2 科学计算 Numpy

NumPy库的名字由“Numerical Python”缩写而来。如果是利用Anaconda安装的Python，则自带NumPy库，无须单独安装。 安装Numpy库：

conda install numpy #(推荐，能够顺便安装关联的库)

pip install numpy

在程序中使用时，需要先导入库，一般将np作为Numpy的简称（本章中的示例程序均需导入Numpy库，在此统一说明）。

In [1]:

```
# -*- coding:utf-8 -*-

import numpy as np
```

2.1 多维数组

多维数组（n-dimensional array，简称ndarray），是在数据处理领域中必用的数据结构，类似于基本数据结构中的列表：有序且内容可修改。我们可以将多维数组看成Python 基本数据类型的扩展，其提供更多的属性和方法。

2.1.1 创建数组

1.类型转换方式创建

利用类型转换方式创建数组是最常见的数组创建方式，本例中使用np.array类型转换方法分别将元组和列表转换成一维数组和三维数组。

In [2]:

```
a = np.array((1,2,3))
b = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a)
print(b)
```

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

2.批量创建

除了手动给每个数组元素赋值，更多的时候是需要创建数组并按照一定规则批量填充数据。下面介绍使用Numpy提供的批量创建数组中数据的方法

In [3]:

```
a = np.arange(5) #创建初值为0，终值为5（不包含终值），步长为1的数组。
print(a)
a = np.arange(2,5,1) #创建初值为2，终值为5（不包含终值），步长为1的数组。
print(a)
a = np.linspace(2,5,4) #创建初值为2，终值为5（包含终值endpoint=True），元素为4个的等差数组。
print(a)
a = np.logspace(0,2,5) #创建基数为10的等比数组，首个元素为10^0=1，末元素为10^2=100，共5个元素。
print(a)
```

```
[0 1 2 3 4]
[2 3 4]
[2. 3. 4. 5.]
[  1.           3.16227766  10.          31.6227766  100.        ]
```

批量创建N个相同元素的数组。

In [4]:

```
a = np.empty(5) # 生成5个元素，值为随机数的数组（速度快）
print(a)
a = np.zeros(5) # 生成5个值全为0的数组
print(a)
a = np.ones(5) # 生成5个值全为1的数组
print(a)
a = np.full(5, 6) # 生成5个值全为6的数组
print(a)
```

```
[  1.           3.16227766  10.          31.6227766  100.        ]
[0. 0. 0. 0. 0.]
[1. 1. 1. 1. 1.]
[6 6 6 6 6]
```

创建与给定数组形状相同的新数组：本例中，用 zero_like 方法创建了元素全为 0 且形状与 a 相同的数组 b，而创建值全为 1（ones_like）、全为空（empty_like），以及全为某一特定值（full_like）数组的方法与此类似。

In [5]:

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

```
[1, 2, 3]
[0 0 0]
```

Numpy.random系列函数用于创建随机数组：本例使用randint函数创建最小值为1，最大值为3（不包含最大值），元素为5个的整型数组；Numpy.random还提供了rand函数来创建0～1分布的随机样本数组、randn函数来创建标准正态分布样本数组等。

In [6]:

```
np.random.randint(1,3,5)
```

Out[6]:

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

np.from*系列函数用于通过现有的数据创建数组，本例使用np.fromfunction函数创建二维数组九九乘法表，第一个参数是调用的函数名，第二个参数是数组的形状。该系列函数还包括frombuffer，fromstring，fromiter，fromfile，fromregex等函数。

In [7]:

```
def func(i,j):
    return (i+1)*(j+1)
np.fromfunction(func,(9,9))
```

Out[7]:

```
array([[ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.],
       [ 2.,  4.,  6.,  8., 10., 12., 14., 16., 18.],
       [ 3.,  6.,  9., 12., 15., 18., 21., 24., 27.],
       [ 4.,  8., 12., 16., 20., 24., 28., 32., 36.],
       [ 5., 10., 15., 20., 25., 30., 35., 40., 45.],
       [ 6., 12., 18., 24., 30., 36., 42., 48., 54.],
       [ 7., 14., 21., 28., 35., 42., 49., 56., 63.],
       [ 8., 16., 24., 32., 40., 48., 56., 64., 72.],
       [ 9., 18., 27., 36., 45., 54., 63., 72., 81.]])
```

