11.13 发送与接收大型数组¶

问题¶

你要通过网络连接发送和接受连续数据的大型数组，并尽量减少数据的复制操作。

解决方案¶

下面的函数利用 `memoryviews` 来发送和接受大数组：

```
# zerocopy.py

def send_from(arr, dest):
    view = memoryview(arr).cast('B')
    while len(view):
        nsent = dest.send(view)
        view = view[nsent:]

def recv_into(arr, source):
    view = memoryview(arr).cast('B')
    while len(view):
        nrecv = source.recv_into(view)
        view = view[nrecv:]
```

为了测试程序，首先创建一个通过socket连接的服务器和客户端程序：

```
>>> from socket import *
>>> s = socket(AF_INET, SOCK_STREAM)
>>> s.bind(('', 25000))
>>> s.listen(1)
>>> c,a = s.accept()
>>>
```

在客户端（另外一个解释器中）：

```
>>> from socket import *
>>> c = socket(AF_INET, SOCK_STREAM)
>>> c.connect(('localhost', 25000))
>>>
```

本节的目标是你能通过连接传输一个超大数组。这种情况的话，可以通过 `array` 模块或 `numpy` 模块来创建数组：

```
# Server
>>> import numpy
>>> a = numpy.arange(0.0, 50000000.0)
>>> send_from(a, c)
>>>

# Client
>>> import numpy
>>> a = numpy.zeros(shape=50000000, dtype=float)
>>> a[0:10]
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
>>> recv_into(a, c)
>>> a[0:10]
```

```
>>> a[0:10]
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])
>>>
```

讨论¶

在数据密集型分布式计算和平行计算程序中，自己写程序来实现发送/接受大量数据并不常见。 不过，要是你确实想这样做，你可能需要将你的数据转换成原始字节，以便给低层的网络函数使用。 你可能还需要将数据切割成多个块，因为大部分和网络相关的函数并不能一次性发送或接受超大数据块。

一种方法是使用某种机制序列化数据——可能将其转换成一个字节字符串。 不过，这样最终会创建数据的一个复制。 就算你只是零碎的做这些，你的代码最终还是会有大量的小型复制操作。

本节通过使用内存视图展示了一些魔法操作。 本质上，一个内存视图就是一个已存在数组的覆盖层。不仅仅是那样， 内存视图还能以不同的方式转换成不同类型来表现数据。 这个就是下面这个语句的目的：

```
view = memoryview(arr).cast('B')
```

它接受一个数组 arr并将其转换为一个无符号字节的内存视图。这个视图能被传递给socket相关函数， 比如 `socket.send()` 或 `send.recv_into()` 。 在内部，这些方法能够直接操作这个内存区域。例如， `sock.send()` 直接从内存中发生数据而不需要复制。 `send.recv_into()` 使用这个内存区域作为接受操作的输入缓冲区。

剩下的一个难点就是socket函数可能只操作部分数据。 通常来讲，我们得使用很多不同的 `send()` 和 `recv_into()` 来传输整个数组。 不用担心，每次操作后，视图会通过发送或接受字节数量被切割成新的视图。 新的视图同样也是内存覆盖层。因此，还是没有任何的复制操作。

这里有个问题就是接受者必须事先知道有多少数据要被发送， 以便它能预分配一个数组或者确保它能将接受的数据放入一个已经存在的数组中。 如果没办法知道的话，发送者就得先将数据大小发送过来，然后再发送实际的数组数据。