创建ndarray对象

通过 NumPy 的内置函数 array() 可以创建 ndarray 对象，其语法格式如下：

numpy.array(object, dtype = None, copy = True, order = None,subok=False,ndmin = 0)

参数说明

序号	参数	描述说明
1	object	表示一个数组序列。
2	dtype	可选参数，通过它可以更改数组的数据类型。
3	copy	可选参数，表示数组能否被复制，默认是 True。
4	ndmin	用于指定数组的维度。
5	subok	可选参数，类型为bool值，默认False。为True，使用object的内部数据类型；False：使用object数组的数据类型。

引入numpy

```
# 注意默认都会给numpy包设置别名为np
import numpy as np
```

array创建数组：

```
#array()函数，括号内可以是列表、元祖、数组、迭代对象,生成器等
np.array([1,2,3,4,5])
```

```
array([1, 2, 3, 4, 5])
```

```
# 元组
np.array((1,2,3,4,5))
```

```
array([1, 2, 3, 4, 5])
```

```
a = np.array([1,2,3,4,5])
# 数组
np.array(a)
```

```
array([1, 2, 3, 4, 5])
```

```
# 迭代对象
np.array(range(10))
```

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

```
# 生成器
np.array([i**2 for i in range(10)])

```

```
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

```
# 列表中元素类型不相同
np.array([1,1.5,3,4.5,'5'])
```

```
array(['1', '1.5', '3', '4.5', '5'], dtype='<U32')
```

```
ar1 = np.array(range(10))   # 整型
ar1
```

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

```
ar2 = np.array([1,2,3.14,4,5])   # 浮点型
ar2
```

```
array([1.  , 2.  , 3.14, 4.  , 5.  ])
```

```
ar3 = np.array([
                [1,2,3],
                ('a','b','c')
               ])   # 二维数组：嵌套序列（列表，元祖均可）
ar3
```

```
array([['1', '2', '3'],
       ['a', 'b', 'c']], dtype='<U11')
```

```
# 注意嵌套序列数量不一会怎么样
ar4 = np.array([[1,2,3],('a','b','c','d')])
ar4
```

```
array([list([1, 2, 3]), ('a', 'b', 'c', 'd')], dtype=object)
```

```
# 注意嵌套序列数量不一会怎么样
ar4 = np.array([[1,2,3],[1,2,3,4]])
ar4
```

```
array([list([1, 2, 3]), list([1, 2, 3, 4])], dtype=object)
```

练习1

1. 创建10以内的偶数的数组

```

```

```
1
```

1. 设置dtype参数，默认自动识别

```
a = np.array([1,2,3,4,5])
print(a)
# 设置数组元素类型
has_dtype_a = np.array([1,2,3,4,5],dtype='float')
has_dtype_a
# [1.,2.,3.,4.,5.]
```

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

```
array([1., 2., 3., 4., 5.])
```

H5 思考如何将浮点型的数据，设置为整形，会是什么情况？

```
np.array([1.1,2.5,3.8,4,5],dtype='int')

```

```
array([1, 2, 3, 4, 5])
```

2.设置copy参数,默认为True

```
a = np.array([1,2,3,4,5])
# 定义b，复制a
b = np.array(a)
# 输出a和b的id
print('a:', id(a), '  b:', id(b))
print('以上看出a和b的内存地址')

# a ---复制
# b ----未复制
b[0] = 10
print(a)

```

```
a: 2066732212352   b: 2066732213152
以上看出a和b的内存地址
[1 2 3 4 5]
```

```
# 当修改b的元素时，a不会发生变化
b[0] = 10
print('a:', a,'  b:', b)
print('='*10)
```

```
a: [1 2 3 4 5]   b: [10  2  3  4  5]
==========
```

```
a = np.array([1,2,3,4,5])
# 定义b，当设置copy参数为Fasle时，不会创建副本，
# 两个变量会指向相同的内容地址，没有创建新的对象
b = np.array(a, copy=False)
# 输出a和b的id
print('a:', id(a), '  b:', id(b))
print('以上看出a和b的内存地址')
# 由于a和b指向的是相同的内存地址，因此当修改b的元素时，a会发生变化
b[0] = 10
print('a:',a,'  b:',b)
```