2.1.2 访问数组

1.访问数组元素

通过指定索引值访问单个元素，支持正向索引和反向索引。

In [8]:

```
a=np.array([1,2,3,4,5])
print(a[3])
print(a[-1])
print(a[[1,3]]) #通过索引值列表返回多个元素并形成新的数组，新数组与原数组不共享内存，支持多维度索引。
print(a[a>3]) #根据值的范围获取子数组。
print(a[[True,False,True,False,True]]) #以布尔值方式获取数组元素，True为选取对应位置的元素。
```

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

ndarray还支持切片方式获取子数组，切片格式为[起始位置：终止位置：步长]，不包含终止值，使用格式中三个元素的组合取子数组，切片与原数组共享同一空间。

In [9]:

```
print(a[2:]) # 仅指定初始位置
print(a[:-2]) # 仅指定终止位置，并使用反向索引
print(a[2:3]) # 指定初始和终止值（不包含终止值）
print(a[::2]) # 指定步长：每两个元素取一个
print(a[::-1]) # 以倒序返回数组
```

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

在访问多维数组时，用元组（即圆括号）作为下标。

In [10]:

```
b=np.array([[1,2],[3,4]])
print(b[(1,1)])
```

```
4
```

2.常用的数组属性

属性shape用于描述数组的维度。

In [11]:

```
a = np.array([[1,2,3],[4,5,6]])
print(a.shape) #属性shape用于描述数组的维度。
print(a.dtype) #属性dtype用于描述数组的元素类型。
print(a.ndim) #属性ndim用于描述数组维度的个数，也称作秩。
print(a.size) #属性size用于描述数组包含的元素个数。
print(a.nbytes) #属性nbytes用于描述数组所占空间大小。
```

```
(2, 3)
int32
2
6
24
```

2.1.3 修改数组

1.添加数组元素

ndarray方法支持向数组中添加元素后生成新数组：append方法支持在数组末尾添加元素，insert方法支持在指定位置添加元素。

In [12]:

```
a = np.array([1,2,3,4,5])
print(np.append(a, 7)) # 返回结果： [1 2 3 4 5 7]
print(np.insert(a, 0, 0)) # 返回结果： [0 1 2 3 4 5]
```

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

2.删除数组元素

ndarray 方法支持使用索引值删除数组中的元素并返回新数组。本例中，删除数组中索引值为3的元素（即第四个元素）。

In [13]:

```
print(np.delete(a,3))
```

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

3.修改元素值

使用索引值修改单值。

In [14]:

```
a = np.array([1,2,3,4,5,6])
a[0] = 8 # [8 2 3 4 5 6]
```

使用切片方法修改多值。

In [15]:

```
a[2:4] = [88,77] # [ 8  2 88 77  5  6]
```

4.修改形状

使用 reshape 方法可修改数组形状，当参数设置成-1 时为自动计算对应值。reshape 方法返回的数组与原数据共享存储空间。

In [16]:

```
a = np.array([[1,2,3],[4,5,6]]) # shape: (2, 3)
a = a.reshape(3, 2)  # 注意：只是维度变化，不是转置
print(a)
a = a.reshape(1,-1) # shape (6, 1)
print(a)
a = a.reshape(-1) # shape (6, )
print(a)
```

