NumPy数值计算

第二章 NumPy数值计算

Python

2.2 数组操作

2.2 数组操作

NumPy中包含了一些函数用于操作数组，大致分为六类：修改数组形状、翻转数组、修改数组维度、连接数组、分割数组、数组元素的添加与删除。

2.2.1 修改数组形状

1、reshape函数

函数reshape 在不改变数据的条件下修改形状，其格式如下。

```
numpy.reshape(arr, newshape, order='C')  # 全局函数
array.reshape(newshape,order='C')  # 成员函数
```

其函数参数如表所示。

参数	描述
array	要修改形状的数组
newshape	新形状参数，为整数或者整数元组，新的形状应当兼容原有形状
order	'C' 按行，'F'按列

[示例] reshape函数

```
import numpy as np

a = np.arange(12)
print("a=\n",a)

b = np.reshape(a,(2,6),order='C')
print("b=\n",b)

c = a.reshape((1,3,4),order='F')
print("c=\n",c)

d = a.reshape((2,3,2))
print("d=\n",d)
```

```
a=
 [ 0  1  2  3  4  5  6  7  8  9 10 11]

b=
 [[ 0  1  2  3  4  5]
  [ 6  7  8  9 10 11]]

c=
 [[[ 0  3  6  9]
   [ 1  4  7 10]
   [ 2  5  8 11]]]

d=
 [[[ 0  1]
  [ 2  3]
  [ 4  5]]

 [[ 6  7]
  [ 8  9]
  [10 11]]]
```

2、flatten函数

将高维度数组展平，返回一个一维的数组，其格式如下：

array. flatten(order='C')

```
import numpy as np


a = np.arange(6).reshape(2,3)
print("a=\n",a)
b = a.flatten()
print("b=\n",b)
c = a.flatten(order='F')
print("c=\n",c)
d = a.flat[::] # 数组的flat属性返回flatiter对象，用切片得到列表
print("d=\n",d)
```

```
a=
 [[0 1 2]
 [3 4 5]]

b=
 [0 1 2 3 4 5]

c=
 [0 3 1 4 2 5]

d=
 [0 1 2 3 4 5]
```

3、ravel函数

函数ravel 展平数组元素，返回的是数组视图，其函数格式如下所示：

numpy.ravel(a, order='C')

array.ravel(order='C')

与flatten()的不同：

1）使用ravel()创建的新数组实际上是对父数组的引用（即“视图”）。这意味着对新数组的任何更改也将影响父数组。因为ravel不创建拷贝，所以它的内存效率很高。

2）ravel函数有成员函数和全局函数两种形式，flatten只有成员函数形式。

[示例] ravel和flatten的区别

```
import numpy as np

a = np.arange(12).reshape(3,4)
print(a)

c = a.flatten()  # d指向新数组
c[0] = -1    # 修改d[0]
print(c)
print(a)   # a[0]未被修改

b = np.ravel(a)  # b指向a数组
b[0] = -2    # 修改b[0]
print(b)
print(a)   # a[0]被修改
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

[-1  1  2  3  4  5  6  7  8  9 10 11]

[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

[-2  1  2  3  4  5  6  7  8  9 10 11]

[[-2  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

2.2 数组操作

2.2.2 翻转数组

数组翻转函数如表所示。

函数	描述
transpose	对换数组的维度
swapaxes	对换数组的两个轴

1、numpy.transpose函数

交换数组的维度索引值（轴序号），格式如下所示：

numpy.transpose(arr, axes)

narray.transpose(axes)

参数说明如下所示：

- arr：要操作的数组。
- axes：整数序列，对应维度**序号**。若是2维，则可省略该参数，直接交换行列，即转置。

[示例] 一维数组用transpose函数和T属性实现转置

```
import numpy as np
a = np.arange(6).reshape(2,3)
print('a=', a)

b = a.transpose()
print('b=', b)

c = np.transpose(a)
print('c=', c)

d = a.T
print('d=', d)
```

```
a= [[0 1 2]
    [3 4 5]]

b= [[0 3]
    [1 4]
    [2 5]]

c= [[0 3]
    [1 4]
    [2 5]]

d= [[0 3]
    [1 4]
    [2 5]]
```

[示例] 多维数组的transpose转置

```
import numpy as np
a = np.arange(16).reshape((2,2,4))
print(a)
# 注意参数是维度序号
b = a.transpose(2,1,0)  # b的形状变为(4,2,2)
print(b)
```

```
[[[ 0  1  2  3]
  [ 4  5  6  7]]

 [[ 8  9 10 11]
  [12 13 14 15]]]
```

原维度顺序是(0,1,2)，转(2,1,0)，即元素的第0轴和第2轴**序号置换**，第0轴和第2轴的**长度置换**。
以数组中的元素2为例，原位置为：a[0][0][2]
将其第一维和第三维序号交换，因此，2转置后的位置为：b[2][0][0]。

```
[[[ 0  8]
  [ 4 12]]

 [[ 1  9]
  [ 5 13]]

 [[ 2 10]
  [ 6 14]]

