python底层维护了一个缓存池，该缓存池是一个**PyBytesObject***类型的数组。

```
static PyBytesObject *characters[UCHAR_MAX + 1];
```

Python内部创建单字节bytes对象时，先检查目标对象是否已在缓存池中。PyBytes_FromStringAndSize函数是负责创建bytes对象的一个常用的Python/C API，位于Objects/bytesobject.c中：

```
PyObject *
PyBytes_FromStringAndSize(const char *str, Py_ssize_t size)
{
    //PyBytesObject对象的指针
```

```
    PyBytesObject *op;
    if (size < 0) {
        //显然size不可以小于0
        PyErr_SetString(PyExc_SystemError,
            "Negative size passed to PyBytes_FromStringAndSize");
        return NULL;
    }
    //如果size为1，表明创建的是单字节对象
    //当然str不可以为NULL，而且获取到的字节必须要在characters里面
    if (size == 1 && str != NULL &&
        (op = characters[*str & UCHAR_MAX]) != NULL)
    {
#ifdef COUNT_ALLOCS
        _Py_one_strings++;
#endif
        //增加引用计数，返回指针
        Py_INCREF(op);
        return (PyObject *)op;
    }

    //否则话创建新的PyBytesObject，此时是个空
    op = (PyBytesObject *)_PyBytes_FromSize(size, 0);
    if (op == NULL)
        return NULL;
    if (str == NULL)
        return (PyObject *) op;

    //不管size是多少，都直接拷贝即可
    memcpy(op->ob_sval, str, size);
    //但是size是1的话，除了拷贝还会放到缓存池characters中
    if (size == 1) {
        characters[*str & UCHAR_MAX] = op;
        Py_INCREF(op);
    }
    //返回其指针
    return (PyObject *) op;
}
```

由此可见，当Python程序开始运行时，字节序列缓存池是空的。但随着**单字节bytes对象**的创建，缓存池中的对象慢慢多了起来。

这样一来，单字节序列首次创建后便在缓存池中缓存起来；后续再次使用时， Python直接从缓存池中取，避免重复创建和销毁。与前面章节介绍的小整数对象池一样，字节序列缓存池也只能容纳为数不多的 256 个单字节序列，但使用频率非常高。

缓存池技术作为一种以空间换时间的优化手段，只需较小的内存为代价，便可明显提升执行效率。

```
>>> a1 = b"a"
>>> a2 = b"a"
>>> a1 is a2
True
>>>
>>> a1 = b"ab"
>>> a2 = b"ab"
>>> a1 is a2
False
>>>
```

显然此时不需要解释了，单字节bytes对象会缓存起来，不是单字节则不会缓存。

小结

以上就是bytes对象的全部内容，我们说：

- bytes对象是一个变长、不可变对象，内部的值是通过一个C的字符数组来维护的；
- bytes也是序列型操作，它支持的操作在bytes_as_sequence和bytes_as_mapping中；
- Python内部通过维护字节序列缓存池来优化单字节bytes对象的创建和销毁操作；
- 缓存池是一种常用的以空间换时间的优化技术；

最后，Python除了bytes对象之外，还有一个bytearray对象，它的表现和bytes对象是完全一致的，但一个是可变对象、一个是不可变对象。bytearray对象底层对应的结构体是**PyByteArrayObject**，其相关操作、以及类型对象都位于bytearrayobject.c中，有兴趣可以看一下。

下一篇我们来介绍字符串。

插句题外话，我被集中隔离了。。。。。。我在 4 楼办公，而 5 楼有一个确诊病例。最后显示我和他是密接，要在酒店隔离到二月十号，但是我没中招。所以想提醒大家，一定要注意防护哦，曾以为病例离我很远，没想到它也悄无声息地来到了我的身边。

收录于合集 #CPython 97

‹ 上一篇
《源码探秘 CPython》19. 字符集和字符编码

下一篇 ›
《源码探秘 CPython》17. bytes 对象的行为（下）

文章已于2022-06-12修改