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

5.修改类型

首先，查看Numpy库支持的所有数据类型。

In [17]:

```
print(np.typeDict.items())
```

```
dict_items([('?', <class 'numpy.bool_'>), (0, <class 'numpy.bool_'>), ('byte', <class 'numpy.int8'>), ('b', <class 'numpy.int8'>), (1,
<class 'numpy.int8'>), ('ubyte', <class 'numpy.uint8'>), ('B', <class 'numpy.uint8'>), (2, <class 'numpy.uint8'>), ('short', <class 'nu
mpy.int16'>), ('h', <class 'numpy.int16'>), (3, <class 'numpy.int16'>), ('ushort', <class 'numpy.uint16'>), ('H', <class 'numpy.uint1
6'>), (4, <class 'numpy.uint16'>), ('i', <class 'numpy.intc'>), (5, <class 'numpy.intc'>), ('uint', <class 'numpy.uint32'>), ('I', <cla
ss 'numpy.uintc'>), (6, <class 'numpy.uintc'>), ('intp', <class 'numpy.int64'>), ('p', <class 'numpy.int64'>), (9, <class 'numpy.int6
4'>), ('uintp', <class 'numpy.uint64'>), ('P', <class 'numpy.uint64'>), (10, <class 'numpy.uint64'>), ('long', <class 'numpy.int32'>),
('l', <class 'numpy.int32'>), (7, <class 'numpy.int32'>), ('L', <class 'numpy.uint32'>), (8, <class 'numpy.uint32'>), ('longlong', <cla
ss 'numpy.int64'>), ('q', <class 'numpy.int64'>), ('ulonglong', <class 'numpy.uint64'>), ('Q', <class 'numpy.uint64'>), ('half', <class
'numpy.float16'>), ('e', <class 'numpy.float16'>), (23, <class 'numpy.float16'>), ('f', <class 'numpy.float32'>), (11, <class 'numpy.fl
oat32'>), ('double', <class 'numpy.float64'>), ('d', <class 'numpy.float64'>), (12, <class 'numpy.float64'>), ('longdouble', <class 'nu
mpy.longdouble'>), ('g', <class 'numpy.longdouble'>), (13, <class 'numpy.longdouble'>), ('cfloat', <class 'numpy.complex128'>), ('F', <
class 'numpy.complex64'>), (14, <class 'numpy.complex64'>), ('cdouble', <class 'numpy.complex128'>), ('D', <class 'numpy.complex128'>),
(15, <class 'numpy.complex128'>), ('clongdouble', <class 'numpy.clongdouble'>), ('G', <class 'numpy.clongdouble'>), (16, <class 'numpy.
clongdouble'>), ('O', <class 'numpy.object_'>), (17, <class 'numpy.object_'>), ('S', <class 'numpy.bytes_'>), (18, <class 'numpy.bytes
_'>), ('unicode', <class 'numpy.str_'>), ('U', <class 'numpy.str_'>), (19, <class 'numpy.str_'>), ('void', <class 'numpy.void'>), ('V',
<class 'numpy.void'>), (20, <class 'numpy.void'>), ('M', <class 'numpy.datetime64'>), (21, <class 'numpy.datetime64'>), ('m', <class 'n
umpy.timedelta64'>), (22, <class 'numpy.timedelta64'>), ('bool8', <class 'numpy.bool_'>), ('b1', <class 'numpy.bool_'>), ('int64', <cla
ss 'numpy.int64'>), ('i8', <class 'numpy.int64'>), ('uint64', <class 'numpy.uint64'>), ('u8', <class 'numpy.uint64'>), ('float16', <cla
ss 'numpy.float16'>), ('f2', <class 'numpy.float16'>), ('float32', <class 'numpy.float32'>), ('f4', <class 'numpy.float32'>), ('float6
4', <class 'numpy.float64'>), ('f8', <class 'numpy.float64'>), ('complex64', <class 'numpy.complex64'>), ('c8', <class 'numpy.complex6
4'>), ('complex128', <class 'numpy.complex128'>), ('c16', <class 'numpy.complex128'>), ('object0', <class 'numpy.object_'>), ('bytes0',
<class 'numpy.bytes_'>), ('str0', <class 'numpy.str_'>), ('void0', <class 'numpy.void'>), ('datetime64', <class 'numpy.datetime64'>),
('M8', <class 'numpy.datetime64'>), ('timedelta64', <class 'numpy.timedelta64'>), ('m8', <class 'numpy.timedelta64'>), ('Bytes0', <clas
s 'numpy.bytes_'>), ('Datetime64', <class 'numpy.datetime64'>), ('Str0', <class 'numpy.str_'>), ('Uint64', <class 'numpy.uint64'>), ('i
nt32', <class 'numpy.int32'>), ('i4', <class 'numpy.int32'>), ('uint32', <class 'numpy.uint32'>), ('u4', <class 'numpy.uint32'>), ('int
16', <class 'numpy.int16'>), ('i2', <class 'numpy.int16'>), ('uint16', <class 'numpy.uint16'>), ('u2', <class 'numpy.uint16'>), ('int
8', <class 'numpy.int8'>), ('i1', <class 'numpy.int8'>), ('uint8', <class 'numpy.uint8'>), ('u1', <class 'numpy.uint8'>), ('complex_',
<class 'numpy.complex128'>), ('int0', <class 'numpy.int64'>), ('uint0', <class 'numpy.uint64'>), ('single', <class 'numpy.float32'>),
('csingle', <class 'numpy.complex64'>), ('singlecomplex', <class 'numpy.complex64'>), ('float_', <class 'numpy.float64'>), ('intc', <cl
ass 'numpy.intc'>), ('uintc', <class 'numpy.uintc'>), ('int_', <class 'numpy.int32'>), ('longfloat', <class 'numpy.longdouble'>), ('clo
ngfloat', <class 'numpy.clongdouble'>), ('longcomplex', <class 'numpy.clongdouble'>), ('bool_', <class 'numpy.bool_'>), ('bytes_', <cla
ss 'numpy.bytes_'>), ('string_', <class 'numpy.bytes_'>), ('str_', <class 'numpy.str_'>), ('unicode_', <class 'numpy.str_'>), ('object
_', <class 'numpy.object_'>), ('int', <class 'numpy.int32'>), ('float', <class 'numpy.float64'>), ('complex', <class 'numpy.complex12
8'>), ('bool', <class 'numpy.bool_'>), ('object', <class 'numpy.object_'>), ('str', <class 'numpy.str_'>), ('bytes', <class 'numpy.byte
s_'>), ('a', <class 'numpy.bytes_'>)])
```

```
<ipython-input-17-5b5f08af9ab7>:1: DeprecationWarning: `np.typeDict` is a deprecated alias for `np.sctypeDict`.
  print(np.typeDict.items())
```

