NumPy数值计算

Python

数组的运算

1 数组和标量间的运算

数组与标量的算术运算，以及**相同维度的数组**的算术运算都是直接应用到元素中，也就是**元素级**运算。

[示例] 数组和标量运算

```
import numpy as np

a = np.arange(6)
print('a=',a)

b = a*10
print('b=',b)

c = a.reshape(2,3)
print('c=',c)

d = c*100
print('d=',d)
```

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

b= [ 0 10 20 30 40 50]

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

d= [[  0  100  200]
    [300 400 500]]
```

[示例] 每日票房

数组中保存了某一部电影连续5天的票房记录，请输出每日票房收入

```
import numpy as np
a = np.array([1233, 4578, 13211, 43328, 50995])
print('a[1:] =  ',a[1:])
print('a[:-1] = ',a[:-1])
b = a[1:] - a[:-1]
print('每日票房=', b)
```

```
a[1:] =     [ 4578 13211 43328 50995]
a[:-1] =    [ 1233  4578 13211 43328]
每日票房= [ 3345  8633 30117  7667]
```

2 广播

- Numpy中的基本运算（加、减、乘、除、求余等等）都是元素级别的，但是这仅仅局限于**两个数组的形状相同**的前提下。

- **广播**是指NumPy在算术运算期间处理不同形状的数组的能力。如果两个数组的维数不相同，numpy将会自动触发广播机制**较小的数组会被广播到较大数组的大小**，以便使它们的形状可兼容。

- 若形状不兼容，则不能进行运算。

- 两个数组从后缘维度（从末尾开始算起的维度）开始一一比较，若**两者的长度相等，或其中一方的长度为1**，则认为他们是广播兼容的。
  - shape为（2，3，3）的数组能够和（3，2）的数组广播兼容吗？No
  - shape为（3，3，2）的数组能够和（3，2）的数组广播兼容吗？Yes
  - shape为（2，3，2）的数组能够和（3，1）的数组广播兼容吗？Yes
  - shape为（2，3，2）的数组能够和（1，2）的数组广播兼容吗？Yes
  - shape为（4，3，2）的数组能够和（1，1）的数组广播兼容吗？Yes
- 换句话说，就是两个形状甚至维度不同的数组进行运算，会在**缺失或长度为1**的维度的那个轴上延生补齐到大小相等，然后再做计算。

[示例] 广播

```
import numpy as np
a = np.ones((2,3))
b = np.arange(3)
print('a=',a)
print('b=',b)
print("a+b=",a+b)
```

```
a= [[1. 1. 1.]
    [1. 1. 1.]]

b= [0 1 2]

a+b= [[1. 2. 3.]
      [1. 2. 3.]]
```

a

1.	1.	1.
1.	1.	1.

b

0	1	2
0	1	2

a的shape： (2,3)
b的shape： (3,)
b最后一维为3，a与b广播兼容，b广播为shape(2,3)的数组再进行计算

[示例] 广播

```
import numpy as np
a = np.array([1,2,3])
b = np.array([[4],[5],[6]])
print('a=',a)
print('b=',b)
print("a+b=",a+b)
```

```
a= [1 2 3]

b= [[4]
    [5]
    [6]]

a+b= [[5 6 7]
      [6 7 8]
      [7 8 9]]
```

a

1	2	3
1	2	3
1	2	3

b

4	4	4
5	5	5
6	6	6

a的shape： (3,)
b的shape: (3,1)
b最后一维为1，a与b广播兼容，b广播为(3,3)，a广播为(3,3)

[示例] 广播

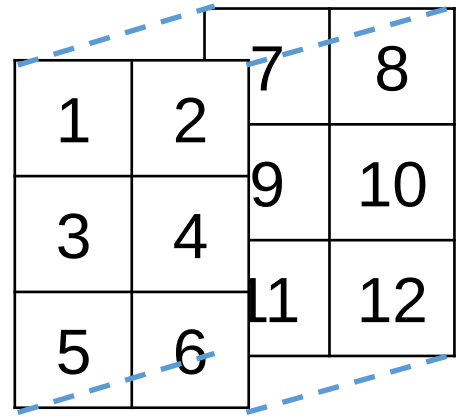
```
import numpy as np
a = np.arange(1,13).reshape(2,3,2)
b = np.array([[4],[5],[6]])   # shape(3,1)
print('a=',a)
print('b=',b)
print("a+b=",a+b)
```

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

    [[ 7  8]
     [ 9 10]
     [11 12]]]
```