 [[ 3 11]
  [ 7 15]]]
```

【示例】transpose函数的应用

```
# 多维数组的transpose转置
import numpy as np
l = [[[92.3,92.8,94.5,93.6,95.1],
      [92.4,92.9,93.7,94.2,94.8]],
     [[32.3,34.2,34.5,33.6,35.1],
      [33.5,32.5,33.2,34.2,36.4]]]
a = np.array(l)
print(a)
b = a.transpose(2,1,0)
print()
print(b)
```

		周一	周二	周三	周四	周五
Baidu	开盘	92.3	92.8	94.5	93.6	95.1
	收盘	92.4	92.9	93.7	94.2	94.8
JD	开盘	32.3	34.2	34.5	33.6	35.1
	收盘	33.5	32.5	33.2	34.2	36.4

		Baidu	JD
周一	开盘	92.3	32.3
	收盘	92.4	33.5
周二	开盘	92.8	34.2
	收盘	92.9	32.5
周三	开盘	94.5	34.5
	收盘	93.7	33.2
周四	开盘	93.6	33.6
	收盘	94.2	34.2
周五	开盘	95.1	35.1
	收盘	94.8	36.4

2、numpy.swapaxes函数

numpy.swapaxes函数用于交换数组的**两个轴**，格式如下所示：

numpy.swapaxes(arr, axis1, axis2)

narray.swapaxes(axis1, axis2)

swapaxes实际上也是针对轴索引进行变化，区别就在于transpose可以一次传入多个参数，对若干轴进行交换，而swapaxes只能两两置换，且swapaxes(i,j)和swapaxes(j,i)没有区别。

[示例] swapaxes函数

```
import numpy as np


a = np.arange(16).reshape((2,2,4))
print("a=\n", a)


b = a.swapaxes(2,0)   # 转置结果与a.transpose(2,1,0)相同
print("b=\n", b)
```

```
[[[ 0  1  2  3]
  [ 4  5  6  7]]

 [[ 8  9 10 11]
  [12 13 14 15]]]
```

```
[[[ 0  8]
  [ 4 12]]

 [[ 1  9]
  [ 5 13]]

 [[ 2 10]
  [ 6 14]]

 [[ 3 11]
  [ 7 15]]]
```

2.2.3 连接数组

数组连接函数如表所示。

函数	描述
concatenate	沿轴连接数组或序列
stack	沿着新轴堆叠一系列数组
hstack	水平堆叠序列中的数组（列方向）
vstack	竖直堆叠序列中的数组（行方向）

1、numpy.concatenate

numpy.concatenate函数用于**沿指定轴**连接**形状兼容、维度相同**的两个或多个数组，格式如下所示。

numpy.concatenate((a1, a2, ...), axis=0)

参数:

✓ (a1, a2, ...)：形状兼容的数组组成的元组。

✓ axis：沿着它连接数组的轴序号，默认为 0。

[示例] 沿0轴连接，数组的1轴长度要相等。

```
import numpy as np

a = np.zeros((2,3),'int')
print('a=\n', a)

b = np.ones((3,3),dtype='i')
print('b=\n',b)

c = np.concatenate((a,b),0)    # 沿0轴连接
print("c=\n", c)
```

```
a=
 [[0 0 0]
  [0 0 0]]

b=
 [[1 1 1]
  [1 1 1]
  [1 1 1]]

c=
 [[0 0 0]
  [0 0 0]
  [1 1 1]
  [1 1 1]
  [1 1 1]]
```

[示例] 沿1轴连接，数组的0轴长度要相等。

```
import numpy as np

a = np.zeros((2,3),'int')
print('a=\n', a)

b = np.ones((2,4),dtype='i')
print('b=\n',b)

c = np.concatenate((a,b),1)   # 沿1轴连接
print("c=\n", c)
```

```
a=
 [[0 0 0]
  [0 0 0]]

b=
 [[1 1 1 1]
  [1 1 1 1]]

c=
 [[0 0 0 1 1 1 1]
  [0 0 0 1 1 1 1]]
```

2、numpy.hstack

- 在水平方向上堆叠**两个或多个**形状兼容的数组形成新的数组。
- 机器学习数据集准备过程中，可以用于将数据列与标签列在水平方向上合并，从而得到带标签的数据集。

```
import numpy as np

a = np.array([['姓名'] ,['年龄'], ['籍贯']])
b = np.array([['张三','李四','王五'], [18,19,16], ['长沙','上海','南京']])

c = np.hstack((a,b))   # 等同于np.concatenate((a,b),1)
print('c=\n', c)
```

```
a=
 [['姓名']
 ['年龄']
 ['籍贯']]
```

```
b=
 [['张三' '李四' '王五']
 ['18' '19' '16']
 ['长沙' '上海' '南京']]
```

```
c=
 [['姓名' '张三' '李四' '王五']
 ['年龄' '18' '19' '16']
 ['籍贯' '长沙' '上海' '南京']]
```

3、numpy.vstack

- 在垂直方向上堆叠两个或多个形状兼容的数组形成新的数组。
- 机器学习数据集准备过程中，可以用于将标题行以及从多个数据文件中加载的数据行在垂直方向上合并，从而将所有数据集整合为一个数据集。

```
import numpy as np
a = np.array(['姓名', '年龄', '籍贯'])
b = np.array([['张三',18,'长沙'],['李四',19,'上海'],['王五',16,'南京']])

c = np.vstack((a,b))  # 一维a数组可以与二维b数组vstack
# np.concatenate((a,b),0)出错，需要将a改为[['姓名','年龄','籍贯']]
print("c=\n", c)
```

