5.4 读写字节数据¶

问题¶

你想读写二进制文件，比如图片，声音文件等等。

解决方案¶

使用模式为 `rb` 或 `wb` 的 `open()` 函数来读取或写入二进制数据。比如：

```
# Read the entire file as a single byte string
with open('somefile.bin', 'rb') as f:
    data = f.read()

# Write binary data to a file
with open('somefile.bin', 'wb') as f:
    f.write(b'Hello World')
```

在读取二进制数据时，需要指明的是所有返回的数据都是字节字符串格式的，而不是文本字符串。 类似的，在写入的时候，必须保证参数是以字节形式对外暴露数据的对象(比如字节字符串，字节数组对象等)。

讨论¶

在读取二进制数据的时候，字节字符串和文本字符串的语义差异可能会导致一个潜在的陷阱。 特别需要注意的是，索引和迭代动作返回的是字节的值而不是字节字符串。比如：

```
>>> # Text string
>>> t = 'Hello World'
>>> t[0]
'H'
>>> for c in t:
...     print(c)
...
H
e
l
l
o
...
>>> # Byte string
>>> b = b'Hello World'
>>> b[0]
72
>>> for c in b:
...     print(c)
...
72
101
108
108
111
...
>>>
```

如果你想从二进制模式的文件中读取或写入文本数据，必须确保要进行解码和编码操作。比如：

```
with open('somefile.bin', 'rb') as f:
    data = f.read(16)
    text = data.decode('utf-8')

with open('somefile.bin', 'wb') as f:
    text = 'Hello World'
    f.write(text.encode('utf-8'))
```

二进制I/O还有一个鲜为人知的特性就是数组和C结构体类型能直接被写入，而不需要中间转换为自己对象。比如：

```
import array
nums = array.array('i', [1, 2, 3, 4])
with open('data.bin','wb') as f:
    f.write(nums)
```

这个适用于任何实现了被称之为"缓冲接口"的对象，这种对象会直接暴露其底层的内存缓冲区给能处理它的操作。 二进制数据的写入就是这类操作之一。

很多对象还允许通过使用文件对象的 `readinto()` 方法直接读取二进制数据到其底层的内存中去。比如：

```
>>> import array
>>> a = array.array('i', [0, 0, 0, 0, 0, 0, 0, 0])
>>> with open('data.bin', 'rb') as f:
...     f.readinto(a)
...
16
>>> a
array('i', [1, 2, 3, 4, 0, 0, 0, 0])
>>>
```

但是使用这种技术的时候需要格外小心，因为它通常具有平台相关性，并且可能会依赖字长和字节顺序(高位优先和低位优先)。 可以查看5.9小节中另外一个读取二进制数据到可修改缓冲区的例子。