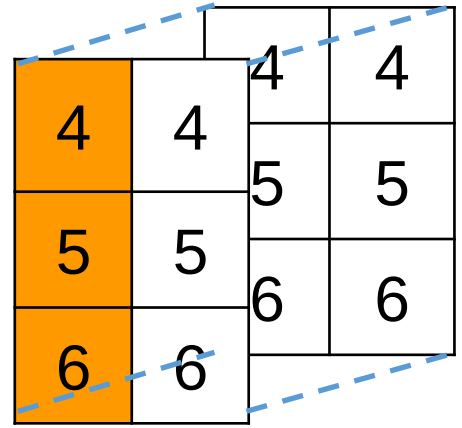
```
b= [[4]
    [5]
    [6]]
```

```
a+b= [[[ 5  6]
       [ 8  9]
       [11 12]]

      [[11 12]
       [14 15]
       [17 18]]]
```

a



b



3 numpy全局算术函数

函数	描述
abs, fabs	计算绝对值
sqrt	计算元素的平方根
square	计算元素的平方
exp	计算以自然常数e为底的幂次方
log, log10, log2, log1p	log以自然对数为底, log1p(x)即ln(1+x)
sign	计算元素的符号：1：正数 0：0 -1：负数
ceil	计算大于或等于元素的最小整数
floor	计算小于或等于元素的最大整数
around	四舍六入五留双到给定的小数位
rint	四舍六入五留双到整数
modf	分别返回浮点数的小数和整数数部分的数组

[示例] around函数

```
import numpy as np
a = np.array([1.2, 15.55, 123.45])
print('原数组：  ',a)
print('保留1位小数:',np.around(a, decimals = 1))
print('保留0位小数:',np.around(a))  # decimals 默认值为0
print('保留-1位小数:',np.around(a, decimals = -1))
print('保留-2位小数:',np.around(a, decimals = -2))
```

```
原数组:          [  1.2    15.55  123.45    0.537 125.32 ]
保留1位小数:  [  1.2    15.6    123.4      0.5     125.3   ]
保留0位小数:  [  1.      16.      123.       1.       125.    ]
保留-1位小数: [  0.      20.      120.       0.       130.    ]
保留-2位小数: [  0.      0.       100.       0.       100.    ]
```

[示例] rint、sign、modf函数

```
import numpy as np

a = np.array([[1.2,2.5,3.6],
                    [-4.3,5.5,-6.2]])
print('a=\n',a)

b = np.rint(a)
print('b=\n',b)

c = np.sign(a)
print('c=\n',c)

d = np.modf(a)
print('d=\n',d)
```

```
a=
 [[ 1.2  2.5  3.6]
  [-4.3  5.5 -6.2]]

b=
 [[ 1.  2.  4.]
  [-4.  6. -6.]]

c=
 [[ 1.  1.  1.]
  [-1.  1. -1.]]

d=
 (array([[ 0.2,  0.5,  0.6],
         [-0.3,  0.5, -0.2]]),
  array([[ 1.,  2.,  3.],
         [-4.,  5., -6.]]))
```

函数	描述
isnan	返回布尔数组标识哪些元素是 NaN （不是一个数）
isfinite isinf	判断元素是有限的数 判断元素是否无限大

[示例] isnan函数

```
import numpy as np

lst= [[4, 9, 16], [-4, 16, -9]]
a = np.array(lst)
print('a=\n', a)

b = np.sqrt(a)
print('b=\n', b)

c = np.isnan(b)
print('c=\n', c)
```

```
a=
 [[ 4   9  16]
  [-4  16  -9]]

b=
 [[ 2.   3.   4. ]
  [nan  4.   nan]]

c=
 [[False False False]
  [ True False  True]]
```

[示例] isinf函数

```
import numpy as np

a = np.arange(12).reshape(3,-1)
print(a)

b = 10/a
print(b.round(1))

c = np.isinf(b)
print(c)

n = np.argwhere(c==True)
print(n)
```