In [18]:

```
a = np.array([1,2,3,4,5,6], dtype=np.int64) #指定类型并创建数组。
print(a.dtype)
a = a.astype(np.float32) #转换数组中数据的类型，并查看其转换后的具体类型。
print(a.dtype)
print(a.dtype.type)
```

```
int64
float32
<class 'numpy.float32'>
```

2.2 数组元素运算

ufunc（universal function）是对数组中每个元素运算的函数，比循环处理速度更快且写法简单。Numpy提供了很多数组相关的ufunc方法，同时也支持自定义ufunc函数。Numpy提供的ufunc函数分为一元函数（Unary ufuncs）和二元函数（Binary ufuncs）。

2.2.1 一元函数

一元函数是参数为单个值或者单个数组的函数，如取整、三角函数等，常用的函数如表所示。

功能	函数	描述
指数对数函数	Sqrt	计算平方根
	square	计算平方
	exp	计算以 e 为底的指数函数
	log，log10，log2，log1p	计算以 e/10/2/1+x 为底的对数
取整函数	ceil	向上取整
	floor	向下取整
	round	四舍五入
判断类型	isnan	判断是否为空值
	isinf	判断是否为无限大数
	isfinate	判断是否为有限大数
三角函数	cos，cosh，sin，sinh，tan，tanh	三角函数
	arcos，arcosh，arsin，arsinh，artan，artanh	反三角函数

一元函数的使用方法与一般函数的相同，形如：

In [19]:

```
print(np.abs([-3,-2,5]))
```

```
[3 2 5]
```

2.2.2 二元函数

二元函数是参数为两个数组的函数，包括算术运算和布尔运算。

1.算术运算

本例中利用四则运算符实现两个数组间的算术运算，并返回新数组。先定义数组：

In [20]:

```
a = np.array([1,2,3])
b = np.array([4,5,6])
print(a+b) #加
print(b-a) #减
print(a*b) #乘
print(b/a) #除
print(b//a) #整除
print(b%a) #整除取余数
```

```
[5 7 9]
[3 3 3]
[ 4 10 18]
[4.  2.5 2. ]
[4 2 2]
[0 1 0]
```