```
a=
 ['姓名' '年龄' '籍贯']
```

```
b=
 [['张三' '18' '长沙']
  ['李四' '19' '上海']
  ['王五' '16' '南京']]
```

```
c=
 [['姓名' '年龄' '籍贯']
  ['张三' '18' '长沙']
  ['李四' '19' '上海']
  ['王五' '16' '南京']]
```

4、 numpy.stack

numpy.stack(arrays, axis=0)

将一系列数组沿一个**新轴**合并。其中axis指定了新轴在所得数组维度中的索引值。如果axis=0，将沿第一个维度，如果axis=-1将沿最后一个维度。

注意：

- concatenate：沿现有轴连接数组，不增加新维度。
- stack：沿新轴堆叠数组，增加一个新维度。

[示例] 沿0轴堆叠3个二维数组（将多个数组作为一个整体处理时，适合在数据的最外层添加一个新的维度）　　此例维度含义： 0：产品 1：月份 2：地区

```
import numpy as np
A = np.array([[111,112,113],
              [211,212,213]])
B = np.array([[121,122,123],
              [221,222,223]])
C = np.array([[131,132,133],
              [231,232,233]])
s0 = np.stack((A,B,C), axis=0)
print(s0.shape)
print(s0)
A_region1_sales = s0[:, 0, 0]
total_sales = np.sum(A_region1_sales)
print("产品A第一季度在地区1的总销售量:", total_sales)
```

```
(3, 2, 3)
[[[111 112 113]
  [211 212 213]]

 [[121 122 123]
  [221 222 223]]

 [[131 132 133]
  [231 232 233]]]
产品A第一季度在地区1的总销售量: 363
```

[示例] 沿1轴堆叠3个二维数组（当需要将多个数组按行堆叠时，适合在数据的中间维度添加一个新的维度）

```
import numpy as np
A = np.array([[111,112,113],
              [211,212,213]])
B = np.array([[121,122,123],
              [221,222,223]])
C = np.array([[131,132,133],
              [231,232,233]])
s0 = np.stack((A,B,C), axis=1)
print(s0.shape)
print(s0)
A_region1_sales = s0[0, :, 0]
total_sales = np.sum(A_region1_sales)
print("产品A第一季度在地区1的总销售量:", total_sales)
```

```
(2, 3, 3)
[[[111 112 113]
  [121 122 123]
  [131 132 133]]

 [[211 212 213]
  [221 222 223]
  [231 232 233]]]
产品A第一季度在地区1的总销售量: 363
```

[示例] 沿2轴堆叠3个二维数组（当需要将多个数组按列（或某个特定维度）堆叠时，适合在数据的内层维度添加一个新的维度）

```
import numpy as np
A = np.array([[111,112,113],
              [211,212,213]])
B = np.array([[121,122,123],
              [221,222,223]])
C = np.array([[131,132,133],
              [231,232,233]])
s0 = np.stack((A,B,C), axis=2)
print(s0.shape)
print(s0)
A_region1_sales = s0[0, 0, :]
total_sales = np.sum(A_region1_sales)
print("产品A第一季度在地区1的总销售量:", total_sales)
```

```
(2, 3, 3)
[[[111 121 131]
  [112 122 132]
  [113 123 133]]

 [[211 221 231]
  [212 222 232]
  [213 223 233]]]
产品A第一季度在地区1的总销售量: 363
```

2.2.4 分割数组

分割数组函数如表所示。

函数	描述
split	将一个数组分割为多个子数组
hsplit	将一个数组水平分割为多个子数组（按列）
vsplit	将一个数组垂直分割为多个子数组（按行）

1、 numpy.split

numpy.split函数沿特定的轴将数组分割为**子数组**，返回值增加一维：

numpy.split(array, indices_or_sections, axis)

参数array：被分割的数组。

indices_or_sections：如果是一个整数，就用该数平均切分，如果是一个数组，为沿轴切分的位置。

axis：沿着哪个维度进行切分，**默认为0**。

[示例] 横向切分数组

```
import numpy as np

a = np.arange(12).reshape(4,3)
print('a=',a)

# 沿0轴均切为2份
b = np.split(a,2)
print("b=",b)

print('array(b)=\n',np.array(b))

#沿0轴在2、3位置切分数组
c = np.split(a, [2,3])
print('c=',c)
```

```
a= [[ 0  1  2]
    [ 3  4  5]
    [ 6  7  8]
    [ 9 10 11]]
```

切分位置1

切分位置2

切分位置3

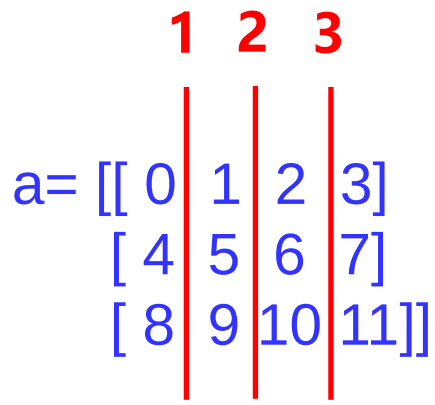
```
b= [array([[0, 1, 2], [3, 4, 5]]),
    array([[ 6,7, 8], [ 9, 10, 11]])]
```

```
array(b)=
 [[[ 0  1  2]
  [ 3  4  5]]