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

[[ inf  10.   5.   3.3]
 [ 2.5  2.   1.7  1.4]
 [ 1.2  1.1  1.   0.9]]

[[ True False False False]
 [False False False False]
 [False False False False]]

[[0 0]]
```

np.argwhere获取 Numpy 数组中指定元素的索引位置

函数	描述
add（+）	数组对应元素相加
substract（-）	数组对应元素相减
multiply（*）	点积，矩阵对应位置相乘，必要时使用广播规则
divide（/）	除法
mod（%）	模运算（求余）
power	用第二个数组作为指数，计算第一个数组中的元素的幂

[示例] add函数

```
import numpy as np


a = np.array([[0,1,2],[3,4,5]])
b = np.array([2,2,2])   # 计算时会广播为2*3的数组


print('add(a,b)=\n', np.add(a,b))
print('a+b=\n', a+b)
```

a

0	1	2
3	4	5

b

2	2	2
2	2	2

```
add(a,b)=
 [[2 3 4]
  [5 6 7]]

a+b=
 [[2 3 4]
  [5 6 7]]
```

[示例] subtract函数

```
import numpy as np


a = np.array([[0,1,2],[3,4,5]])
b = np.array([2,2,2])


print('subtract(a,b)=\n', np.subtract(a,b))
print('a-b=\n', a-b)
```

a

0	1	2
3	4	5

b

2	2	2
2	2	2

```
subtract(a,b)=
 [[-2 -1  0]
  [ 1  2  3]]

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

[示例] multiply函数

```
import numpy as np


a = np.array([[0,1,2],[3,4,5]])
b = np.array([2,2,2])


print('multiply(a,b)=\n', np.multiply(a,b))
print('a*b=\n', a*b)
```

a

0	1	2
3	4	5

b

2	2	2
2	2	2

```
multiply(a,b)=
 [[ 0  2  4]
  [ 6  8 10]]

a*b=
 [[ 0  2  4]
  [ 6  8 10]]
```

[示例] divide函数

```
import numpy as np


a = np.array([[0,1,2],[3,4,5]])
b = np.array([2,2,2])


print('divide(a,b)=\n', np.divide(a,b))
print('a/b=\n', a/b)
```

a

0	1	2
3	4	5

b

2	2	2
2	2	2

```
divide(a,b)=
 [[0.   0.5  1. ]
  [1.5  2.   2.5]]

a/b=
 [[0.   0.5  1. ]
  [1.5  2.   2.5]]
```

[示例] 求余函数

```
import numpy as np


a = np.array([[0,1,2],[3,4,5]])
b = np.array([2,2,2])


print('mod(a,b)=\n', np.mod(a,b))
print('a%b=\n', a%b)
```

a

0	1	2
3	4	5

b

2	2	2
2	2	2

```
mod(a,b)=
 [[0 1 0]
  [1 0 1]]

a%b=
 [[0 1 0]
  [1 0 1]]
```

函数	描述
minimum maximum	两数组**对应元素**比大小取其小者 两数组**对应元素**比大小取其大者
copysign	将第二个数组中各元素的符号赋值给第一个数组的对应元素
greater greater_equal less less_equal equal not_equal	基于元素的比较，产生布尔数组。 等价于>，>=，<，<=，==，!=

[示例] 比较函数

```
import numpy as np


a = np.array([[0,1,2],[3,4,5]])
b = np.array([2,2,-2])


print('minimum(a,b)=\n', np.minimum(a,b))
print('maximum(a,b)=\n', np.maximum(a,b))
print('copysign(a,b)=\n', np.copysign(a,b))
print('greater(a,b)=\n', np.greater(a,b))
```

a

0	1	2
3	4	5

b

2	2	-2
2	2	-2

```
minimum(a,b)=
 [[ 0  1 -2]
  [ 2  2 -2]]
maximum(a,b)=
 [[2 2 2]
  [3 4 5]]
copysign(a,b)=
 [[ 0.  1. -2.]
  [ 3.  4. -5.]]
greater(a,b)=
 [[False False  True]
  [ True  True  True]]
```