2.布尔运算

布尔运算是指使用“>”“<”“>=”“<=”“==”“！=”等逻辑运算符比较两个数组，并返回布尔型数据，其中每个元素是两个数组中对应数据比较的结果。

In [21]:

```
a = np.array([1,2,3])
b = np.array([1,3,5])
print(a<b)
print(a==b)
```

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

2.2.3 广播

前面介绍了当两个数组形状相同时，可以进行算术运算和布尔运算，具体方法是两个数组对应位置的数据运算。而当它们的形状不同时，数据就会将自动扩展对齐维数较大的数组后，再进行运算，这种从低维向高维的自动扩展被称为广播（broadcasting）。

最常见的广播形式如下，即它将被减数1扩展成为与减数形状相同的数组[1，1，1]后，再进行二元减法运算：

In [22]:

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

```
[0 1 2]
```

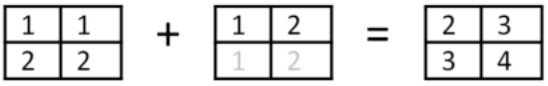
相对复杂的是对多维数组的广播，具体沿哪一个轴扩展取决于待扩展数组的形状，简单的规则是它会沿缺失的方向扩展。首先构造数据：

In [23]:

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

```
[[1 1]
 [2 2]]
```

当纵轴向缺少数据时，向纵轴方向扩展，如图所示。



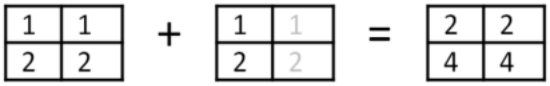
当第二个数组中仅有一行数据时，自动扩展为两行，第二行的内容与第一行相同，扩展之后再进行后续运算。

In [24]:

```
b=np.array([1,2])
print("shape", b.shape, a+b)
```

```
shape (2,) [[2 3]
 [3 4]]
```

当横轴向缺少数据时，向横轴方向扩展，如图所示。



当第二个数组中仅有一列数据时，自动扩展为两列，第二列的内容与第一列相同，扩展之后再进行后续运算。

In [25]:

```
c=np.array([[1],[2]])
print("shape", c.shape, a+c)
```

```
shape (2, 1) [[2 2]
 [4 4]]
```

2.2.4 自定义ufunc

自定义ufunc函数包括两个主要步骤：第一步，定义函数，其方法和定义普通函数的方法相同。第二步，利用np.frompyfunc将函数转换为元素运算函数，它的第二个和第三个参数分别为之前定义的普通函数的入参和返回值个数。定义好之后，就可以将数组作为参数和其他参数一起调用函数了。

In [26]:

```
def my_ufunc(x, low, high):
    if x < low:
        return -1
    elif x > high:
        return 1
    else:
        return 0
my_ufunci = np.frompyfunc(my_ufunc, 3, 1)
```

In [27]:

```
x = np.arange(1,10,1)
y2 = my_ufunci(x, 3, 6)
print(x)
print(y2)
```

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

2.3 常用函数

Numpy 为科学计算设计专门提供了庞大的函数库以简化代码量并提高程序运行效率，这样可以使程序员不用再关注具体的实现细节，而有更多的时间和精力去关注程序的目标、框架和逻辑。

2.3.1 分段函数

分段函数高效简练，使用它可以将if条件选择和select分支仅通过一条语句实现对数组的处理，方便的同时还可以提高代码的可读性。

1.where函数

where函数是if/else条件选择语句对数组操作的精简写法，其语法如下：

x = np.where(condition, y, z)

其中，参数condition，y，z都是大小相同的数组（或用于生成数组的表达式），condition为布尔型。当其元素值为True且选择相应位置y数组的值为False时，选择数组z中的值，返回值是新值组成的数组。

In [28]:

```
x=np.arange(10)
print(x)
print(np.where(x<5, x, 9-x))
```

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

2.select函数

当判断条件为多个时，需要多次调用where函数，而用select函数可用单个语句实现该功能，其语法如下：

select(condlist, choicelist, default=0)

