Numpy数组操作

数组广播机制：

广播，就是`Numpy`不同维度数组之间可以进行运算的功能。简单来说，如果两个数组的维数不相同，则元素到元素的操作是不可能的。 然而，在 `NumPy` 中仍然可以对形状不相似的数组进行操作，因为它拥有广播功能。 较小的数组会**广播**到较大数组的大小，以便使它们的形状可兼容。

数组与数的计算：

在`Python`列表中，想要对列表中所有的元素都加一个数，要么采用`map`函数，要么循环整个列表进行操作。但是`NumPy`中的数组可以直接在数组上进行操作。示例代码如下：

```
import numpy as np
a1 = np.random.random((3,4))
print(a1)
# 如果想要在a1数组上所有元素都乘以10，那么可以通过以下来实现
a2 = a1*10
print(a2)
# 也可以使用round让所有的元素只保留2位小数
a3 = a2.round(2)
```

以上例子是相乘，其实相加、相减、相除也都是类似的。

数组与数组的计算：

1. 结构相同的数组之间的运算：

```
a1 = np.arange(0,24).reshape((3,8))
a2 = np.random.randint(1,10,size=(3,8))
a3 = a1 + a2 #相减/相除/相乘都是可以的
print(a1)
print(a2)
print(a3)
```

2. 与行数相同并且只有1列的数组之间的运算：

```
a1 = np.random.randint(10,20,size=(3,8)) #3行8列
a2 = np.random.randint(1,10,size=(3,1)) #3行1列
a3 = a1 - a2 #行数相同，且a2只有1列，能互相运算
print(a3)
```

3. 与列数相同并且只有1行的数组之间的运算：

```
a1 = np.random.randint(10,20,size=(3,8)) #3行8列
a2 = np.random.randint(1,10,size=(1,8))
a3 = a1 - a2
print(a3)
```

广播原则：

如果两个数组的后缘维度（trailing dimension，即从末尾开始算起的维度）的轴长度相符或其中一方的长度为1，则认为他们是广播兼容的。广播会在缺失和（或）长度为1的维度上进行。。看以下案例分析：

1. `shape`为`(3,8,2)`的数组能和`(8,3)`的数组进行运算吗？
   分析：不能，因为按照广播原则，从后面往前面数，`(3,8,2)`和`(8,3)`中的`2`和`3`不相等，所以不能进行运算。
2. `shape`为`(3,8,2)`的数组能和`(8,1)`的数组进行运算吗？
   分析：能，因为按照广播原则，从后面往前面数，`(3,8,2)`和`(8,1)`中的`2`和`1`虽然不相等，但是因为有一方的长度为`1`，所以能参与运算。
3. `shape`为`(3,1,4)`的数组能和`(8,1)`的数组进行运算吗？
   分析：能，因为按照广播原则，从后面往前面数，`(3,1,4)`和`(8,1)`中的`4`和`1`虽然不相等且`1`和`8`不相等，但是因为这两项中有一方的长度为`1`，所以能参与运算。

数组形状的操作：

可以通过一些函数，非常方便的操作数组的形状。

reshape和resize方法：

两个方法都是用来修改数组形状的，但是有一些不同。

1. `reshape`是将数组转换成指定的形状，然后返回转换后的结果，对于原数组的形状是不会发生改变的。调用方式：

   ```
   a1 = np.random.randint(0,10,size=(3,4))
   a2 = a1.reshape((2,6)) #将修改后的结果返回，不会影响原数组本身
   ```

2. `resize`是将数组转换成指定的形状，会直接修改数组本身。并不会返回任何值。调用方式：

   ```
   a1 = np.random.randint(0,10,size=(3,4))
   a1.resize((2,6)) #a1本身发生了改变
   ```

flatten和ravel方法：

两个方法都是将多维数组转换为一维数组，但是有以下不同：

1. `flatten`是将数组转换为一维数组后，然后将这个拷贝返回回去，所以后续对这个返回值进行修改不会影响之前的数组。

2. `ravel`是将数组转换为一维数组后，将这个视图（可以理解为引用）返回回去，所以后续对这个返回值进行修改会影响之前的数组。
   比如以下代码：

```
x = np.array([[1, 2], [3, 4]])
x.flatten()[1] = 100 #此时的x[1]的位置元素还是1
x.ravel()[1] = 100 #此时x[1]的位置元素是100
```

不同数组的组合：

如果有多个数组想要组合在一起，也可以通过其中的一些函数来实现。

1. `vstack`：将数组按垂直方向进行叠加。数组的列数必须相同才能叠加。示例代码如下：

```
a1 = np.random.randint(0,10,size=(3,5))
a2 = np.random.randint(0,10,size=(1,5))
a3 = np.vstack([a1,a2])
```

2. `hstack`：将数组按水平方向进行叠加。数组的行必须相同才能叠加。示例代码如下：

```
a1 = np.random.randint(0,10,size=(3,2))
a2 = np.random.randint(0,10,size=(3,1))
a3 = np.hstack([a1,a2])
```