4 集合运算

函数	使用说明
unique(x)	唯一值
intersect1d(x,y)	交集（输出结果的顺序未规定）
union1d(x,y)	并集
isin(x,y)	x的元素是否在y中，返回布尔型数组

1、unique：唯一值

np.unique方法用于找出数组中的**唯一值**，并返回已**排序**的结果，格式如下：

numpy.unique(arr, return_index, return_inverse, return_counts)

- arr：输入数组，如果不是一维数组则会**展开成一维数组**。
- return_index：若为true，返回新数组元素在旧数组中的位置（下标），并以列表形式存储。
- return_inverse：若为true，返回旧数组元素在新数组中的位置（下标），并以列表形式储。
- return_counts：若为true，返回去重数组中的元素在原数组中的出现次数。

[示例] unique函数（仅去重）

```
import numpy as np


a = np.array([1, 3, 3, 3, 2, 2, 4, 4, 4, 4])
print ('去重后的数组: ',np.unique(a))  # 唯一值并排序
```

去重后的数组: [1 2 3 4]

[示例] unique函数（返回去重数组和索引数组）

```
import numpy as np
a = np.array([1, 3, 3, 3, 2, 2, 4, 4, 4, 4])
u, index = np.unique(a, return_index = True)
print('去重后的新数组: ', u)
print ('新数组元素在旧数组中的位置：', index)
print('从旧数组得到新数组：', a[index])
```

```
去重后的新数组:  [1 2 3 4]
新数组元素在旧数组中的位置：  [0 4 1 6]
从旧数组得到新数组：  [1 2 3 4]
```

[示例] unique函数（返回去重数组和反索引数组）

```
import numpy as np
a = np.array([1, 3, 3, 3, 2, 2, 4, 4, 4, 4])
u, inverse = np.unique(a, return_inverse = True)
print('去重后的新数组: ',u)
print('旧数组元素在新数组中的位置：',inverse)
print('从新数组得到旧数组', u[inverse])
```

```
去重后的新数组:  [1 2 3 4]
旧数组元素在新数组中的位置：  [0 2 2 2 1 1 3 3 3 3]
从新数组得到旧数组 [1 3 3 3 2 2 4 4 4 4]
```

[示例] unique函数（返回去重数组和在原数组中的重复次数）

```
import numpy as np


a = np.array([1, 3, 3, 3, 2, 2, 4, 4, 4, 4])
u, count_arr = np.unique(a, return_counts = True)
print ('去重后的新数组: ', u)
print ('去重数组中的元素在原数组中的重复数量：', count_arr)
```

```
去重后的新数组: [1 2 3 4]
去重数组中的元素在原数组中的重复数量： [1 2 3 4]
```

2、isin：是否包含

函数返回一个与ar1长度相同的布尔数组。若**ar1的元素在ar2中**有，该数组元素为True，否则为False。

numpy.isin(ar1, ar2, assume_unique=False, invert=False)

- assume_unique： bool, 可选。如果为True，则假定数组元素都是唯一值，这可以加快计算速度。 默认值为False。
- invert： bool, 可选。如果为True，则返回的数组中的值将被反转，即，在ar1中的一个元素位于ar2中时为False，否则为True。 默认值为False。

[示例] isin函数

```
import numpy as np

x = np.array([1,2,3,4,5,6]).reshape(2,3)
print(x)

y = np.array([1,3,5,7])
print(y)

print(np.isin(x, y))

print(np.isin(y, x))
```

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

[1 3 5 7]

[[ True False  True]
 [False  True False]]

[ True  True  True False]
```

[示例] 集合函数

```
import numpy as np

x = np.array([1,2,3,4,5,6]).reshape(2,3)
print(x)

y = np.array([1,3,5,7])
print(y)

print(np.intersect1d(x,y))
print(np.union1d(x,y))
print(np.setxor1d(x,y))  # 交集取反（并集-交集）
```

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

[1 3 5 7]

[1 3 5]

[1 2 3 4 5 6 7]

[2 4 6 7]
```

5 统计运算

NumPy库支持对整个数组或按指定轴向的数据进行统计计算。统计函数有axis参数，用于计算指定轴方向的统计值，**axis默认值为None，此时把数组当成一维数组。**