其中，condlist是条件列表，choicelist是值列表，default是默认值。示例将小于3的数置为-1，3～6的置为0，大于6的置为1，例程如下：

In [29]:

```
a=np.arange(10)
print(np.select([x<3,x>6], [-1,1], 0))
```

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

3.piecewise函数

piecewise函数是select函数的扩展，它不但支持按不同条件取值，还支持按条件运行不同函数或lambda表达式。其语法如下：

picewise(X, condlist, funclist)

语法与 select函数略有不同：第一个参数是待转换数组 x，第二个参数 condlist是条件列表，第三个参数funclist是函数列表。当满足condlist中的条件时，执行对应位置funclist中的函数并取其返回结果。

In [30]:

```
def func1(x):
    return x*2

def func2(x):
   return x*3

a=np.arange(10)
print(np.piecewise(x, [x<3,x>6], [func1,func2]))
print(np.piecewise(x, [x<3,x>6], [lambda x: x * 2, lambda x: x * 3]))
```

```
[ 0  2  4  0  0  0  0 21 24 27]
[ 0  2  4  0  0  0  0 21 24 27]
```

2.3.2 统计函数

本小节将介绍针对数值类型的统计函数的使用方法。

1.均值、方差、分位数

In [31]:

```
a=np.arange(10,0,-1)
print(a)
print(a.mean()) #均值
print(a.var()) #方差
print(a.std()) #标准差
print(np.average(a, weights=np.arange(0,10,1))) #加权平均值，其中weights为赋予数组中每个值的权重。
print(np.median(a)) #中数，如果数组元素个数为偶数，则计算中间两数的平均值
print(np.percentile(a, 75)) #分位数，75分位数是排序后计算在75%位置的数值。
```

```
[10  9  8  7  6  5  4  3  2  1]
5.5
8.25
2.8722813232690143
3.6666666666666665
5.5
7.75
```

2.极值和排序

计算数组的最大值（max）、最小值（min），以及最大值和最小值之差（ptp）。

In [32]:

```
print(a.min()) #最小值
print(a.max()) #最大值
print(a.ptp()) #最大值和最小值之差
print(a.argmin()) #返回沿轴的最小值的索引
print(a.argmax()) #返回沿轴的最大值的索引
print(a.argsort()) #返回沿轴排序的索引
a.sort() #返回排序结果
print(a)
```

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

3.统计

首先创建测试数组，然后用unique方法统计数组中所有不同的值。

In [33]:

```
a=np.random.randint(0,5,10)
print(a)
print(np.unique(a)) # 统计数组中所有不同的值
print(np.bincount(a)) #统计整数数组中每个元素出现的次数
print(np.histogram(a,bins=5)) #统计一维数组数据分布直方图，用bins方法指定区间各数。
                              #函数返回两个数组：第一个数组是每个区间值的个数，第二个数组是各区间的边界位置。
```

```
[2 2 2 3 0 0 4 1 0 0]
[0 1 2 3 4]
[4 1 3 1 1]
(array([4, 1, 3, 1, 1], dtype=int64), array([0. , 0.8, 1.6, 2.4, 3.2, 4. ]))
```

2.3.3 组合与分割

1.组合

利用concatenate和stack系列函数可连接两个数组以创建数组。首先定义测试数组，然后使用stack函数连接两个数组，从返回结果中可以看到，连接后新数组的秩增加。

In [34]:

```
a=np.array([[1,2],[3,4]])
b=np.array([[5,6],[7,8]])
d=np.stack((a,b))
print("shape", d.shape)
print("dim", d.ndim)
print(d)

print(np.column_stack((a,b))) #沿第0轴连接数组（横向连接），而纵向连接用row_stack方法和vstack方法。
print(np.hstack((a,b))) #沿第0轴连接数组（横向连接）

print(np.concatenate([a,b],axis=0))#既可横向连接，也可纵向连接，可通过 axis 参数指定连接方向，是使用频率最高的数组连接方法。

print(np.ravel(a)) #flatten 方法和 reval 方法都可将多维数组展成一维，差别是flatten方法返回一份拷贝copy
c = a.flatten()
print(c)
```

```
shape (2, 2, 2)
dim 3
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
[[1 2 5 6]
 [3 4 7 8]]
[[1 2 5 6]
 [3 4 7 8]]
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
[1 2 3 4]
[1 2 3 4]
```

