BIG DATA

第3章 NumPy基础

本章学习目标

- 掌握创建 ndarray 对象的各种方法。
- 掌握 ndarray 对象索引、切片、拆分、合并等基本操作。
- 理解数组运算的广播机制。
- 掌握 ndarray 的各种运算及常用函数。
- 掌握基本的统计运算。

NumPy 是 Python 科学计算和数值分析中最基本的扩展库，其前身是 Jim Hugunin 等于 1995 年开发的 Numeric 模块。2005 年，Travis Oliphant 在 Numeric 基础上综合了功能类似的 Numarray 模块的优点，并扩展了其他功能和特性，开发了 NumPy 的第 1 个版本。NumPy 使用 BSD 许可证协议，源代码开放。NumPy 功能强大，支持多维度的数组与矩阵运算，提供了大量的通用函数，支持线性代数运算、数值积分、傅里叶变换、统计计算及随机数生成等。同时，NumPy 还为 SciPy、Pandas 等常用 Python 第三方库提供底层支持。

本章主要介绍 NumPy 中多维数组对象 ndarray 的创建、操作、常用函数、数组运算、统计分析等基本功能。NumPy 是第三方库，所以并不包括在官方的 Python 发行版本中，可以使用 pip 命令下载安装，或是使用已经默认包含了 NumPy、Pandas 等常用三方库的 Anaconda 发行版本。在使用之前，需要用 import 语句显式地导入 NumPy，出于方便的目的，在导入 NumPy 时可以起一个简化的别名，习惯上使用 np，格式如下：

```
import numpy as np
```

3.1 多维数组对象 ndarray

NumPy 的核心是多维数组对象 ndarray。ndarray 是一个数据容器，要求其中的数据元素是同构的，即具有相同的数据类型。Python 内置的列表对象，虽然也可以当作数组来使用，但由于列表中元素的存储是引用语义，其内部保存的不是数据元素本身，而是指向这些元素的引用，因此其所引用的数据元素可以是任意类型，这种实现方式必然会导致开销增加、效率降低。Python 标准库中还提供了 array 数组模块，但 array 只支持一维数组，且没有提供各种运算函数，不适合复杂数值计算的要求。NumPy 中的 ndarray 多维数组对象能够弥补这些不足。

3.1.1 ndarray 对象的创建

1. 使用 array()函数创建 ndarray 对象

NumPy 中创建 ndarray 对象最常见的途径是使用 array()函数，其语法格式如下：

```
numpy.array(obj, dtype = None, ndmin = 0)
```

其中，obj 接收用于创建数组的 Python 序列对象，如 list、tuple 等；dtype 指定数组中元素的数据类型，默认为 None；ndmin 指定生成数组的最小维度。

【例 3-1】 使用 array()函数创建数组。

```
In[1]: np.array([1,2,3,4])
Out[1]: array([1, 2, 3, 4])
In[2]: np.array([[1,2,3],[4,5,6]],dtype = np.float32)
Out[2]:
    array([[1., 2., 3.],
        [4., 5., 6.]], dtype = float32)
```

例 3-1 中分别以一维列表和二维列表为参数创建 ndarray 对象。其中，第 1 个对象为一维数组对象，第 2 个对象为 2×2 的二维数组对象，数组的维度可以通过嵌套的中括号的层数直观看到。参数 dtype 为默认值时，NumPy 会根据 obj 参数提供的数据自动推导出适当的类型。

ndarray 数组元素的数据类型包括整型、浮点型以及复数等，常见类型如表 3.1 所示。

表 3.1 NumPy 中常见的数据类型

数 据 类 型	说　　明
bool	布尔值 True 或 False
int8	字节(-2^7 到 2^7-1)
int16	有符号 16 位整数(-2^{15} 至 $2^{15}-1$)
int32	有符号 32 位整数(-2^{32} 至 $2^{32}-1$)
int64	有符号 64 位整数(-2^{64} 至 $2^{64}-1$)
uint8	无符号 8 位整数(0 到 2^8-1)
uint16	无符号 16 位整数(0 到 $2^{16}-1$)
uint32	无符号 32 位整数(0 到 $2^{32}-1$)
uint64	无符号 64 位整数(0 到 $2^{64}-1$)
float32	单精度浮点型，1 位符号，8 位阶码，23 位底数
float64 / float_	双精度浮点型，1 位符号，11 位阶码，52 位底数
complex64	复数，实部和虚部由两个 32 位浮点数表示
complex128 / complex_	复数，实部和虚部由两个 64 位浮点数表示

如果创建数组时的数据类型不一致，请注意这里所说的不一致是指精度上的差异。例如，既有整数，又有浮点数。此时，NumPy 会统一向精度较高的类型转换，并使 ndarray 对象中元素类型保持一致。

2. 使用 arange()函数创建数组

arrange()函数用于创建指定数值区间内均匀间隔的数值序列构成的 ndarray 对象，其

语法格式如下：

```
np.arrange(start, stop, step, dtype)
```

其中，start 为起始值，默认值为 0，可以缺省；stop 为终止值，生成的数组中不包括终止值；step 为步长，默认值为 1，可以缺省；dtype 指定数组中元素的数据类型，默认为 None。

arrange()函数和 Python 内置的 range()对象非常类似，不同点在于 range()产生的是一个可迭代的整数序列对象，而 arange()得到的是 ndarray 数组对象，且 arange()函数中的 start、stop 和 step 参数可以为浮点数。

【例 3-2】 使用 arange()函数创建数组。

```
In[3]: np.arange(1,10)
Out[3]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
In[4]: np.arange(1.5,4.5,0.5)
Out[4]: array([1.5, 2. , 2.5, 3. , 3.5, 4. ])
```

注意：在 Jupyter Notebook 中，如果没有使用 print()函数，而是直接输出 ndarray 数组，则会显示为 array[…]的样式，但并不影响对结果的理解和数组元素值。

3. 创建含有等比及等差数列的数组对象

可以使用 linspace()函数创建等差数列，语法如下：

```
linspace(start, stop, num, endpoint = True)
```

其中，参数 start、stop 和 num 分别表示等差数列的起始值、终止值和数列中元素的个数；参数 endpoint 表示是否包括终止值 stop，默认值为 True，即数列中包含 stop。

【例 3-3】 使用 linspace 函数创建等差数列数组。

```
In[5]: np.linspace(1,12,5)
Out[5]: array([ 1., 3.75, 6.5 , 9.25, 12. ])
In[6]: np.linspace(1,12,5,endpoint = False)
Out[6]: array([1. , 3.2, 5.4, 7.6, 9.8])
```

logspace()函数可以用于创建包含等比数列的 ndarray 对象，语法格式如下：

```
logspace(start, stop, num, endpoint = True, base = 10)
```