方法	**使用说明**
sum	求和
mean	算数平均数
std、var	标准差、方差(反映数据的离散程度)
min、max	最小值和最大值
argmin、argmax	最小和最大元素的索引
cumsum	所有元素的累计和
cumprod	所有元素的累计积

[示例] 统计函数

```
import numpy as np
a = np.arange(9).reshape(3,3)
print(a)
print('sum=',np.sum(a))
print('mean=',np.mean(a))
print('std=',np.std(a))
print('var=',np.var(a))
print('argmin=',np.argmin(a))
print('argmax=',np.argmax(a))
print('cumsum=',np.cumsum(a))
print('cumprod=',np.cumprod(a))
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
sum= 36
mean= 4.0
std= 2.581988897471611
var= 6.666666666666667
argmin= 0
argmax= 8
cumsum= [ 0  1  3  6 10 15 21 28 36]
cumprod= [0 0 0 0 0 0 0 0 0]
```

$$\text{std} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2} \qquad \text{var} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

累计和，新数组的每一个元素都是原数组此位置及前面所有项之和

[示例] 统计函数（对数组某一轴进行统计）

```
import numpy as np
a = np.array([[1,3,2],[-1,7,4]])
print(a)
print('第0轴sum=',np.sum(a,0))
print('第1轴mean=',np.mean(a,1))
np.set_printoptions(precision=3)
print('第0轴std=',np.std(a,0))
print('第1轴var=',np.var(a,1))
print('第0轴argmin=',np.argmin(a,0))
print('第1轴argmax=',np.argmax(a,1))
print('第0轴cumsum=',np.cumsum(a,0))
print('第1轴cumprod=',np.cumprod(a,1))
```

```
[[ 1  3  2]
 [-1  7  4]]
第0轴sum= [ 0 10  6]
第1轴mean= [2.         3.33333333]
第0轴std= [1. 2. 1.]
第1轴var= [ 0.667 10.889]
第0轴argmin= [1 0 0]
第1轴argmax= [1 1]
第0轴cumsum= [[ 1  3  2]
              [ 0 10  6]]
第1轴cumprod= [[  1   3   6]
               [ -1  -7 -28]]
```

6 排序

1、numpy.sort()

函数sort()返回输入数组的排序副本，格式如下：

numpy.sort(a, axis=-1, kind= None, order=None)

- axis：指定沿哪个轴进行排序。
  - 默认值：-1（最后一个轴）。
  - 如果 axis=None，则数组会先展平（变为 1 维），再排序。
- kind：指定排序算法，可选值包括：
  - 'quicksort'（快速排序，默认）
  - 'mergesort'（归并排序，稳定）
  - 'heapsort'（堆排序）
  - 'stable'（稳定排序，自动选择算法）
- order
  - 当 a 是结构化数组时，指定按哪个字段排序。
  - 默认值：None（按数组的自然顺序排序）。

[示例] 二维数组排序

```
import numpy as np

a = np.array([[2,5,6,1],[8,6,4,3],[6,4,9,0]])
print('a=\n',a)

# 展开后排序
print('axis=None:\n',np.sort(a, axis=None))

# 沿着最后的轴（1轴）排序
print('axis=-1:\n',np.sort(a))

# 沿着第0轴排序
print('axis=0:\n', np.sort(a, axis=0))
```

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

axis=None:
 [0 1 2 3 4 4 5 6 6 6 8 9]

axis= -1:
 [[1 2 5 6]
  [3 4 6 8]
  [0 4 6 9]]

axis=0:
 [[2 4 4 0]
  [6 5 6 1]
  [8 6 9 3]]
```

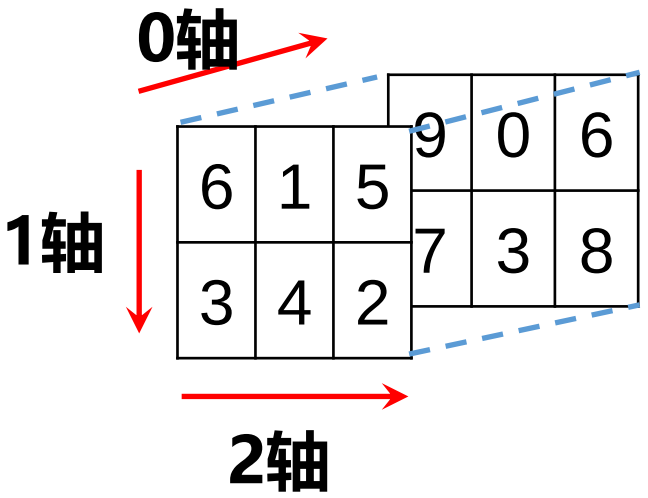
[示例] 三维数组排序