```
a: 2066732267520   b: 2066732267520
以上看出a和b的内存地址
a: [10  2  3  4  5]   b: [10  2  3  4  5]
```

1 ndmin 用于指定数组的维度

```
a = np.array([1,2,3])
print(a)

a = np.array([1,2,3], ndmin=2)
a

```

```
[1 2 3]
```

```
array([[[1, 2, 3]]])
```

4.subok参数，类型为bool值，默认False。为True，使用object的内部数据类型；False：使用object数组的数据类型。

```python
# 创建一个矩阵
a = np.mat([1,2,3,4])
# 输出为矩阵类型
print(type(a))

#既要复制一份副本，又要保持原类型
at = np.array(a,subok=True)
af = np.array(a)  # 默认为False
print('at,subok为True:',type(at))
print('af,subok为False:',type(af))
print(id(at),id(a))

```

```
<class 'numpy.matrix'>
at,subok为True: <class 'numpy.matrix'>
af,subok为False: <class 'numpy.ndarray'>
2066738151720 2066738151608
```

书写代码是需要注意的内容：

```python
#定义个数组
a = np.array([2,4,3,1])
# 在定义b时，如果想复制a的几种方案：

# 1.使用np.array()
b = np.array(a)
print('b = np.array(a): ',id(b),id(a))

# 2.使用数组的copy()方法
c = a.copy()
print('c = a.copy(): ',id(c),id(a))

# 注意不能直接使用=号复制，直接使用=号，会使2个变量指向相同的内存地址
d = a
# 修改d也会相应的修改a
print('d = a: ',id(d),id(a))
```

```
b = np.array(a):  2066731363744 2066731901216
c = a.copy():  2066732267520 2066731901216
d = a:  2066731901216 2066731901216
```

练习

完成以下内容，完成后截图，图片上写上姓名发到钉钉群里。

1. 创建一个一维数组

2. 创建一个二维数组
3. 创建嵌套序列数量不一样的数组，查看结果
4. 测试数组a，将数组赋值给b，修改b中的一个元素，查看a是否变化。
5. 紧接着4.如果想让b变化不影响a，如何实现

arange()生成区间数组

根据 start 与 stop 指定的范围以及 step 设定的步长，生成一个 ndarray。

numpy.arange(start, stop, step, dtype)

参数说明

序号	参数	描述说明
1	start	起始值，默认为0
2	stop	终止值（不包含）
3	step	步长，默认为1
4	dtype	返回ndarray的数据类型，如果没有提供，则会使用输入数据的类型。

```
np.arange(10)
# np.array(range(10))
```

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

```
# 可以使用浮点型数值
np.arange(3.1)
```

```
array([0., 1., 2., 3.])
```

```
# 这个的结果?
range(3.1)
# a --- [0., 1., 2., 3.]
# b --- [0, 1, 2, 3]
# c --- 错误
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-25-3e6ea257d2e7> in <module>
      1 # 这个的结果?
----> 2 range(3.1)
      3 # a --- [0., 1., 2., 3.]
      4 # b --- [0, 1, 2, 3]
      5 # c --- 错误
```

```
TypeError: 'float' object cannot be interpreted as an integer
```

```
# 返回浮点型的，也可以指定类型
x = np.arange(5, dtype =  float)
x
```

```
array([0., 1., 2., 3., 4.])
```

设置了起始值、终止值及步长：

```
# 起始10 ，终止值20 步长2
np.arange(10,20,2)

```

```
array([10, 12, 14, 16, 18])
```

```
# 起始0 ，终止值10 步长3

ar2 = np.arange(20,3) # 这个书写是否正确?
print(ar2)
# A -正确
# B -   错误
```

```
[]
```

```
#正确的书写格式是什么
# 起始0 ，终止值10 步长3

ar2 = np.arange(0,20,3) # 这个书写是否正确?
print(ar2)
ar3 = np.arange(20,step=3)
ar3

```