 [[ 6  7  8]
  [ 9 10 11]]]
```

```
c= [array([[0, 1, 2], [3, 4, 5]]),
    array([[6, 7, 8]]),
    array([[ 9, 10, 11]])]
```

[示例] 纵向切分数组

```
import numpy as np

a = np.arange(12).reshape(3,4)
print('a=',a)

b = np.split(a,2,axis=1)   # 沿1轴均切为2份
print('b=',b)

c = np.split(a,[2,3],1) # 沿1轴在2、3位置切分数组
print('c=',c)
```




```
b= [array([[0, 1], [4, 5], [8, 9]]),
    array([[2, 3],[6, 7],[10, 11]])]

c= [array([[0,1], [4, 5],[8, 9]]),
    array([[2],[6],[10]]),
    array([[3],[7],[11]])]
```

2、numpy.hsplit

numpy.hsplit函数用于**水平切分数组**，不能均分则出错。

numpy.hsplit(array, indices_or_sections)

3、numpy.vsplit

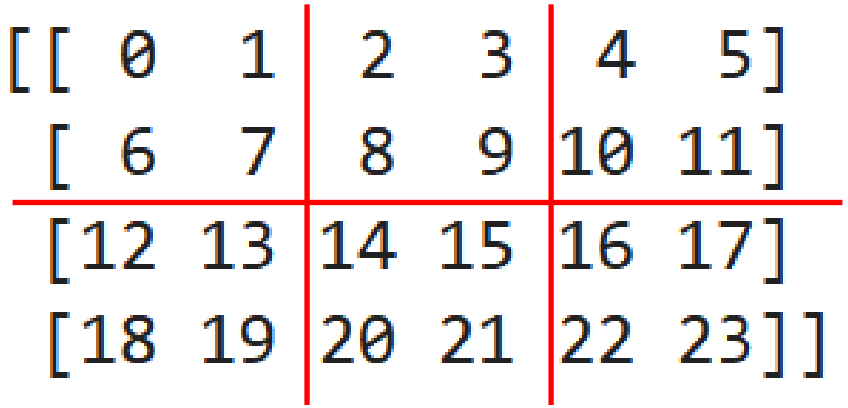
numpy.vsplit沿着**垂直**轴切**分数组**，不能均分则出错。

numpy.hsplit(array, indices_or_sections)

参数：indices_or_sections：如果是一个整数，就用该数平均切分，如果是一个数组，为沿轴切分的位置。

[示例] 数组分割

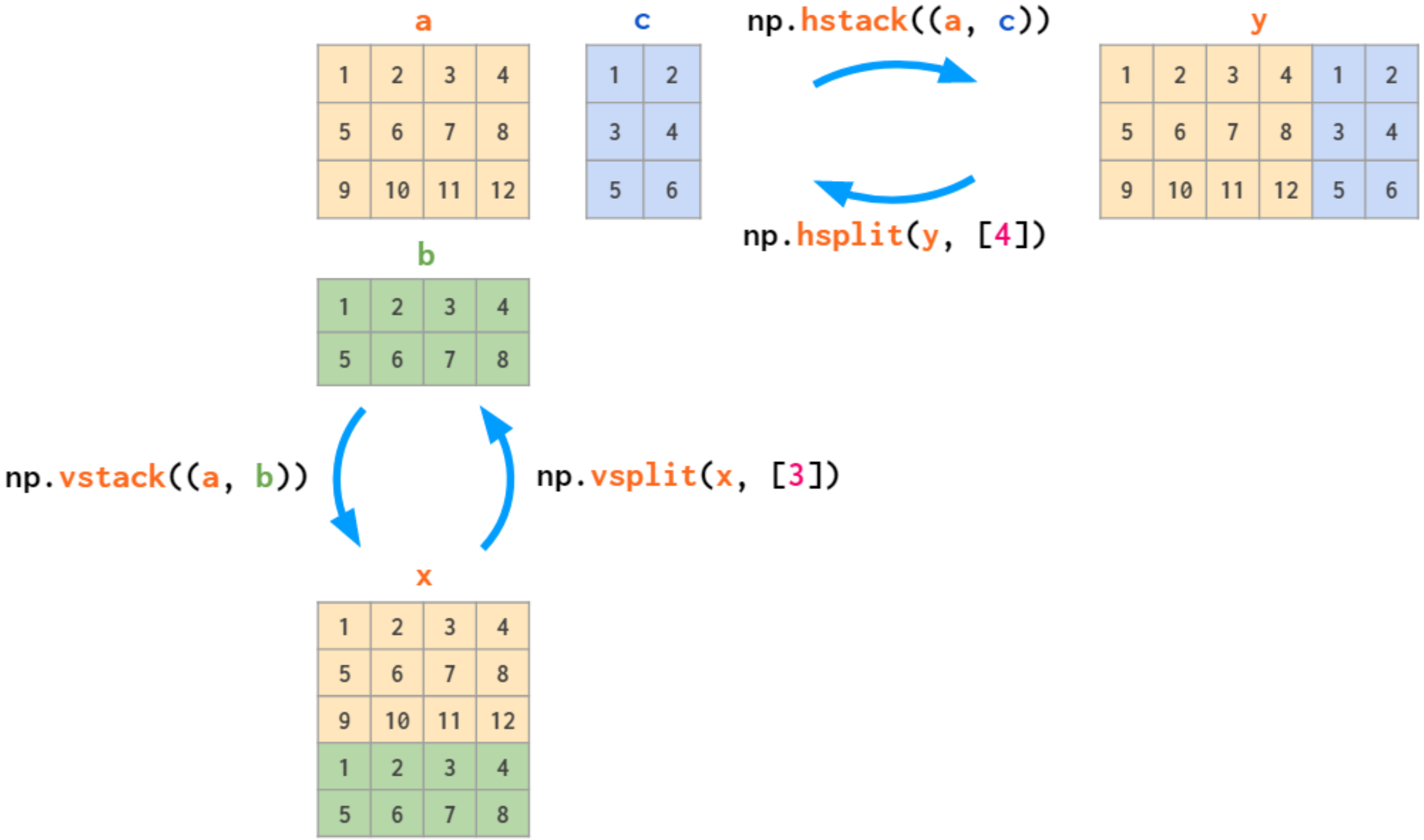
```
import numpy as np