其中，参数 num 及 endpoint 的含义与 linspace()函数相同，数列的起始值和终止值分别是 base 的 start 次幂和 base 的 stop 次幂；base 的默认值为 10。

【例 3-4】 使用 logspace()函数创建等比数列数组。

```
In[7]: np.logspace(1,2,5)
Out[7]: array([ 10. , 17.7827941 , 31.6227766 , 56.23413252, 100. ])
In[8]: np.logspace(0,3,4,base = 2)
Out[8]: array([1., 2., 4., 8.])
```

4. 创建全 0 数组和全 1 数组

zeros()函数可以创建指定大小，元素为 0 的数组对象，语法格式如下：

```
np.zeros(shape, dtype = float)
```

zeros_like()函数创建与指定数组具有相同大小和数据类型，且元素值为 0 的数组。

【例 3-5】 使用 zeros()函数和 zeros_like()函数。

```
In[9]: np.zeros((3,4))
Out[9]:
    array([[0., 0., 0., 0.],
           [0., 0., 0., 0.],
           [0., 0., 0., 0.]])
In[10]:
    a = np.array([[1,2,3],[4,5,6]])
    a
Out[10]:
    array([[1, 2, 3],
           [4, 5, 6]])
In[11]: np.zeros_like(a)
Out[11]:
    array([[0, 0, 0],
           [0, 0, 0]])
```

ones()函数和 ones_like()函数可以创建全 1 数组，用法与 zeros()函数 zeros_like()函数类似，读者可自行练习。

5. 创建对角矩阵

eye()函数可以创建一条对角线上元素值为 1，其他位置为 0 的矩阵，语法如下：

```
eye(N, M = None, K = 0)
```

其中，N 为数组对象的行数；M 为数组对象的列数，默认值为 N；K 为默认值 0 时主对角线元素为 1，K > 0 时全 1 对角线向右上方偏移，K < 0 时全 1 对角线向左下方偏移，偏移的尺寸根据 K 的具体值而定。

diag()函数可以创建指定主对角线元素值的对角矩阵。

【例 3-6】 使用 eye()函数和 diag()函数创建对角矩阵。

```
In[12]: np.eye(4)
Out[12]:
    array([[1., 0., 0., 0.],
           [0., 1., 0., 0.],
           [0., 0., 1., 0.],
           [0., 0., 0., 1.]])
In[13]: np.eye(4,k = 1)
Out[13]:
    array([[0., 1., 0., 0.],
           [0., 0., 1., 0.],
           [0., 0., 0., 1.],
           [0., 0., 0., 0.]])
In[14]: np.diag([1,2,3,4])
Out[14]:
    array([[1, 0, 0, 0],
           [0, 2, 0, 0],
```

```
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

3.1.2 ndarray 对象的属性

ndarray 数组对象的常用属性如表 3.2 所示。

表 3.2 ndarray 的属性

属　　性	说　　明
T	返回数组的转置
dtype	数组元素的类型
shape	返回数组的形状，即数组每维的大小
ndim	返回数组的维度
size	返回数组中元素的个数
itemsize	返回数组中每个元素的尺寸

【例 3-7】 ndarray 数组属性示例。

```
In[16]:
    a = np.array([[1,2,3],[4,5,6]])
    a
Out[16]:
    array([[1, 2, 3],
           [4, 5, 6]])
In[17]: a.T
Out[17]:
    array([[1, 4],
           [2, 5],
           [3, 6]])
In[18]: a.size
Out[18]: 6
In[19]: a.shape
Out[19]: (2,3)
In[20]: a.ndim
Out[20]: 2
In[21]: a.itemsize
Out[21]: 4
```

通过例 3-7，可以很直观地理解 ndarray 数组对象的各个属性。下面再简单强调一下 shape 属性。假设有一个二维数组对象的 shape 为(3,4)，可以把 3 理解为数组的行数，把 4 理解为数组的列数，如图 3.1 所示。

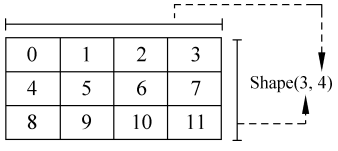


图 3.1 二维数组的形状

对于三维数组，需要注意表示其形状信息的方式，如图 3.2 所示，shape 为(2,3,4)的三维数组，可以将其视为 2 个 shape 为(3,4)的二维数组的组合。此时，三维数组的 shape 不能写成(3,4,2)。

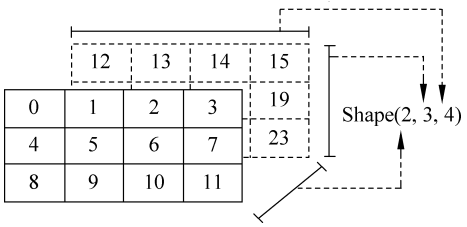


图 3.2　三维数组的形状

3.1.3　随机数数组

numpy.random 模块提供了多种用于生成随机数的函数，且功能较 Python 内置的 random 模块有所扩充，包括简单随机数、随机分布以及随机排列等。numpy.random 模块中的主要函数如表 3.3 所示。

表 3.3　numpy.random 模块中的常用函数

函　　数	说　　明
rand(d0,d1,…,dn)	生成[0,1)上均匀分布随机数数组，数组形状由参数 d0～dn 确定，若默认参数则生成单个数据
randn(d0,d1,…,dn)	生成标准正态分布随机数数组，数组形状由参数 d0～dn 确定，若默认参数则生成单个数据
randint(low,high=None,size=None)	生成指定区间[low,high)范围内的随机整数数组，形状由参数 size 确定
normal(loc=0,scale=1,size=None)	生成均值为 loc、标准差为 scale 的高斯分布随机数组，形状由参数 size 确定
binomial(n,p,size=None)	生成二项分布的随机数数组，形状由参数 size 确定
poisson(lam=1,size=None)	生成泊松分布的随机数数组，形状由参数 size 确定
uniform(low=0,high=1,size=None)	生成[low,high)区间内均匀分布的随机数数组，形状由参数 size 确定
shuffle(x)	随机排列对象 x 中元素的顺序，x 可以是数组或列表，操作直接在对象 x 上进行，改变对象 x 的值，函数无返回值
permutation(x)	将对象 x 中的元素顺序随机排列并返回一个包含随机排列结果的新数组，不直接在对象 x 上操作，原对象 x 保持不变

【例 3-8】　numpy.random 随机数函数示例。