```
[ 0  3  6  9 12 15 18]
```

```
array([ 0,  3,  6,  9, 12, 15, 18])
```

```
# 如果数组太大而无法打印，NumPy会自动跳过数组的中心部分，并只打印边角：
np.arange(10000)
```

```
array([   0,    1,    2, ..., 9997, 9998, 9999])
```

题目：

在庆祝教师节活动中，学校为了烘托节日气氛，在200米长的校园主干道一侧，

从起点开始，每间隔3米插一面彩旗，由近到远排成一排，

问：1.最后一面彩旗会插到终点处吗？

2.一共应插多少面彩旗？

```
# 1最后一面彩旗会插到终点处吗?
np.arange(0,200+1,3)
len(np.arange(0,200+1,3))
```

```
67
```

如何防止 float 不精确影响numpy.arange

> 注意：ceil((stop - start)/step)确定项目数，小浮点不精确(stop = .400000001)可以向列表中添加意外值。

想得到一个长度为3的、从0.1开始的、间隔为0.1的数组，想当然地如下coding，结果意料之外：

```
np.arange(0.1,0.4,0.1)
```

```
array([0.1, 0.2, 0.3, 0.4])
```

linspace() 创建等差数列

返回在间隔[开始，停止]上计算的num个均匀间隔的样本。数组是一个等差数列构成

np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)

参数说明

序号	参数	描述说明
1	start	必填项，序列的起始值，
2	stop	必填项，序列的终止值，如果endpoint为true，该值包含于数列中
3	num	要生成的等步长的样本数量，默认为50
4	endpoint	该值为 true 时，数列中包含stop值，反之不包含，默认是True。
5	retstep	如果为 True 时，生成的数组中会显示间距，反之不显示。
6	dtype	ndarray 的数据类型

```
# 以下实例用到三个参数，设置起始点为 1 ，终止点为 10，数列个数为 10。
a = np.linspace(1,10,10)
a
```

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

```
a = np.linspace(1,10,endpoint=False)
a
```

```
array([1.  , 1.18, 1.36, 1.54, 1.72, 1.9 , 2.08, 2.26, 2.44, 2.62, 2.8
,
       2.98, 3.16, 3.34, 3.52, 3.7 , 3.88, 4.06, 4.24, 4.42, 4.6 ,
4.78,
       4.96, 5.14, 5.32, 5.5 , 5.68, 5.86, 6.04, 6.22, 6.4 , 6.58,
6.76,
       6.94, 7.12, 7.3 , 7.48, 7.66, 7.84, 8.02, 8.2 , 8.38, 8.56,
8.74,
       8.92, 9.1 , 9.28, 9.46, 9.64, 9.82])
```

```
# 使用等差数列 实现输出0 0.5 1 1.5 2 2.5 3 3.5 4
# 选择题： A 还是 B
A = np.linspace(0, 4, 9)
print(A)
B = np.linspace(0, 4.1, 9)
print(B)
```

```
[0.  0.5 1.  1.5 2.  2.5 3.  3.5 4. ]
[0.     0.5125 1.025  1.5375 2.05   2.5625 3.075  3.5875 4.1   ]
```

```
# 一下实例用到三个参数，设置起始位置为2.0，终点为3,0 数列个数为5
ar1 = np.linspace(2.0, 3.0, num=5)
ar1
```

```
array([2.  , 2.25, 2.5 , 2.75, 3.  ])
```

```
# 设置参数endpoint 为False时，不包含终止值
ar1 = np.linspace(2.0, 3.0, num=5, endpoint=False)
ar1
```

```
array([2. , 2.2, 2.4, 2.6, 2.8])
```

```
#设置retstep显示计算后的步长
ar1 = np.linspace(2.0,3.0,num=5, retstep=True)
print(ar1)
type(ar1)
```

```
(array([2.  , 2.25, 2.5 , 2.75, 3.  ]), 0.25)
```

```
tuple
```

```
#设置retstep显示计算后的步长
ar1 = np.linspace(2.0,3.0,num=5,endpoint=False,retstep=True)
ar1
```

```
#想得到一个长度为10的、从0.1开始的、间隔为0.1的数组
np.linspace(0.1,1,10)
```

```
array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ])
```

等差数列 在线性回归经常作为样本集

如：生成x_data，值为[0, 100]之间500个等差数列数据集合作为样本特征，根据目标线性方程y=3*x+2，生成相应的标签集合y_data