a = np.arange(24).reshape(4,6)
print('a=',a)

d = np.hsplit(a,3)  # 纵向均切为3个子数组
print('d=',d)

e = np.vsplit(a,2)  # 横向均切为2个子数组
print('e=',e)
```




```
d= [array([[ 0,  1], [ 6,  7],
           [12, 13], [18, 19]]),
    array([[ 2,  3], [ 8,  9],
           [14, 15],[20, 21]]),
    array([[ 4,  5],  [10, 11],
           [16, 17], [22, 23]])]
```

```
e= [array([[ 0,  1,  2,  3,  4,  5],
           [ 6,  7,  8,  9, 10, 11]]),
    array([[12, 13, 14, 15, 16, 17],
           [18, 19, 20, 21, 22, 23]])]
```

split与stack对应



2.2.5 数组元素添加与删除

数组元素添加和删除的函数如表所示。

函数	元素及描述
resize	返回指定形状的新数组
append	将值添加到数组末尾
insert	沿指定轴将值插入到指定下标之前
delete	删掉某个轴的子数组，并返回删除后的新数组

1、numpy.resize

按照shape的形状**拓展**arr，**不够的**依次按原数组单个元素排列顺序补全，返回指定大小的新数组。

```
numpy.resize(arr, shape)   # 全局函数，返回新形状数组，不修改原数组arr
arr.resize(shape)          # 成员函数，原地修改arr形状，返回值为None
```

- arr：要修改大小的数组。
- shape：新形状(数组或序列）。

注意：
numpy.resize可以多退少补数据，array.reshape必须和原数组数据量一样

[示例] 改变数组形状

```
import numpy as np

a = np.arange(6).reshape(3,2)
print ("a=\n", a)

a.resize(2,3)  # 改变形状
print ("a=\n", a)

b = np.resize(a,(1,3))   # 截断数据
print ("c=\n", b)

c = np.resize(a,(4,2))   # 添加数据
print ("c=\n", c)
```

```
a=
 [[0 1]
 [2 3]
 [4 5]]

a=
 [[0 1 2]
  [3 4 5]]

c=
 [[0 1 2]]

c=
 [[0 1]
  [2 3]
  [4 5]
  [0 1]]
```

2、numpy.append

- numpy.append函数在数组的末尾添加值，**输入数组的形状必须与原数组匹配**，否则出错。
- append 格式：numpy.append(arr, values, axis=None)
  - arr：输入数组。
  - values：要向arr添加的值，**需要和arr形状**（除了要添加的轴）和**维度相同**。
  - axis：默认为 None。
    - axis为None时，是横向加成，返回一维数组。
    - axis为0时，数据是加在下边（列数要相同）；
    - axis为1时，数组是加在右边（行数要相同）。

[示例] 数组末尾添加元素

```
import numpy as np

b = np.arange(6).reshape(3,2)
print('b=\n',b)

c = np.append(b,[6,7])   # axis无定义，返回一维数组
print('c=\n',c)

d = np.append(b,[[11,12]],axis=0)
print('d=\n',d)

e = np.append(b,[[100],[200],[300]],axis=1)
print('e=\n',e)
```

```
b=
 [[0 1]
  [2 3]
  [4 5]]

c=
 [0 1 2 3 4 5 6 7]

d=
 [[ 0  1]
  [ 2  3]
  [ 4  5]
  [11 12]]

e=
 [[  0   1 100]
  [  2   3 200]
  [  4   5 300]]
```

3、numpy.insert

numpy.insert函数在给定索引之前，沿给定轴在输入数组中插入值。

numpy.insert(arr, index, values, axis=None)

- arr：输入数组。
- index：在其之前插入值的索引，可以是序列数据（如元组、列表），实现在多个索引之前插入数据。
- values：要插入的数据，可以是序列数据，但其形状必须与原数组匹配。
- axis：沿着它插入的轴，**如果未提供，则输入数组会被展开。**

[示例] 指定位置插入元素

```
import numpy as np

b = np.arange(6).reshape(3,2)
print('b=\n',b)

f = np.insert(b,2,11)   # 未给定axis，b被展开为一维数组
print('f=\n',f)

h = np.insert(b,2,10, axis=0)
print('h=\n',h)

i = np.insert(b,1,10, axis=1)  # insert(b,1,(10,10,10),axis=1)
print('i=\n',i)
```