```
In[22]: np.random.rand()                    #生成一个[0,1)区间均匀分布的随机数
Out[22]: 0.6024582870251984
In[23]: np.random.randn(2,3)                #生成标准正态分布的随机数数组,shape 为 2×3
Out[23]:
    array([[ -1.2969686 ,  -0.95285734, 3.42566191],
           [ -0.48757191, 0.73181136,  -0.7868375 ]])
In[24]: np.random.randint(1,100,(3,5))      #生成[1,100)区间随机整数数组,shape 为 3×5
Out[24]:
    array([[95, 95, 93, 50, 54],
```

```
           [34, 37, 65, 82, 48],
           [94, 70, 54, 55, 43]])
In[25]:
    arr = np.arange(10)
    np.random.permutation(arr)          #随机排列数组 arr 的元素,返回新数组,arr 不变
Out[25]:array([4, 8, 0, 2, 9, 7, 1, 3, 5, 6])
In[26]:
    np.random.shuffle(arr)              #随机排列数组 arr 的元素,直接修改 arr,无返回值
    arr
Out[26]:array([6, 0, 7, 4, 2, 3, 1, 8, 9, 5])
```

3.2 数组的基本操作

3.2.1 数组的索引和切片

1. 单个元素的索引

索引是指数组元素所在位置的编号,可以通过索引选取数组中的某些元素或为索引处的元素赋值。最基本的索引形式是用中括号加数字,与 Python 中列表对象的索引类似。

【例 3-9】 数组的索引。

```
In[27]:
    a = np.arange(10)
    a[3]
Out[27]: 3
In[28]: a[ - 4]                     #负索引,表示从后向前进行索引
Out[28]: 6
In[29]:
    b = np.array([[1,2,3],[4,5,6],[7,8,9]])
    b[1,2]                          #二维数组的索引
Out[29]: 6
In[30]:
    b[1][2] = 100                   #通过索引修改数组元素的值
    b
Out[30]:
    array([[1, 2, 3],
           [ 4, 5, 100],
           [ 7, 8, 9]])
```

数组每个维度上的索引都是从 0 开始的。此外,二维数组索引可以有两种书写方式,b[1,2]和 b[1][2],均表示数组 b 中第 1 行、第 2 列的元素。

2. 索引数组

如果想要访问数组中的多个元素,可以将这些元素的索引汇集在一起构成索引数组或索引列表,这些索引的顺序无特殊要求,而且可以重复。

【例 3-10】 索引数组。

```
In[31]:
    a = np.arange(10)
```

```
    index = [1,3,5,3]
    a[index]
Out[31]: array([ 1, 3, 5, 3])
```

例 3-10 中，通过索引数组 index 访问 a 中索引为 1、3、5、3 的元素。其中，索引 3 出现了两次。通过索引数组也可以修改原数组中元素的值，这是修改的索引数组所指定的多个元素。

如果将例 3-10 中的索引数组应用于二维 ndarray 对象，则表示按行进行索引。

【例 3-11】 用索引数组返回二维数组中的指定行。

```
In[32]:
    a = np.random.randint(1,100,(4,4))
    a
Out[32]:
    array([[92, 62, 80, 10],
           [91, 11, 46, 66],
           [ 4, 46, 97, 53],
           [77, 28, 49, 47]])
In[33]:
    index = [0,1,3,0]
    a[index]
Out[33]:
    array([[92, 62, 80, 10],
           [91, 11, 46, 66],
           [77, 28, 49, 47],
           [92, 62, 80, 10]])
```

从例 3-11 可以看出，对二维数组 a 使用索引 index，返回的是数组 a 中第 0 行、第 1 行、第 3 行、第 0 行构成的二维数组。

对于二维数组，还可以将索引数组只作为行或列某一个维度上的索引，这样可以很灵活地选择数组中指定行和指定列位置上的元素。

【例 3-12】 用索引数组返回二维数组指定行、列位置的元素，假设本例和例 3-11 连续，即继续对数组 a 进行索引操作。

```
In[34]: a[1,index]          #选取数组中第 1 行和第 0 列、第 1 列、第 3 列、第 0 列对应位置元素
                              构成的数组
Out[34]: array([91, 11, 66, 91])
In[35]: a[index,1]          #选取数组中第 0 行、第 1 行、第 3 行、第 0 行和第 1 列对应位置元素
                              构成的数组
Out[35]: array([62, 11, 28, 62])
```

如果在二维数组两个维度上的索引均以索引数组的形式给出，此时要求这两个索引数组的大小一致，这种方式是从两个索引数组的对应位置上选取两个数组构成索引。

【例 3-13】 使用两个索引数组。

```
In[36]:
    b = np.random.randint(1,20,(4,4))
    b
```

```
Out[36]:
    array([[15, 9, 17, 14],
           [17, 4, 5, 14],
           [16, 3, 14, 19],
           [11, 16, 18, 15]])
In[37]: b[[1,2],[0,3]]
Out[37]: array([17, 19])
```

例 3-13 中，b[[1,2],[0,3]]不是表示索引数组元素 b[1,2]和 b[0,3]，而是将[1,2]和[0,3]对应位置上的数构成一对索引，即 1 和 0 对应，2 和 3 对应，继而索引数组元素 b[1,0]和 b[2,3]。

3. 布尔索引

利用布尔索引，可以选择数组中满足指定条件的部分元素。

【例 3-14】 布尔索引。

```
In[38]:
    a = np.random.randint(1,10,(2,4))
    a
Out[38]:
    array([[7, 9, 1, 3],
           [8, 7, 1, 6]])
In[39]:
    index = a > 3
    a[index]
Out[39]:array([7, 9, 8, 7, 6])
```

例 3-14 中，索引是一个关系表达式 a > 3，从最终结果上看是将数组 a 中所有满足大于 3 的元素选取出来。但是，a 是一个 2×4 的二维数组，而 3 则是一个标量，数组和标量之间为什么能够直接进行比较，这涉及本书将在后面介绍的数组运算及广播机制。在此，先简单做一些说明。ndarray 数组的一个强大之处就是可以不用循环完成数组元素的批量运算，例 3-14 中的比较运算 a > 3，实际上分别判断数组 a 中的每个元素是否大于 3，结果是一个布尔型的数组，这个布尔型数组与原数组 a 的大小是一致的，根据此布尔数组中哪些元素值为 True，选取原数组中对应位置上的元素。

4. 数组切片

数组切片操作是选取数组中的一部分元素构成新数组，从形式上与 Python 中列表切片一样，将用冒号隔开的数字置于中括号之中：数组名[start:stop:step]，表示在索引范围为[start, stop)的区间内，以 step 为步长选取元素，stop 不包括在内。start 默认值为 0，stop 默认值为数组维度的大小，此时切片结果包括 stop，step 默认值为 1。对 ndarray 数组切片操作得到的是原数组的视图，并不产生副本，对切片所得数组元素的修改会反映到原数组上。

【例 3-15】 一维数组切片操作。