3. `concatenate([],axis)`：将两个数组进行叠加，但是具体是按水平方向还是按垂直方向。则要看`axis`的参数，如果`axis=0`，那么代表的是往垂直方向（行）叠加，如果`axis=1`，那么代表的是往水平方向（列）上叠加，如果`axis=None`，那么会将两个数组组合成一个一维数组。需要注意的是，如果往水平方向上叠加，那么行必须相同，如果是往垂直方向叠加，那么列必须相同。示例代码如下：

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])
np.concatenate([a, b], axis=0)
# 结果：
array([[1, 2],
    [3, 4],
    [5, 6]])

np.concatenate([a, b.T], axis=1)
# 结果：
array([[1, 2, 5],
    [3, 4, 6]])

np.concatenate([a, b], axis=None)
# 结果：
array([1, 2, 3, 4, 5, 6])
```

数组的切割：

通过`hsplit`和`vsplit`以及`array_split`可以将一个数组进行切割。

1. `hsplit`：按照水平方向进行切割。用于指定分割成几列，可以使用数字来代表分成几部分，也可以使用数组来代表分割的地方。示例代码如下：

```
a1 = np.arange(16.0).reshape(4, 4)
np.hsplit(a1,2) #分割成两部分
>>> array([[ 0.,  1.],
       [ 4.,  5.],
       [ 8.,  9.],
       [12., 13.]]), array([[ 2.,  3.],
       [ 6.,  7.],
       [10., 11.],
       [14., 15.]])

np.hsplit(a1,[1,2]) #代表在下标为1的地方切一刀，下标为2的地方切一刀，分成三部分
>>> [array([[ 0.],
       [ 4.],
       [ 8.],
       [12.]]), array([[ 1.],
       [ 5.],
       [ 9.],
       [13.]]), array([[ 2.,  3.],
       [ 6.,  7.],
       [10., 11.],
       [14., 15.]])]
```

2. `vsplit`：按照垂直方向进行切割。用于指定分割成几行，可以使用数字来代表分成几部分，也可以使用数组来代表分割的地方。示例代码如下：

```
np.vsplit(x,2) #代表按照行总共分成2个数组
>>> [array([[0., 1., 2., 3.],
       [4., 5., 6., 7.]]), array([[ 8.,  9., 10., 11.],
       [12., 13., 14., 15.]])]

np.vsplit(x,(1,2)) #代表按照行进行划分，在下标为1的地方和下标为2的地方分割
>>> [array([[0., 1., 2., 3.]]),
    array([[4., 5., 6., 7.]]),
    array([[ 8.,  9., 10., 11.],
           [12., 13., 14., 15.]])]
```

3. `split/array_split(array,indicate_or_seciont,axis)`：用于指定切割方式，在切割的时候需要指定是按照行还是按照列，`axis=1`代表按照列，`axis=0`代表按照行。示例代码如下：

```
np.array_split(x,2,axis=0) #按照垂直方向切割成2部分
>>> [array([[0., 1., 2., 3.],
       [4., 5., 6., 7.]]), array([[ 8.,  9., 10., 11.],
       [12., 13., 14., 15.]])]
```

数组（矩阵）转置和轴对换：

`numpy`中的数组其实就是线性代数中的矩阵。矩阵是可以进行转置的。`ndarray`有一个`T`属性，可以返回这个数组的转置的结果。示例代码如下：

```
a1 = np.arange(0,24).reshape((4,6))
a2 = a1.T
print(a2)
```

另外还有一个方法叫做`transpose`，这个方法返回的是一个View，也即修改返回值，会影响到原来数组。示例代码如下：

```
a1 = np.arange(0,24).reshape((4,6))
a2 = a1.transpose()
```

为什么要进行矩阵转置呢，有时候在做一些计算的时候需要用到。比如做矩阵的内积的时候。就必须将矩阵进行转置后再乘以之前的矩阵：

```
a1 = np.arange(0,24).reshape((4,6))
a2 = a1.T
print(a1.dot(a2))
```

深拷贝和浅拷贝

在操作数组的时候，它们的数据有时候拷贝进一个新的数组，有时候又不是。这经常是初学者感到困惑。下面有三种情况：

不拷贝：

如果只是简单的赋值，那么不会进行拷贝。示例代码如下：

```
a = np.arange(12)
b = a #这种情况不会进行拷贝
print(b is a) #返回True，说明b和a是相同的
```

View或者浅拷贝：

有些情况，会进行变量的拷贝，但是他们所指向的内存空间都是一样的，那么这种情况叫做浅拷贝，或者叫做`View(视图)`。比如以下代码：

```
a = np.arange(12)
c = a.view()
print(c is a) #返回False，说明c和a是两个不同的变量
c[0] = 100
print(a[0]) #打印100，说明对c上的改变，会影响a上面的值，说明他们指向的内存空间还是一样的，这种叫做浅拷贝，或者说是view
```

深拷贝：

将之前数据完完整整的拷贝一份放到另外一块内存空间中，这样就是两个完全不同的值了。示例代码如下：

```
a = np.arange(12)
d = a.copy()
print(d is a) #返回False，说明d和a是两个不同的变量
d[0] = 100
print(a[0]) #打印0，说明d和a指向的内存空间完全不同了。
```

例子：

像之前讲到的`flatten`和`ravel`就是这种情况，`ravel`返回的就是View，而`flatten`返回的就是深拷贝。