```
b=
 [[0 1]
  [2 3]
  [4 5]]
f=
 [ 0  1 11  2  3  4  5]
h=
 [[ 0  1]
  [ 2  3]
  [10 10]
  [ 4  5]]
i=
 [[ 0 10  1]
  [ 2 10  3]
  [ 4 10  5]]
```

5、numpy.delete

numpy.delete函数返回从输入数组中删除指定子数组的新数组。 与insert()函数的情况一样，**如果未提供轴参数，则输入数组将展开。**

numpy.delete(arr, obj, axis)

- arr：输入数组。
- obj：可取值为**slice（前闭后开）**、 int 、int类型的array、int类型的序列等，指示要沿指定轴删除的子数组的**索引**。
- axis未指定：arr会先按行展开，然后按照obj删除，返回一个行矩阵。
- axis = 0：按行删除
- axis = 1：按列删除

[示例] 删除指定元素

```
import numpy as np
a = np.arange(10,19).reshape(3,3)
print('a=\n',a)
b = np.delete(a,[0,1])  # 没有设置轴参数，展开后删除
print('b=\n',b)
k = np.delete(a,2,axis=0)  # 删除第2行
print('k=\n',k)
l = np.delete(a,2,axis=1)  # 删除第2列
print('l=\n',l)
h = np.delete(a,slice(1,3),axis=1) # 删除1~2列
print("h=\n", h)
i = np.delete(a,[1,2],axis=1)  # 删除1、2列
print("i=\n", i)
```

```
a=
 [[10 11 12]
  [13 14 15]
  [16 17 18]]

b=
 [12 13 14 15 16 17 18]

k=
 [[10 11 12]
  [13 14 15]]

h=
 [[10]
  [13]
  [16]]

l=
 [[10 11]
  [13 14]
  [16 17]]

i=
 [[10]
  [13]
  [16]]
```

第二章 NumPy数值计算

Python

2.3 访问数组元素

2.3 访问数组元素

2.3.1 整数索引

arr[第0维索引, 第1维索引, ...第N维索引]

arr[第0维索引][第1维索引]...[第N维索引]

注意：中括号中的整数索引个数要**小于等于数组维度。**

[示例]一维数组的整数索引

```
import numpy as np

a = np.linspace(-10,10,11)  # 创建一个从-10到10的11长度的等差数组
print('a=', a)

print('a[-1]=', a[-1])

a[2] = 200
print('a=', a)
```

```
a= [-10.  -8.  -6.  -4.  -2.   0.   2.   4.   6.   8.  10.]
a[-1]= 10.0
a= [-10.  -8. 200.  -4.  -2.   0.   2.   4.   6.   8.  10.]
```

[示例] 二维数组的整数索引

```
import numpy as np

a = np.arange(12).reshape(3,4)
print('a=\n',a)

print('a[1,2]=', a[1,2])  # a[1][2]

print('a[1]=', a[1])   # a[1]
```

```
a=
 [[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

a[1,2]= 6

a[1]= [4 5 6 7]
```

[示例] 多维数组的整数索引

注意：整数索引个数M < 数组维度N时，结果为m-n维数组

```
import numpy as np
a = np.arange(12).reshape(3,2,2)
print('a=\n',a)
# 3个整数索引，结果为3-3维数组
print('a[1,0,1]=', a[1,0,1])  # a[1][0][1]
# 2个整数索引，结果为3-2维数组
print('a[2,0]=', a[2,0])   # a[2][0]
# 1个整数索引，结果为3-1维数组
print('a[0]=\n', a[0])
```

```
a=
 [[[ 0  1]
   [ 2  3]]

  [[ 4  5]
   [ 6  7]]

  [[ 8  9]
   [10 11]]]
```

```
a[1,0,1]= 5

a[2,0]= [8 9]

a[0]=
 [[0 1]
  [2 3]]
```

2.3 访问数组元素

2.3.2 花式索引

花式索引（Fancy indexing）是指利用**数组、列表、元组等可迭代类型**进行索引。

（1）列表做花式索引

[示例] 列表做一维数组的花式索引

```
import numpy as np

a = np.arange(12) ** 2
print('a=',a)

print('a[[1,1,3,8]]=', a[[1,1,3,8]])
```

```
a= [  0   1   4   9  16  25  36  49  64  81 100 121]
a[[1,1,3,8]]= [ 1  1  9 64]
```

[示例] 列表做二维数组的花式索引

```
import numpy as np

a = np.arange(9).reshape(3, 3)
print('a=\n',a)

# 1个列表做索引，取a[0]和a[2]
print('a[[0,2]] =\n', a[[0,2]] )

# 2个列表做索引，取a[0,1]、a[2,2]、a[1,1]
print('a[[0,2,1], [1,2,1]] =', a[[0,2,1], [1,2,1]])
```

```
a=
 [[0 1 2]
  [3 4 5]
  [6 7 8]]

a[[0,2]] =
 [[0 1 2]
  [6 7 8]]

a[[0,2,1], [1,2,1]] = [1 8 4]
```

[示例] 列表做三维数组的花式索引

```
import numpy as np
a = np.arange(36).reshape(3,3,4)
print('a=\n',a)
# 1个列表做索引，取a[0]和a[1]
print(a[[0,1]])
# 2个列表做索引，取a[0,2]和a[1,1]
print(a[[0,1],[2,1]])
# 2个列表做索引，取a[0,2]和a[0,1]
print(a[[0],[2,1]])  # 其实进行了广播
# 3个列表做索引，取a[0,1,2]和a[1,2,3]
print(a[[0,1],[1,2],[2,3]])
```