```
In[40]:
    a = np.arange(15)
    a
```

```
Out[40]: array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
In[41]: a[1:8:2]                    #从索引 1 到 7 区间内,以步长为 2 选取元素
Out[41]: array([1, 3, 5, 7])
In[42]: a[:5]                       #选取索引从 0 到 4 的元素
Out[42]: array([0, 1, 2, 3, 4])
In[43]: a[10:]                      #选取索引从 10 开始到数组结尾的所有元素
Out[43]: array([10, 11, 12, 13, 14])
In[44]: a[5::3]                     #索引从 5 开始到数组结尾,以 3 为步长选取元素
Out[44]: array([ 5, 8, 11, 14])
In[45]: a[::]                       #选取数组中所有元素
Out[45]: array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
```

对于二维数组,上述切片操作的语法同样适用,不过需要在切片时分别指定行和列选取的范围。此外,如果要选取的行或列索引不连续,则可以通过索引数组指定。

【例 3-16】 二维数组切片。

```
In[46]:
    a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]])
    a
Out[46]:
    array([[ 1, 2, 3, 4],
           [ 5, 6, 7, 8],
           [ 9, 10, 11, 12],
           [13, 14, 15, 16]])
In[47]: a[0:2,1:3]                  #选取第 0 行至第 1 行,第 1 列至第 2 列的元素
Out[47]:
    array([[2, 3],
           [6, 7]])
In[48]:a[1:3]                       #选取第 1 行至第 2 行
Out[48]:
    array([[ 5, 6, 7, 8],
           [ 9, 10, 11, 12]])
In[49]: a[:,1:3]                    #选取第 1 列至第 2 列
Out[49]:
    array([[ 2, 3],
           [ 6, 7],
           [10, 11],
           [14, 15]])
In[50]: a[[0,2],0:2]                #选取第 0 行和第 2 行,第 0 列至第 1 列的元素
Out[50]:
    array([[ 1, 2],
           [ 9, 10]])
```

3.2.2 数组形状变换

ndarray 数组对象提供了一些改变数组形状的方法,主要包括数组重塑及数组展平。

1. 数组重塑

数组重塑可以简单理解为改变数组的 shape,reshape()函数和 resize()函数可用于实现

这一目的。语法格式如下：

```
ndarray.reshape(new_shape)
ndarray.resize(new_shape)
```

上述两个函数在使用形式上基本相同，参数 new_shape 表示数组重塑后的 shape，需要注意这两个函数的区别：reshape()函数返回的是原数组改变形状后的视图，原数组的形状保持不变，但修改视图中的元素，原数组中的元素也会随之改变；resize()函数属于原地工作方法，不返回值，而是直接改变原数组的形状。

【例 3-17】 数组重塑。

```
In[51]:
    a = np.arange(1,13)
    a
Out[51]: array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
In[52]: a.reshape(3,4)                    #重塑数组的形状为(3,4)
Out[52]:
    array([[ 1, 2, 3, 4],
           [ 5, 6, 7, 8],
           [ 9, 10, 11, 12]])
In[53]: a                                 #数组 a 形状保持不变
Out[53]: array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
In[54]:
    a.resize(2,6)                         #重塑数组的形状为(2,6),无返回输出
    a                                     #数组 a 的形状变为(2,6)
Out[54]:
    array([[ 1, 2, 3, 4, 5, 6],
           [ 7, 8, 9, 10, 11, 12]])
```

2. 数组展平

数组展平是将数组扁平化为一维数组，ravel()函数和 flatten()函数用于数组展平，区别在于 ravel()函数返回将原数组展平后的一维数组视图，修改视图中的元素，原数组元素会随之改变；flatten()函数返回数组展平后的一维数组的副本，是新的对象。这两个函数都不会改变原数组的形状。

【例 3-18】 数组展平。

```
In[55]:
    a = np.arange(12).reshape(3,4)
    a
Out[55]:
    array([[ 0, 1, 2, 3],
           [ 4, 5, 6, 7],
           [ 8, 9, 10, 11]])
In[56]: a.ravel()                              #返回数组展平后的视图
Out[56]: array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
In[57]: a#执行 ravel()函数后,原数组 shape 保持不变
Out[57]:
    array([[ 0, 1, 2, 3],
```

```
          [ 4, 5, 6, 7],
          [ 8, 9, 10, 11]])
In[58]: a.flatten()                          #返回数组展平后的副本
Out[58]: array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
In[59]: a#执行 flatten()函数后,原数组 shape 保持不变
Out[59]:
    array([[ 0, 1, 2],
          [ 3, 4, 5],
          [ 6, 7, 8],
          [ 9, 10, 11]])
```

此外,还可以使用显式修改数组形状的方法来实现数组“降维或展平”的效果。例如:

```
In[60]:
    a = np.arange(12).reshape(3,4)
    a.shape = (1, - 1)
    a
Out[60]: array[[0,1,2,3,4,5,6,7,8,9,10,11]]
```

语句 a.shape=(1,-1)将重新定义二维数组 a 的形状,新的形状为(1,-1)。其中,第 1 个参数 1 表示新形状的行数是 1,第 2 个参数-1,表示数组的列数由系统根据元素个数和行数自动推导得出,这里很明显为 12。但需要注意:输出数组带有两层中括号,说明此时数组 a 实际上仍然是一个二维数组,并非真正降维。

3.2.3 数组转置和轴对换

在 ndarray 数组的很多运算和操作中都会涉及“轴”(axis)的概念,有必要先对轴做简要说明。数组中,轴的个数和数组的维度是相同的,即一维数组有 1 个轴,二维数组有 2 个轴,三维数组有 3 个轴,以此类推。一维数组无须进一步讨论。对于二维数组,可以将 2 个轴分别对应到数组的行和列,如图 3.3 所示。

当 axis=0 时,表示沿着数组的每列进行规定的操作或运算;当 axis=1 时,表示按照数组的每行进行规定的操作或运算。

对于三维数组的 3 个轴,0 轴和 1 轴不再对应行和列的概念,如图 3.4 所示。

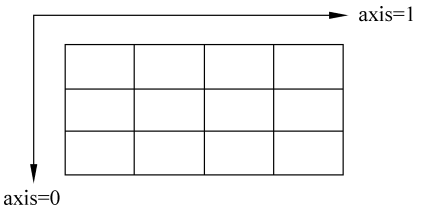


图 3.3 二维数组的轴

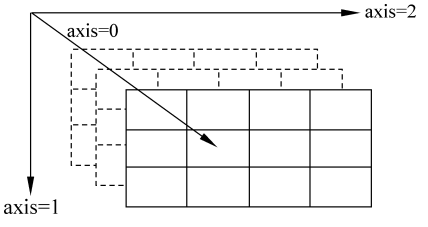


图 3.4 三维数组的轴

从上述图示说明可以看出,多维数组的轴实际上和数组各个维上的索引顺序是一致的,参见图 3.2。

ndarray 数组的转置和轴对换也可视为是一种数组重塑,可以通过 transpose()函数和 swapaxes()函数完成此类操作。

transpose()函数的语法格式如下：