```
x_data = np.linspace(0,100,500)
x_data
```

等比数列

返回在间隔[开始，停止]上计算的num个均匀间隔的样本。**数组是一个等比数列构成**

np.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)

参数说明

序号	参数	描述说明
1	start	必填项，序列的起始值，
2	stop	必填项，序列的终止值，如果endpoint为true，该值包含于数列中
3	num	要生成的等步长的样本数量，默认为50
4	endpoint	该值为 true 时，数列中包含stop值，反之不包含，默认是True。
5	base	对数 log 的底数
6	dtype	ndarray 的数据类型

```
a = np.logspace(0,9,10,base=2)
a
```

```
array([  1.,   2.,   4.,   8.,  16.,  32.,  64., 128., 256., 512.])
```

np.logspace(A,B,C,base=D)

- A: 生成数组的起始值为D的A次方
- B:生成数组的结束值为D的B次方

- C:总共生成C个数
- D:指数型数组的底数为D，当省略base=D时，默认底数为10

```
# 我们先使用前3个参数，将[1,5]均匀分成3个数，得到{1,3,5}，
# 然后利用第4个参数base=2(默认是10) 使用指数函数可以得到最终输出结果{2^1,2^3,2^5}
np.logspace(1,5,3,base=2)

```

```
array([ 2.,  8., 32.])
```

```
# 取得1到2之间10个常用对数

np.logspace(1.0,2.0,num=10)
```

```
array([ 10.        ,  12.91549665,  16.68100537,  21.5443469 ,
        27.82559402,  35.93813664,  46.41588834,  59.94842503,
        77.42636827, 100.        ])
```

```
a =  np.linspace(1.0,2.0,num=10)
print(a)
10 ** a
```

```
[1.         1.11111111 1.22222222 1.33333333 1.44444444 1.55555556
 1.66666667 1.77777778 1.88888889 2.        ]
```

```
array([ 10.        ,  12.91549665,  16.68100537,  21.5443469 ,
        27.82559402,  35.93813664,  46.41588834,  59.94842503,
        77.42636827, 100.        ])
```

练习题

一个穷人到富人那里去借钱，原以为富人不愿意，哪知富人一口答应了下来，
但提出了如下条件：

- 在30天中，富人第一天借给穷人1万元，第二天借给2万，以后每天所借的钱数都比上一天的多一万；
- 但 借钱第一天，穷人还1分钱，第二天还2分钱，以后每天所还的钱数都是上一天的两倍。
- 30天后互不相欠，

穷人听后觉得挺划算，本想定下来，但又想到富人是个吝啬出了名的，怕上当，所以很为难。

- 同学们思考下，如何用我们刚才学到的方法计算出结果，确定是否向富人借钱？

```
# 1确定富人总共借给我多少？
# 等差
# 2.我一共需要还多钱
# 等比
# sum
```

- np.array ----
- np.arange
- np.linspace
- np.logspace

全0数列

创建指定大小的数组，数组元素以 0 来填充

numpy.zeros(shape, dtype = float, order = 'C')

参数说明

序号	参数	描述说明
1	shape	数组形状
2	dtype	数据类型，可选

```
# 默认为浮点数
np.zeros(5)
```

```
array([0., 0., 0., 0., 0.])
```

```
# 设置为整形
np.zeros((5,), dtype = 'int')
```

```
array([0, 0, 0, 0, 0])
```

```
# 2行2列的全0数组
np.zeros((2,2))
```

```
array([[0., 0.],
       [0., 0.]])
```

```
#zeros_like返回具有与给定数组相同的形状和类型的零数组
ar1 = np.array([[1,2,3],[4,5,6]])
np.zeros_like(ar1)
```

```
array([[0, 0, 0],
       [0, 0, 0]])
```

全1数列

```python
# 全为1的数列
ar5 = np.ones(9)
ar6 = np.ones((2,3,4))
ar7 = np.ones_like(ar3)
print('ar5:',ar5)
print('ar6:',ar6)
print('ar7:',ar7)
```

```
ar5: [1. 1. 1. 1. 1. 1. 1. 1. 1.]
ar6: [[[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]