```
a=
 [[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]

 [[24 25 26 27]
  [28 29 30 31]
  [32 33 34 35]]]
```

```
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]

[[ 8  9 10 11]
 [16 17 18 19]]

[[ 8  9 10 11]
 [ 4  5  6  7]]

[ 6 23]
```

（2）numpy数组做花式索引

[示例] 数组做一维数组的花式索引

```
import numpy as np

a = np.arange(12) ** 2   # 形状为(12,)
print('a=', a)
# 形状为(2,2)的数组做索引，结果形状为(2,2)
i = np.arange(0,4).reshape(2,2)
print('i=\n', i)
print('a[i]=\n', a[i])
# 形状为(2,2,2)的数组做索引，结果形状为(2,2,2)
i = np.arange(0,8).reshape(2,2,2) # 形状为(2,2,2)
print('i=\n', i)
print('a[i]=\n', a[i])
```

```
a= [  0   1   4   9  16  25  36  49  64  81 100 121]

i=
 [[0 1]
  [2 3]]

a[i]=
 [[0 1]
  [4 9]]
```

```
i=
 [[[0 1]
   [2 3]]

  [[4 5]
   [6 7]]]

a[i]=
 [[[ 0  1]
   [ 4  9]]

  [[16 25]
   [36 49]]]
```

[示例] 数组做二维数组的花式索引

```
import numpy as np

a = np.arange(12).reshape(4, 3)  # 形状为(4,3)
print('a=\n',a)

# 形状为(2,)的1维数组做索引，结果形状为(2,3)
i = np.array([2, 3])
print('a[i]=\n', a[i])

# 形状为(2,2)的2维数组做索引，结果形状为(2,2,3)
j = np.array([[2,3],[1,3]])
print('a[j]=\n', a[j])
```

```
a=
 [[ 0  1  2]
  [ 3  4  5]
  [ 6  7  8]
  [ 9 10 11]]

a[i]=
 [[ 6  7  8]
  [ 9 10 11]]

a[j]=
 [[[ 6  7  8]
   [ 9 10 11]]

  [[ 3  4  5]
   [ 9 10 11]]]
```

2.3 访问数组元素

2.3.3 切片访问多个数组元素

arr[start，end，step]

[示例] 一维数组切片

```
import numpy as np


a = np.arange(12)
print('a=',a)
print('a[1:3]=',a[1:3])
print('a[9:]=',a[9:])
print('a[1:7:2]=',a[1:7:2])
print('a[:]=',a[:])
```

```
a= [ 0  1  2  3  4  5  6  7  8  9 10 11]
a[1:3]= [1 2]
a[9:]= [ 9 10 11]
a[1:7:2]= [1 3 5]
a[:]= [ 0  1  2  3  4  5  6  7  8  9 10 11]
```

[示例] 二维数组切片

```
import numpy as np

a = np.arange(12).reshape(3,4)
print('a=\n', a)

# 0轴取1:2
print('a[1:2]=\n', a[1:2])

# 0轴取1:2, 1轴取1:3
print('a[1:2, 1:3] =\n', a[1:2, 1:3])

# 0轴取0:2,1轴取1:3
print("a[:2,1:3] =\n", a[:2,1:3] )
```

```
a=
 [[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

a[1:2]=
 [[4 5 6 7]]

a[1:2, 1:3] =
 [[5 6]]

a[:2,1:3] =
 [[1 2]
  [5 6]]
```

[示例] 三维数组切片

```
import numpy as np

a = np.arange(27).reshape(3,3,3)
print('a=\n',a)

print('a[1:3]=\n',a[1:3])

print('a[:2,1:2]=\n',a[:2,1:2])

print('a[:2,1:2,:2]=\n',a[:2,1:2,:2])
```

```
a=
 [[[ 0  1  2]
  [ 3  4  5]
  [ 6  7  8]]

 [[ 9 10 11]
  [12 13 14]
  [15 16 17]]

 [[18 19 20]
  [21 22 23]
  [24 25 26]]]
```

```
a[1:3]=
 [[[ 9 10 11]
   [12 13 14]
   [15 16 17]]

  [[18 19 20]
   [21 22 23]
   [24 25 26]]]

a[:2,1:2]=
 [[[ 3  4  5]]

  [[12 13 14]]]

a[:2,1:2,:2]=
 [[[ 3  4]]

  [[12 13]]]
```

2.3 访问数组元素

2.3.4 整数和切片混合访问多个数组元素

[示例] 二维数组切片混合整数索引

```
import numpy as np

a = np.arange(12).reshape(3,4)
print('a=\n', a)

#混合传入切片和索引，0轴取:2, 1轴取1
print("a[:2,1] =\n", a[:2,1] )

# 0轴取2, 1轴取1:4
print("a[2,1:4] =\n", a[2,1:4])
```

```
a=
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

a[:2,1] =
 [1 5]

a[2,1:4] =
 [ 9 10 11]
```

[示例] 三维数组切片混合整数索引

```
import numpy as np
a = np.arange(27).reshape(3,3,3)
print('a=\n',a)

print('a[0,0,:]=\n', a[0,0,:])

print('a[0,0,1:3]=\n', a[0,0,1:])

print('a[0,1:]=\n', a[0,1:])
```

```
a=
 [[[ 0  1  2]
  [ 3  4  5]
  [ 6  7  8]]