```
ndarray.transpose( * axes)
```

其功能是返回按指定顺序进行轴对换后的数组，参数 axes 以元组形式给出，表示数组各个轴交换后的顺序。对于二维数组，此参数可为默认值，此时返回结果为行列转置后的数组。

【例 3-19】 用 transpose()函数进行数组轴对换。

```
In[61]:
    a = np.arange(15).reshape(3,5)
    a
Out[61]:
    array([[ 0, 1, 2, 3, 4],
           [ 5, 6, 7, 8, 9],
           [10, 11, 12, 13, 14]])
In[62]: a.transpose()                    # 参数为默认值,将 3×5 的二维数组转置为 5×3
Out[62]:
    array([[ 0, 5, 10],
           [ 1, 6, 11],
           [ 2, 7, 12],
           [ 3, 8, 13],
           [ 4, 9, 14]])
In[63]:
    b = np.arange(36).reshape(2,3,6)
    b
Out[63]:
    array([[[ 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]]])
In[64]: b.transpose((2,0,1))             # 将 2×3×6 的三维数组轴对换为 6×2×3 的三维数组
Out[64]:
    array([[[ 0, 6, 12],
            [18, 24, 30]],
           [[ 1, 7, 13],
            [19, 25, 31]],
           [[ 2, 8, 14],
            [20, 26, 32]],
           [[ 3, 9, 15],
            [21, 27, 33]],
           [[ 4, 10, 16],
            [22, 28, 34]],
           [[ 5, 11, 17],
            [23, 29, 35]]])
```

例 3-19 中，b.transpose((2,0,1))运行的结果直观上有些不好分析，可以这样理解：本来三维数组 3 个轴的顺序是(0,1,2)，用 transpose()函数交换这些轴的位置，参数(2,0,1)表示将 0 轴交换到 1 轴的位置，将 1 轴交换到 2 轴的位置，将 2 轴交换到 0 轴的位置，明确

了轴对换的规则后再分析运行结果就容易理解了。

swapaxes()函数的语法格式如下：

```
ndarray.swapaxes(axis1,axis2)
```

其功能是返回将数组的 axis1 和 axis2 两个轴对换后的数组。也就是说，不管是针对二维数组，还是更高维数组，swapaxes()函数只能交换数组的两个轴，这两个轴在参数表中的顺序可以任意。

【例 3-20】 用 swapaxes()函数进行数组轴对换。

```
In[65]:
    a = np.arange(12).reshape(3,4)
    a
Out[65]:
    array([[ 0, 1, 2, 3],
           [ 4, 5, 6, 7],
           [ 8, 9, 10, 11]])
In[66]: a.swapaxes(0,1)            # 交换二维数组的 0 轴和 1 轴，写成 a.swapaxes(1,0)即可
Out[66]:
    array([[ 0, 5, 10],
           [ 1, 6, 11],
           [ 2, 7, 12],
           [ 3, 8, 13],
           [ 4, 9, 14]])
In[67]:
    b = np.arange(36).reshape(2,3,6)
    b
Out[67]:
    array([[[ 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]]])
In[68]: b.swapaxes(1,2)            # 交换三维数组的 1 轴和 2 轴
Out[68]:
    array([[[ 0, 6, 12],
            [ 1, 7, 13],
            [ 2, 8, 14],
            [ 3, 9, 15],
            [ 4, 10, 16],
            [ 5, 11, 17]],
           [[18, 24, 30],
            [19, 25, 31],
            [20, 26, 32],
            [21, 27, 33],
            [22, 28, 34],
            [23, 29, 35]]])
```

3.2.4 数组的合并与拆分

有时需要将不同的数组通过合并操作，拼接为一个新的较大的数组；同样地，也可以将一个较大的数组通过拆分操作得到多个较小的数组。数组的合并和拆分均可以在水平方向或是垂直方向上进行，或者说是在不同的轴上进行。

1. 数组合并

NumPy 中的数组合并函数主要包括 concatenate()、vstack()和 hstack()。

concatenate()函数的语法格式如下：

```
np.concatenate((array1,array2,…),axis=0)
```

其功能是根据参数 axis 指定的轴，对数组(array1,array2…)进行合并，axis 的默认值为 0。

【例 3-21】 利用 concatenate()函数合并数组。

```
In[69]:
    a=np.array([[1,2,3],[4,5,6]])
    b=np.array([[7,8,9],[10,11,12]])
    np.concatenate((a,b),axis=0)
Out[69]:
    array([[ 1, 2, 3],
           [ 4, 5, 6],
           [ 7, 8, 9],
           [10, 11, 12]])
In[70]: np.concatenate((a,b),axis=1)
Out[70]:
    array([[ 1, 2, 3, 7, 8, 9],
           [ 4, 5, 6, 10, 11, 12]])
```

从例 3-21 的结果中可以看出，axis=0 时，进行纵向的数组合并；axis=1 时，进行横向的数组合并。

vstack()函数和 hstack()函数的功能分别是纵向合并数组和横向合并数组，实际上相当于 axis 参数分别取值为 1 和 0 时的 concatenate()函数，如图 3.5 所示。

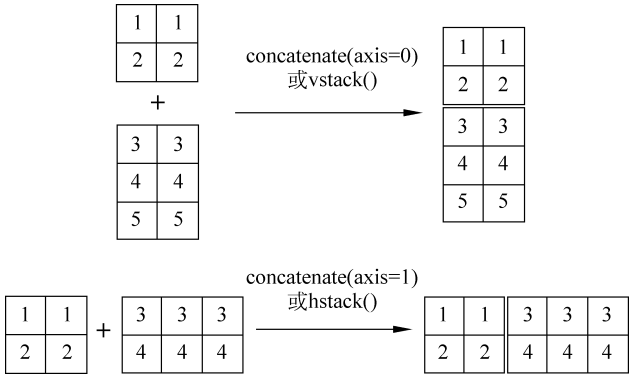


图 3.5 数组合并函数示意图

vstack()函数和 hstack()函数的语法格式如下：

```
np.vstack((array1,array2,…))
np.hstack((array1,array2,…))
```

【例 3-22】 hstack()函数和 vstack()函数。