```
import numpy as np

b = np.array([[[6,1,5],[3,4,2]],[[9,0,6],[7,3,8]]])
print("b=", b)
print('b排序(沿0轴):',np.sort(b,0))
print('b排序(沿1轴):',np.sort(b,1))
print('b排序(沿2轴):',np.sort(b,2))
```

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

    [[9 0 6]
     [7 3 8]]]
b排序(沿0轴): [[[6 0 5]
               [3 3 2]]

              [[9 1 6]
               [7 4 8]]]
b排序(沿1轴): [[[3 1 2]
               [6 4 5]]

              [[7 0 6]
               [9 3 8]]]
b排序(沿2轴): [[[1 5 6]
               [2 3 4]]

              [[0 6 9]
               [3 7 8]]]
```

数组b的形状： **(2, 2, 3)**

对应维度序号： **0 1 2**



[示例] 数组按字段排序

```
import numpy as np
# 创建一个结构列表
dt = [('name', 'S5'), ('height', float), ('age', int)]
values = [('Tom', 1.8, 21), ('Tony', 1.9, 28),('Jim', 1.7, 18)]
a = np.array(values, dtype=dt)
b = np.sort(a, order='height')   # 按height字段排序
print('a=',a)
print('b=',b)
```

```
a= [(b'Tom', 1.8, 21)  (b'Tony', 1.9, 28)  (b'Jim', 1.7, 18)]
b= [(b'Jim', 1.7, 18)   (b'Tom', 1.8, 21)   (b'Tony', 1.9, 28)]
```

前缀为b的字符串是字节字符串,内容存储的是原始字节数据,不能直接显示中文等字符。

2、numpy.argsort()

返回数组值从小到大的**索引值**。

numpy.argsort(a, axis=-1, kind='quicksort', order=None)

参数：

- a为要排序的数组
- axis：按哪一维进行排序，默认-1，the last axis。
- kind：排序算法的选择，有quicksort，mergesort，heapsort

[示例] 获得一维数组的排序索引

```
import numpy as np

x = np.array([6, 3, 4])
print('sort_x=', np.sort(x))
print('argsort(x)=',np.argsort(x))  # 返回排序数组在原数组中的序号，默认升序
print('argsort(-x)=',np.argsort(-x)) # -x按升序排列，实际x按降序排列
print(x[np.argsort(-x)])      # 按索引排列x的元素，实现了降序排列
```

```
sort_x= [3 4 6]
argsort(x)= [1 2 0]
argsort(-x)= [0 2 1]
[6 4 3]
```

[示例] 获得二维数组的0轴排序索引

```
import numpy as np


x = np.array([[0,5,6], [2,1,3]])
print('x=', x)
print('sort_x=', np.sort(x,axis=0))
print('argsortx=', np.argsort(x, axis=0))
```

```
x =          [[0 5 6]
              [2 1 3]]

sort_x=      [[0 1 3]
              [2 5 6]]

argsortx= [[0 1 1]
           [1 0 0]]
```

[示例] 获得二维数组的1轴排序索引

```
import numpy as np


x = np.array([[0,5,6], [2,1,3]])
print('x=', x)
print('sort_x=', np.sort(x,axis=1))
print('argsortx=', np.argsort(x, axis=1))
```

```
x=          [[0 5 6]
             [2 1 3]]

sort_x=     [[0 5 6]
             [1 2 3]]

argsortx=   [[0 1 2]
             [1 0 2]]
```