 [[ 9 10 11]
  [12 13 14]
  [15 16 17]]

 [[18 19 20]
  [21 22 23]
  [24 25 26]]]
```

```
a[0,0,:]=
 [0 1 2]

a[0,0,1:3]=
 [1 2]

a[0,1:]=
 [[3 4 5]
  [6 7 8]]
```

[示例] 三维数组切片混合整数索引（取出的数据为二维）

```
import numpy as np

a = np.arange(1,7).reshape(2,3)
b = np.arange(11,17).reshape(2,3)
c = np.stack((a, b), axis=0)
print('c=\n',c)

print('从c中取出a:\n', c[0, :])
print('从c中取出b:\n', c[1, :])

c = np.stack((a, b), axis=1)
print('c=\n', c)
print('从c中取出a:\n', c[:, 0])
print('从c中取出b:\n', c[:, 1])
```

```
c=
 [[[ 1  2  3]
  [ 4  5  6]]

 [[11 12 13]
  [14 15 16]]]

从c中取出a:
 [[1 2 3]
 [4 5 6]]

从c中取出b:
 [[11 12 13]
 [14 15 16]]
```

```
c=
 [[[ 1  2  3]
  [11 12 13]]

 [[ 4  5  6]
  [14 15 16]]]

从c中取出a:
 [[1 2 3]
 [4 5 6]]

从c中取出b:
 [[11 12 13]
 [14 15 16]]
```

思考：如何创建一个10*10的ndarray对象，且矩阵边界全为1，里面全为0

```
import numpy as np


a = np.ones(shape=(10,10),dtype=int)
a[1:-1,1:-1]=0
print(a)
```

```
[[1 1 1 1 1 1 1 1 1 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 1 1 1 1 1 1 1 1 1]]
```

思考：如何创建一个10*10的ndarray对象，且矩阵边界全为1，里面全为0

```
import numpy as np


b = np.zeros(shape=(10,10),dtype=int)
b[[0,-1]] = 1
b[:,0] = 1
b[:,-1] = 1
print(b)
```

```
[[1 1 1 1 1 1 1 1 1 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 1 1 1 1 1 1 1 1 1]]
```

思考：如何创建一个10*10的ndarray对象，且矩阵边界全为1，里面全为0

```
import numpy as np


c = np.zeros(shape=(8,8),dtype=int)
d = np.ones(shape=(8,1),dtype=int)
e = np.ones(shape=(10,),dtype=int)
f = np.hstack((c,d))
f = np.hstack((d,f))
f = np.vstack((e,f))
f = np.vstack((f,e))
print(f)
```

```
[[1 1 1 1 1 1 1 1 1 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0 0 1]
 [1 1 1 1 1 1 1 1 1 1]]
```

2.3 访问数组元素

2.3.5 布尔型索引

布尔型索引是指使用**布尔数组**来索引目标数组，以此找出与布尔数组中**值为True的对应的**目标数组中的数据。

用布尔索引总是会返回一份新创建的数据，原本的数据不会被改变。

[示例] 布尔索引

```
import numpy as np


a = np.arange(12).reshape(3,4)
print('a=\n',a)
mask = a>5
print('mask=\n', mask)
print('a[mask]=\n', a[mask])  # a[a>5]
```

```
a=
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

mask=
 [[False False False False]
  [False False  True  True]
  [ True  True  True  True]]

a[mask]=
 [ 6  7  8  9 10 11]
```

[示例] 布尔索引选择一个数

问题：孙对JD的评分是多少？

```
import numpy as np
from numpy.random import randint

n = np.array(['赵','钱','孙'])
s = np.array(['淘宝','拼多多','京东'])
# 3*3数组，模拟以上6人对三家店的评分
data=randint(1,6,(3,3))
print(data)

print(n=='孙')
print(s=='京东')
# 0轴取n=='孙'的true，1轴取s=='京东'的true
print('孙对京东的评分：', data[n=='孙', s=='京东'])
```

```
[[5 1 4]
 [3 4 3]
 [3 1 3]]
[False False  True]
[False False  True]
孙对京东的评分： [3]
```

[示例] 利用布尔索引，还可以把满足条件的位置的值替换成其他的值。

```
import numpy as np
from numpy.random import randint

n = np.array(['赵','钱','孙'])
s = np.array(['淘宝','拼多多','京东'])
data=randint(1,6,(3,3))
print(data)
data[n=='孙', s=='淘宝'] = 4
s[s=='京东'] = '抖音'
print('data=\n',data)
print('stores=\n',s)
```

```
[[1 2 1]
 [5 2 1]
 [1 1 2]]
data=
 [[1 2 1]
 [5 2 1]
 [4 1 2]]
stores=
 ['淘宝' '拼多多' '抖音']
```

[示例] 布尔索引与切片混用

```
import numpy as np
from numpy.random import randint

n = np.array(['赵','钱','孙'])
s = np.array(['淘宝','拼多多','京东'])
data=randint(1,6,(3,3))
print(data)

print('孙对淘宝和京东的评分：', data[n=='孙',[0,2]])
print('钱对拼多多和京东的评分：', data[n=='钱',1:])
```

```
[[2 5 5]
 [4 1 3]
 [5 2 1]]
孙对淘宝和京东的评分： [5 1]
钱对拼多多和京东的评分： [[1 3]]
```

END