```
In[71]:
    a = np.array([[1,2,3],[4,5,6]])
    b = np.array([[7,8,9],[10,11,12]])
    np.vstack((a,b))
Out[71]:
    array([[ 1, 2, 3],
           [ 4, 5, 6],
           [ 7, 8, 9],
           [10, 11, 12]])
In[72]: np.hstack((a,b))
Out[72]:
    array([[ 1, 2, 3, 7, 8, 9],
           [ 4, 5, 6, 10, 11, 12]])
```

2. 数组拆分

NumPy 中实现数组拆分的函数包括 split()、hsplit()和 vsplit()。

split()函数的语法格式如下：

```
np.split(array, indices_or_sections, axis = 0)
```

其功能是按照参数 indices_or_sections 指定的拆分方式和参数 axis 所指定的轴，对数组 array 进行拆分。参数 indices_or_sections 如果是一个整数，则表示将数组 array 平均拆分成几个小的数组；indices_or_sections 如果是一个整数序列，则表示对数组 array 进行拆分的位置；参数 axis 的默认值为 0，表示默认按纵向拆分，axis＝1 时，表示按横向拆分。

【例 3-23】 用 split()函数拆分数组。

```
In[73]:
    a = np.arange(24).reshape(4,6)
    a
Out[73]:
    array([[ 0, 1, 2, 3, 4, 5],
           [ 6, 7, 8, 9, 10, 11],
           [12, 13, 14, 15, 16, 17],
           [18, 19, 20, 21, 22, 23]])
In[74]: np.split(a,2)                    #将数组 a 纵向均分为 2 个数组
Out[74]:
    [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]])]
In[75]: np.split(a,[1,3],axis = 1)       #按照参数[1,3]指定的位置,对数组 a 横向拆分
Out[75]:
    [array([[ 0],
```

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

语句 np.split(a,[1,3],axis=1)中的第 2 个参数[1,3]是一个列表,列表中的两个数字表示两个拆分位置,即第 1 列和第 3 列,两个拆分位置会将数组 a 拆分成 3 部分,如图 3.6 所示。

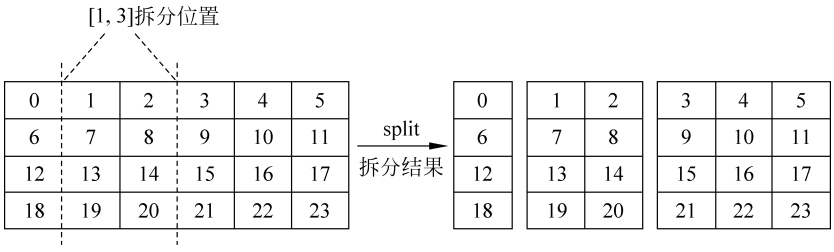


图 3.6 split()函数拆分

hsplit()函数的语法格式为 np.hsplit(array,indices_or_sections),其功能等价于 axis=1 的 split()函数。vsplit()函数的语法格式为 np.vsplit(array,indices_or_sections),其功能等价于 axis=0 的 split()函数。其中,参数的含义与 split()函数相同,读者可自行验证,此处不再赘述。

3.3 数组的运算

3.3.1 数组运算和广播机制

1. 算术运算

首先考虑一个问题,假设有两个长度相同的列表,且两个列表中都是 int 型的数据,如果希望将两个列表中对应位置上的元素相加,则要用循环来完成这个要求。

【例 3-24】 两个列表对应元素相加。

```
In[76]:
    list1 = list(range(10))
    list2 = list(range(10,20))
    list3 = []
    for i in range(10):
        list3.append(list1[i] + list2[i])
    print(list3)
Out[76]: [10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

例 3-24 的做法虽然可以实现要求，但略显烦琐。还可以使用列表推导式来完成这个问题，代码如下：

```
list3 = [x + y for x,y in zip(list1,list2)]
```

虽然代码简洁了很多，但可读性稍差，且列表推导式从本质上还是相当于一个循环。在这种情况下，利用 NumPy 数组的算术运算，可以简洁高效地解决这类问题。NumPy 中数组的运算是向量运算，可以让数组中的每个元素进行指定运算，数组可以和一个标量进行指定运算，也可以让两个数组的对应元素进行指定运算，运算结果仍然为数组。

【例 3-25】 两个数组对应元素相加。

```
In[77]:
    arr1 = np.array(list1)
    arr2 = np.array(list2)
    arr3 = arr1 + arr2
    print(arr3)
Out[77]: [10 12 14 16 18 20 22 24 26 28]
```

例 3-25 中，可以看到在两个数组 arr1 和 arr2 之间可以直接使用算术运算符＋，计算规则是两个数组中对应位置的元素依次相加。数组常用的运算包括加(＋)、减(－)、乘(＊)、除(/)、取余(％)和幂运算(＊＊)等。

【例 3-26】 数组的算术运算。

```
In[78]:
    arr1 = np.array([2,5,0,4,10,7])
    arr2 = np.array([3,1,8,4,9,11])
    arr1 % arr2
Out[78]: array([2, 0, 0, 0, 1, 7], dtype = int32)
In[79]: arr ** 2                    # 数组和标量进行运算
Out[79]: array([ 4, 25, 0, 16, 100, 49], dtype = int32)
```

例 3-26 中，参与运算的两个数组都是一维的，对于二维或多维数组，这些运算的规则也相同。除上述运算之外，NumPy 还提供了一组用于数组运算的数学函数，如表 3.4 所示。

表 3.4 NumPy 常用数学函数

函　　数	功能说明(表中 a,b 均为 ndarray 数组)
abs(a)	计算数组各元素的绝对值
sqrt(a)	计算数组各元素的平方根
square(a)	计算数组各元素的平方
log(a)、$\log_2$(a)、$\log_{10}$(a)	计算数组各元素的自然对数、以 2 为底的对数、以 10 为底的对数
sign(a)	求数组各元素的符号。其中，1 表示正数；0 表示零；－1 表示负数
ceil(a)	数组各元素向上取整
floor(a)	数组各元素向下取整
cos(a)、sin(a)、tan(a)	对数组各元素进行三角函数求值
arcos(a)、arcsin(a)、arctan(a)	对数组各元素进行反三角函数求值
add(a,b)	将两个数组对应位置的元素相加

续表

函　　数	功能说明(表中 a,b 均为 ndarray 数组)
substract(a,b)	将两个数组对应位置的元素相减
multiply(a,b)	将两个数组对应位置的元素相乘
divide(a,b)	将两个数组对应位置的元素相除
power(a,b)	对数组 a 的元素 x,数组 b 中对应位置的元素 y,计算 x 的 y 次方
dot(a,b)	计算两个数组的点乘