[示例] 按某一列对多维数组排序

```
import numpy as np


data = np.array([[3, 1, 5], [1, 2, 4], [2, 3, 6]])
print(data)
print()
sorted_indices = np.argsort(data[:, 2])  # 按第3列排序
sorted_data = data[sorted_indices]
print(sorted_data)
```

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

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

[示例] 按整数字段排序的索引

```
import numpy as np

dt = np.dtype([('sno', 'int16'),('score', 'int8')])
a = np.array([(101,56),(102,98),(103,72),(104,88),(105,65)],dtype=dt)
print('a=',a)
print('按score排序:',np.sort(a, order='score'))
c = np.argsort(a, order='score')
print('排序索引：',c)        # c[0]为最低分数在a中的序号，c[-1]为最高分数在a中的序号
print('最高分数：{}，学号：{}'.format(a[c[-1]][1], a[c[-1]][0]))
```

```
a= [(101, 56) (102, 98) (103, 72) (104, 88) (105, 65)]
按score排序: [(101, 56) (105, 65) (103, 72) (104, 88) (102, 98)]
排序索引：  [0 4 2 3 1]
最高分数：98，学号：102
```

7 搜索

1、numpy.where()

where()有两种形式。

（1）np.where(condition)

参数只有条件condition，**输出满足条件元素的坐标（索引）**。这里的坐标以tuple的形式给出，通常原数组有多少维，输出的tuple中就包含几个数组，**分别对应符合条件元素的各维坐标。**

（2）np.where(condition, x, y)

满足条件condition输出x，不满足输出y。

[示例] 条件搜索一维数组

```
import numpy as np

a = np.array([2,4,6,8,10])
print(np.where(a > 5))          # 返回由索引数组组成的对象
print(a[np.where(a > 5)] ) # 等价于 a[a>5]
```

```
(array([2, 3, 4], dtype=int64),)
[ 6  8 10]
```

a数组中比5大的元素有6，8，10，它们在a中的索引依次为2，3，4

[示例] 条件搜索二维数组

```
import numpy as np
x = np.arange(9.).reshape(3,3)
print('x=', x)
print('大于 3 的元素的索引：',np.where(x > 3))
print('大于 3 的元素：', x[np.where(x>3)])
print(np.where(x>3, '大', '小'))
```

```
x= [[0. 1. 2.]
    [3. 4. 5.]
    [6. 7. 8.]]
大于 3 的元素的索引：  (array([1, 1, 2, 2, 2], dtype=int64),
                        array([1, 2, 0, 1, 2], dtype=int64))
大于 3 的元素：  [4. 5. 6. 7. 8.]
[['小' '小' '小']
 ['小' '大' '大']
 ['大' '大' '大']]
```

x数组中比3大的元素：4的坐标为（1,1），5的坐标为（1,2）...

[示例] 条件搜索三维数组

```
import numpy as np
a = np.arange(27).reshape(3,3,3)
print('a=', a)
print(np.where(a >= 20))
```

```
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]]]
```

```
(array([2, 2, 2, 2, 2, 2, 2], dtype=int64),
 array([0, 1, 1, 1, 2, 2, 2], dtype=int64),
 array([2, 0, 1, 2, 0, 1, 2], dtype=int64))
```

a数组中大于等于20的元素：20的坐标为（2,0,2），21的坐标为（2,1,0）...

2、numpy.extract(condition, arr)

与where函数相似，但extract函数是返回满足条件的**元素**，而不是元素索引。

3、numpy.nonzero()

返回输入数组中非零元素的**索引**。

[示例] 条件搜索三维数组

```
import numpy as np
a = np.array([[0,2,3],[4,0,1],[2,0,0]])
print('a=', a)
print('a中能整除2的数：',np.extract(np.mod(a,2)==0, a))   # 也可以a%2==0
print('a中非0值的位置：',np.nonzero(a))
print('a中非零值：',a[np.nonzero(a)])
```

```
a= [[0 2 3]
    [4 0 1]
    [2 0 0]]
a中能整除2的数：  [0 2 4 0 2 0 0]
a中非0值的位置：  (array([0, 0, 1, 1, 2], dtype=int64),
                   array([1, 2, 0, 2, 0], dtype=int64))
a中非零值：  [2 3 4 1 2]
```

END