 [[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]]
ar7: [1 1 1 1 1 1 1]
```

NumPy 数组属性

NumPy 的数组中比较重要 ndarray 对象属性有：

属性	说明
ndarray.ndim	秩，即轴的数量或维度的数量
ndarray.shape	数组的维度，对于矩阵，n 行 m 列
ndarray.size	数组元素的总个数，相当于 .shape 中 n*m 的值
ndarray.dtype	ndarray 对象的元素类型
ndarray.itemsize	ndarray 对象中每个元素的大小，以字节为单位

1.ndarray.shape

返回一个包含数组维度的元组，对于矩阵，n 行 m 列，它也可以用于调整数组维度

```python
a = np.array([1,2,3,4,5, 6])
```

```
print('一维数组: ',a.shape)

b = np.array([[1, 2, 3], [4, 5, 6]])
print('二维数组: ', b.shape)

c = np.array([
    [
        [1, 2, 3],
        [4, 5, 6]
    ],
    [
        [11, 22, 33],
        [44, 55, 66]
        ]
    ])
print('三维数组: ', c.shape)
```

```
一维数组:  (6,)
二维数组:  (2, 3)
三维数组:  (2, 2, 3)
```

调整维度 reshape

返回调整维度后的副本，而不改变原 ndarray。

```
a = np.array([1,2,3,4,5, 6])
print('一维数组a: ',a.shape)
# 使用a数组，创建一个新的数组b，并向形状修改为2行3列
b = a.reshape((2,3))
print('b的形状: ',b.shape)
print('b: ', b)

print('a的形状: ',a.shape)
print('a: ', a)
# 不可以修改2,3
#c = a.reshape((2,4))
#c

```

```
一维数组a:  (6,)
b的形状:  (2, 3)
b:  [[1 2 3]
 [4 5 6]]
a的形状:  (6,)
a:  [1 2 3 4 5 6]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-57-9cbf3c74bfae> in <module>
     10 # 1---不可以
     11 # 2--可以
---> 12 c = a.reshape((2,4))
     13 c

ValueError: cannot reshape array of size 6 into shape (2,4)
```

调整维度 resize

`numpy.resize(a, new_shape)`

如果新数组大于原始数组，则新数组将填充a的重复副本。

> 请注意，此行为与a.resize(new_shape)不同，后者用零而不是重复的a填充。

```
# a 为2行2列
a=np.array([
        [0,1],
        [2,3]
    ])
# 一a为原数组创建2行3列的新数组
b_2_3 = np.resize(a,(2,10))
b_2_3
```

```
array([[0, 1, 2, 3, 0, 1, 2, 3, 0, 1],
       [2, 3, 0, 1, 2, 3, 0, 1, 2, 3]])
```

```

```

2. ndarray.ndim

返回数组的维度（秩）:轴的数量，或者维度的数量，是一个标量，一维数组的秩为 1，二维数组的秩为 2

```
a = np.array([1,2,3,4,5, 6])

```

```
b = a.reshape((2,3))

c = np.array([
    [
        [1, 2, 3],
        [4, 5, 6]
    ],
    [
        [11, 22, 33],
        [44, 55, 66]
        ]
    ])
print('a的ndim: ',a.ndim)
print('b的ndim: ', b.ndim)
print('c的ndim: ', c.ndim)
```

```
a的ndim:  1
b的ndim:  2
c的ndim:  3
```

3. ndarray.size

数组元素的总个数，相当于 .shape 中 n*m 的值

```
a = np.array([1,2,3,4,5,6])
print('[1,2,3,4,5,6]的size:', a.size)

a = np.array([[1,2,3],[4,5,6]])
print('[[1,2,3],[4,5,6]]的size:', a.size)
```

```
[1,2,3,4,5,6]的size: 6
[[1,2,3],[4,5,6]]的size: 6
```

3. ndarray.dtype

ndarray 对象的元素类型

```
a = np.array([1,2,3,4,5,6])
print(a.dtype)

b = np.array([1.1,2,3,4,5,6])
print(b.dtype)
```

方法astype()

numpy数据类型转换，调用astype返回数据类型修改后的数据，但是源数据的类型不会变

```
a=np.array([1.1, 1.2])
print('a数据类型: ',a.dtype)  #
print('astype修改数据类型:',a.astype('float32').dtype)
print('原数据类型未改变',a.dtype)