表 3.4 中函数的意义都很明晰,从函数名及功能都和 Python 标准库 math 模块中的函数非常相似。实际上,这些函数也都可以应用于普通的数值型标量。例如,np.add(3,4)表示求 3+4,其他不再赘述。

下面对 dot()函数加以简要说明:这个函数如果应用于两个数值标量,其功能就是求两个数的乘积;如果两个参数一个是数组,另一个是标量,则将数组元素依次与标量相乘,结果仍为一个数组;如果应用于两个一维数组,计算规则是两个数组对应位置的元素相乘的结果再相加,结果为一个标量;将 dot()函数应用于二维数组时,其功能类似于线性代数中的矩阵乘法。

【例 3-27】 dot()函数示例。

```
In[80]:
    a = np.array([1,2,3,4])
    b = np.array([2,3,4,5])
    np.dot(a,b)                          # 两个一维数组点乘
Out[80]: 40
In[81]:
    c = np.arange(12).reshape(3,4)
    d = np.arange(12).reshape(4,3)
    np.dot(c,d)                          # 两个二维数组点乘,规则与矩阵乘法相同
Out[81]:
    array([[ 42, 48, 54],
           [114, 136, 158],
           [186, 224, 262]])
In[82]: np.dot(3,5)                      # 两个标量相乘
Out[82]: 15
In[83]: np.dot(3,np.array([1,2,3]))      # 一个标量和数组点乘
Out[83]: array([3, 6, 9])
```

从例 3-27 可以看出,两个数组进行点乘运算时,并不需要相同的形状,只要第 1 个数组的列数和第 2 个数组的行数相等即可,这种运算规则与线性代数中矩阵乘法的规则是一样的。

2. 广播机制

除 dot()函数之外,本节其他示例中,参与运算的两个数组通常都具有相同的形状,那么形状不同的数组是否可以进行运算?实际上,在满足一些条件的前提下,NumPy 具有智能自动填充的功能,当两个数组的形状不相同时,可以对较小数组中的元素进行扩充,使之能够匹配较大数组的形状,这种机制称为广播(broadcasting)。

广播机制的规则可以归纳如下。

(1) 如果两个数组的维度不同,NumPy 的广播机制会为维度较小的数组添加新的轴,使其维度与较大的数组一致。

(2) 尺寸较小的数组沿着新添加的轴复制之前的元素,直到尺寸与较大的数组相同。

(3) 如果两个数组在任何维度上都不匹配,则需要将某维度中尺寸为 1 的数组拉伸,以匹配较大数组的尺寸。

上述规则看似复杂,实际上,通过分析可以得出简化的结论:数组运算过程中,要使广播机制能够起作用,参与运算的两个数组在某个维度上的尺寸要么相等,要么为 1,否则会出现广播错误,运算无法完成。

【例 3-28】 数组运算的广播机制示例 1。

```
In[84]:
    a = np.arange(3)
    a + 1
Out[84]: array([1, 2, 3])
In[85]:
    b = np.array([[1,1,1],[2,2,2],[3,3,3]])
    b + a
Out[85]:
    array([[1, 2, 3],
           [2, 3, 4],
           [3, 4, 5]])
In[86]:
    c = np.arange(1,4).reshape(3,1)
    c + a
Out[86]:
    array([[1, 2, 3],
           [2, 3, 4],
           [3, 4, 5]])
```

例 3-28 中的数据及运算都非常简单,可以通过图形化的方式展示本例中的运算过程,如图 3.7 所示。其中虚线部分就是 NumPy 广播机制自动填充的部分。

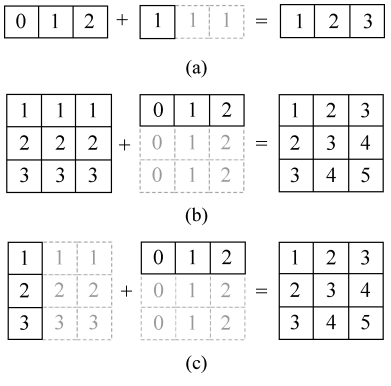


图 3.7 NumPy 数组运算的广播机制

如果两个运算对象的维度差超过 1,如一个三维数组和一个标量或一维数组进行运算,只要满足前面说到的条件,上述广播机制同样适用。

【例 3-29】 数组运算的广播机制示例 2。

```
In[87]:
    a = np.arange(12).reshape(2,3,2)
    b = np.arange(2)
    a * 2
Out[87]:
    array([[[ 0, 2], [ 4, 6], [ 8, 10]],
         [[12, 14], [16, 18], [20, 22]]])
In[88]: a + b
Out[88]:
    array([[[ 0, 2], [ 2, 4], [ 4, 6]],
         [[ 6, 8], [ 8, 10], [10, 12]]])
```

3. 关系运算

ndarray 数组支持的关系运算包括>、<、>=、<=、==、!=，这些运算含义无须解释，运算结果是包含布尔型数据的 ndarray 数组。

【例 3-30】 数组的关系运算。

```
In[89]:
    a = np.random.randint(100,size = 5)
    a
Out[89]: array([30, 24, 80, 4, 49])
In[90]:
    b = np.random.randint(100,size = 5)
    b
Out[90]: array([83, 48, 84, 75, 40])
In[91]: a > b
Out[91]: array([False, False, False, False, True])
In[92]: a % 2 == 0
Out[92]: array([ True, True, True, True, False])
```

NumPy 中还提供了一组用于关系运算的函数，其功能与关系运算符相同，如表 3.5 所示。

表 3.5 数组关系函数

关系运算函数	功能说明(a,b 均为 ndarray 数组)
np.greater(a b)	相当于 a > b
np.greater_equal(a b)	相当于 a >= b
np.less(a b)	相当于 a < b
np.less_equal(a b)	相当于 a <= b
np.equal(a b)	相当于 a == b
np.not_equal(a b)	相当于 a != b

4. 逻辑运算和条件运算

NumPy 中提供了一组进行逻辑运算的函数，如表 3.6 所示。

表 3.6　数组逻辑运算

逻辑运算函数	功能说明(a,b 均为 ndarray 数组)
np.logical_and(a,b)	将数组 a 和 b 对应位置的元素进行逻辑与运算
np.logical_or(a,b)	将数组 a 和 b 对应位置的元素进行逻辑或运算
np.logical_not(a)	对数组 a 中的元素进行逻辑取反运算
np.logical_xor()	将数组 a 和 b 对应位置的元素进行逻辑异或运算
np.any(a, axis=0)	按照 axis 指定的轴判断数组 a 中的元素是否存在 True 或非零值,如果是,则返回结果 True,否则返回 False;如果 axis 参数为默认值,则判断数组中的所有元素
np.all(a, axis=0)	按照 axis 指定的轴判断数组 a 中的元素是否均为 True 或非零值,如果是,则返回结果 True,否则返回 False;如果 axis 参数为默认值,则判断数组中的所有元素

