5.9 读取二进制数据到可变缓冲区中¶

问题¶

你想直接读取二进制数据到一个可变缓冲区中，而不需要做任何的中间复制操作。 或者你想原地修改数据并将它写回到一个文件中去。

解决方案¶

为了读取数据到一个可变数组中，使用文件对象的 `readinto()` 方法。比如：

```
import os.path

def read_into_buffer(filename):
    buf = bytearray(os.path.getsize(filename))
    with open(filename, 'rb') as f:
        f.readinto(buf)
    return buf
```

下面是一个演示这个函数使用方法的例子：

```
>>> # Write a sample file
>>> with open('sample.bin', 'wb') as f:
...     f.write(b'Hello World')
...
>>> buf = read_into_buffer('sample.bin')
>>> buf
bytearray(b'Hello World')
>>> buf[0:5] = b'Hello'
>>> buf
bytearray(b'Hello World')
>>> with open('newsample.bin', 'wb') as f:
...     f.write(buf)
...
11
>>>
```

讨论¶

文件对象的 `readinto()` 方法能被用来为预先分配内存的数组填充数据，甚至包括由 `array` 模块或 `numpy` 库创建的数组。 和普通 `read()` 方法不同的是， `readinto()` 填充已存在的缓冲区而不是为新对象重新分配内存再返回它们。 因此，你可以使用它来避免大量的内存分配操作。 比如，如果你读取一个由相同大小的记录组成的二进制文件时，你可以像下面这样写：

```
record_size = 32 # Size of each record (adjust value)

buf = bytearray(record_size)
with open('somefile', 'rb') as f:
    while True:
        n = f.readinto(buf)
        if n < record_size:
            break
        # Use the contents of buf
        ...
```

另外有一个有趣特性就是 `memoryview` ， 它可以通过零复制的方式对已存在的缓冲区执行切片操作，甚至还能修改它的内容。比如：

```
>>> buf
bytearray(b'Hello World')
>>> m1 = memoryview(buf)
>>> m2 = m1[-5:]
>>> m2
<memory at 0x100681390>
```

```
>>> m2[:] = b'WORLD'
>>> buf
bytearray(b'Hello WORLD')
>>>
```

使用 `f.readinto()` 时需要注意的是，你必须检查它的返回值，也就是实际读取的字节数。

如果字节数小于缓冲区大小，表明数据被截断或者被破坏了(比如你期望每次读取指定数量的字节)。

最后，留心观察其他函数库和模块中和 `into` 相关的函数(比如 `recv_into()` ， `pack_into()` 等)。 Python的很多其他部分已经能支持直接的I/O或数据访问操作，这些操作可被用来填充或修改数组和缓冲区内容。

关于解析二进制结构和 `memoryviews` 使用方法的更高级例子，请参考6.12小节。