# 正确操作
a = a.astype('float32')
print('修改类型后再次操作，类型改变:',a.dtype)
```

ndarray.itemsize

以字节的形式返回数组中每一个元素的大小。

例如，一个元素类型为 float64 的数组 itemsize 属性值为 8(float64 占用 64 个 bits，每个字节长度为 8，所以 64/8，占用 8 个字节）

```
a = np.array([1.1,2.2,3.3])
print('dtype:',a.dtype,' itemsize:',a.itemsize)

b = np.array([1,2,3,4,5])
print('dtype:',b.dtype,' itemsize:',b.itemsize)
```

```
dtype: float64  itemsize: 8
dtype: int32  itemsize: 4
```

数据类型

名称	描述	名称	描述
bool_	布尔型数据类型（True 或者 False）	float_	float64 类型的简写
int_	默认的整数类型（类似于 C 语言中的 long，int32 或 int64）	float16/32/64	半精度浮点数:1 个符号位，5 个指数位，10个尾数位
单精度浮点数:1 个符号位，8 个指数位，23个尾数位
双精度浮点数,包括：1 个符号位，11 个指数位，52个尾数位|
intc	和 C 语言的 int 类型一样，一般是 int32 或 int 64	complex_	复数类型，与 complex128 类型相同
intp	用于索引的整数类型（类似于 C 的 ssize_t，通常为 int32 或 int64）
complex64/128	复数，表示双 32 位浮点数（实数部分和虚数部分）

复数，表示双 64 位浮点数（实数部分和虚数部分）|
| int8/16/32/64 | 代表与1字节相同的8位整数
代表与2字节相同的16位整数
代表与4字节相同的32位整数
代表与8字节相同的64位整数 |str_ | 表示字符串类型 |
| uint8/16/32/64 | 代表1字节（8位）无符号整数
代表与2字节相同的16位整数
代表与4字节相同的32位整数
代表与8字节相同的64位整数 |string_ | 表示字节串类型,也就是bytes类型 |

```
# 将数组中的类型存储为浮点型
a = np.array([1,2,3,4],dtype=np.float64)
a
```

```
# 将数组中的类型存储为布尔类型
a = np.array([0,1,2,3,4],dtype=np.bool_)
print(a)
a = np.array([0,1,2,3,4],dtype=np.float_)
print(a)
```

```
# str_和string_区别
str1 = np.array([1,2,3,4,5,6],dtype=np.str_)
string1 = np.array([1,2,3,4,5,6],dtype=np.string_)

str2 = np.array(['我们',2,3,4,5,6],dtype=np.str_)

print(str1,str1.dtype)
print(string1,string1.dtype)
print(str2,str2.dtype)
```

在内存里统一使用unicode， 记录到硬盘或者编辑文本的时候都转换成了utf8
UTF-8 将Unicode编码后的字符串保存到硬盘的一种压缩编码方式

定义结构化数据

使用数据类型标识码

字符	对应类型	字符	对应类型	字符	对应类型	字符	对应类型
b	代表布尔型	i	带符号整型	u	无符号整型	f	浮点型
c	复数浮点型	m	时间间隔（timedelta）	M	datatime（日期时间）	O	Python对象
S,a	字节串（S）与字符串（a）	U	Unicode	V	原始数据（void）		

还可以将两个字符作为参数传给数据类型的构造函数。此时，第一个字符表示数据类型，第二个字符表示该类型在内存中占用的字节数（2、4、8分别代表精度为16、32、64位的浮点数）：

```
# 首先创建结构化数据类型
dt = np.dtype([('age','i1')])
print(dt)
# 将数据类型应用于 ndarray 对象
students = np.array([(18),(19)],dtype=dt)
students
```

以下示例描述一位老师的姓名、年龄、工资的特征，该结构化数据其包含以下字段：

```
str 字段: name
int 字段: age
float 字段: salary
```

```
import numpy as np
teacher = np.dtype([('name',np.str_,2), ('age', 'i1'), ('salary',
'f4')])
#输出结构化数据teacher
print(teacher)
#将其应用于ndarray对象
b = np.array([('wl', 32, 8357.50),('lh', 28, 7856.80)], dtype =
teacher)
print(b)
```