2.分割

split方法和array_split方法都可用于切分数组，split方法只支持平均分组，而array_split方法尽量平均分组。array_split方法的第一个参数是待切分数组。而当第二个参数设置为整数时，按整数指定的份数切分；当第二个参数设置为数组时，将数组中指定索引值作为切分点进行切分。

In [35]:

```
print(np.split(c,[1,2]))
print(np.split(c, 2))
print(np.array_split(c, 3))

print(np.hsplit(a, 2)) #横向切分数组的
```

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

2.3.4 矩阵与二维数组

矩阵（matrix）是数组的一个分支，即二维数组。在一般情况下，矩阵和数组的使用方法相同，但数组相对更灵活，而矩阵提供了一些二维数组计算的简单方法。在两种方法均可实现功能时，建议使用数组。

矩阵（二维数组）是除一维数组外最常用的数组形式，数据表和图片都会用到该数据结构。本小节除了介绍矩阵的基本操作，还介绍一些线性代数和二维数据表的常用方法。

1.创建矩阵

用np.mat方法可将其他类型的数据转换为矩阵。

In [36]:

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

a = np.mat(np.random.random((2,2))) #随机数矩阵
print(a)
print(np.eye(2)) #单位矩阵
print(np.diag([2,3])) #对角矩阵
```

```
<class 'numpy.matrix'>
[[0.96655663 0.35320864]
 [0.04600534 0.29474943]]
[[1. 0.]
 [0. 1.]]
[[2 0]
 [0 3]]
```

2.线性代数常用方法

线性代数包括行列式、矩阵、线性方程组、向量空间等结构，它们均可用Numpy的矩阵描述。Numpy 也提供了一些线性变换、特征分解、对角化等问题的求解方法，常用函数如下：

In [37]:

```
a = np.mat([[1.,2.],[3.,4.]])
print(np.dot(a,a))     # 矩阵乘积
print(np.multiply(a,a))     # 矩阵点乘
print(a.T)    # 矩阵转置
print(a.I)    # 矩阵求逆
print(np.trace(a))     # 求矩阵的迹
print(np.linalg.eig(a))    # 特征分解
```

```
[[ 7. 10.]
 [15. 22.]]
[[ 1.  4.]
 [ 9. 16.]]
[[1. 3.]
 [2. 4.]]
[[-2.   1. ]
 [ 1.5 -0.5]]
5.0
(array([-0.37228132,  5.37228132]), matrix([[-0.82456484, -0.41597356],
        [ 0.56576746, -0.90937671]]))
```

3.数据表常用方法

沿矩阵的某一轴向运算是常用的数据表统计方法，如对某行求和、对某列求均值等，注意要使用axis参数指定轴向。下例使用求和函数（sum）举例统计不同轴向的用法，其他统计方法以此类推。

In [38]:

```
a = np.mat(np.mat([[1,2,3],[4,5,6]]))
print(a.sum())
print(a.sum(axis=0))
print(a.sum(axis=1))
```

```
21
[[5 7 9]]
[[ 6]
 [15]]
```

In [39]:

```
print(np.cov(a)) #协方差是描述二维数据间相关程度的统计量
```

```
[[1. 1.]
 [1. 1.]]
```

2.3.5 其它常用函数

除了各种数学计算，Numpy 还提供了一些工具函数，用于数组之间以及数组与其他数据之间的转换。例如，前面提到的np.array方法能将其他类型的数据转换成数组。对应的，在将数组转换成列表时使用tolist方法：

In [40]:

```
a = np.mat(np.random.randint(1,3,5))
print(a.tolist(), type(a.tolist()))

b = a.view() #用view方法以视图的方式创建新数组，它与原数组指向同一数据，且数据保存在base指向的数组中。
print(b is a, b.base is a)

c = a.copy() #用copy方法深度复制生成数据副本，它与原数组指向不同数据。
print(c is a, c.base is a)
```

```
[[2, 1, 1, 2, 1]] <class 'list'>
False True
False False
```

In [ ]:

```

```