【例 3-31】 数组逻辑运算函数。

```
In[93]:
    arr1 = np.array([2,5,0,4,10,7])
    np.any(arr1)                          # 一维数组不需要 axis 参数
Out[93]: True
In[94]: np.all(arr1)
Out[94]: False
In[95]:
    arr2 = np.array([[5, 0, 8, 7, 0], [6, 8, 1, 3, 7], [3, 5, 7, 0, 2]])
    np.any(arr2)
Out[95]: True                             # axis 参数为默认值,判断二维数组的所有元素
In[96]: np.any(arr2,axis = 0)             # 按列方向判断数组元素
Out[96]: array([ True, True, True, True, True])
In[97]: np.all(arr2,axis = 1)             # 按行方向判断数组元素
Out[97]: array([False, True, False])
In[98]:
    x = np.array([True,False,True,False])
    y = np.array([True,True,False,False])
    np.logical_xor(x,y)
Out[98]: array([False, True, True, False])
In[99]: np.logical_and(x,y)
Out[99]: array([ True, False, False, False])
```

NumPy 中的 where()函数可用于条件运算,其语法格式为 np.where(condition, x, y),参数 x 和 y 可以是数组,也可以是标量。当 x 和 y 是数组时,判断条件 condition 是否成立,如果成立则返回数组 x 中的对应位置的元素,否则返回数组 y 中对应位置的元素;如果 x 和 y 是标量,则根据 condition 条件是否成立,直接返回 x 或 y 的值。最后,将这些 x 和 y 构成一个数组作为函数返回结果。

如果参数 x 和 y 为默认值,则函数返回一个元组,元组中的每个元素是一个 ndarray 数组,其中的值是原数组中满足 condition 条件的元素在每个维度上的索引,也就是说原数组有几维,函数返回的元组中就有几个数组。

【例 3-32】 where()条件运算。

```
In[100]:
    a = np.arange(10)
    np.where(a % 2 == 0,a + 2,a - 2)
Out[100]:array([ 2,  - 1, 4, 1, 6, 3, 8, 5, 10, 7])
In[101]:
    x = np.array([15,2,74,35,19])
    y = np.array([24,17,55,89,5])
    np.where(x > y, x, y)                    # 返回数组 x 和 y 对应位置上的较大值
Out[101]:array([24, 17, 74, 89, 19])
In[102]:
    b = np.arange(12).reshape(3,4)
    np.where(b % 2 == 0)
Out[102]:
    (array([0, 0, 1, 1, 2, 2], dtype = int64),
    array([0, 2, 0, 2, 0, 2], dtype = int64))
```

例 3-32 中,二维数组 b 形式如下:

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

其中,带有阴影的元素满足 b%2==0,它们的索引分别为(0,0)、(0,2)、(1,0)、(1,2)、(2,0)和(2,2),正是 where()函数返回元组中的两个数组对应位置上的值所组成的。

where()函数还可以嵌套使用,表达更为复杂的判断逻辑。例如,实现类似 np.sign()函数的功能,可以使用 where()函数嵌套完成。

【例 3-33】 where()函数嵌套实现 sign()函数的功能。

```
In[103]:
    arr = np.random.randn(4,4)
    arr
Out[103]:
    array([[ - 0.46121921, 2.18749916, 0.07749087,  - 0.04712332],
         [ - 0.85925889, 0.33265375, 0.68970802,  - 2.58617979],
         [ - 0.85865669, 0.33193506,  - 0.33479476,  - 0.42869469],
         [ 0.11016316, 0.23310991, 1.73513802,  - 1.16844371]])
In[104]: np.where(arr > 0,1,np.where(arr < 0, - 1,0))
Out[104]:
    array([[ - 1, 1, 1,  - 1],
         [ - 1, 1, 1,  - 1],
         [ - 1, 1,  - 1,  - 1],
         [1, 1, 1,  - 1]])
```

3.3.2 数组的排序

NumPy 中用于排序的常用函数是 sort()和 argsort(),语法格式和功能如表 3.7 所示。

表 3.7　NumPy 常用排序函数

排 序 函 数	功 能 说 明
np. sort(array, axis = −1, kind = 'quicksort')	根据 axis 参数指定的轴对数组 array 的元素排序，axis 默认值为−1，表示沿着最后一个轴排序。如果指定 axis 为 None，则表示将数组展平为一维之后进行排序；参数 kind 指定排序算法，默认值为'quicksort'(快速排序)，还可以取值'mergesort'(归并排序)和'heapsort'(堆排序)。函数返回排序后的结果数组，原数组 array 不变
np. argsort(array, axis = −1, kind = 'quicksort')	返回沿着 axis 指定轴对数组 array 排序后的元素在原数组中的索引，原数组保持不变，参数意义与 np. sort()函数相同

【例 3-34】　sort()函数对数组排序。

```
In[105]:
    a = np.random.randint(1,100,(3,5))
    a
Out[105]:
    array([[57, 80, 81, 13, 42],
           [90, 72, 72, 16, 71],
           [24, 71, 93, 28, 87]])
In[106]: np.sort(a, axis = 0)                #沿 0 轴对数组排序,即每列分别排序
Out[106]:
    array([[24, 71, 72, 13, 42],
           [57, 72, 81, 16, 71],
           [90, 80, 93, 28, 87]])
In[107]: np.sort(a, axis = None)             #将数组展平排序
Out[107]: array([13, 16, 24, 28, 42, 57, 71, 71, 72, 72, 80, 81, 87, 90, 93])
In[108]:
    b = np.random.randint(1,100,(2,3,4))
    b
Out[108]:
    array([[[81, 13, 57, 2],
            [86, 53, 35, 47],
            [84, 11, 85, 24]],

           [[99, 95, 62, 40],
            [86, 18, 90, 17],
            [31, 26, 53, 62]]])
In[1090]: np.sort(b, axis = -1)              #沿着最后一个轴,即 2 轴,对三维数组排序
Out[109]:
    array([[[ 2, 13, 57, 81],
            [35, 47, 53, 86],
            [11, 24, 84, 85]],

           [[40, 62, 95, 99],
            [17, 18, 86, 90],
            [26, 31, 53, 62]]])
```

【例 3-35】 argsort()函数示例。

```
In[110]:
    c = np.random.randint(50,size = 10)
    c
Out[110]: array([26, 32, 21, 3, 14, 30, 43, 20, 16, 24])
In[111]: np.argsort(c)
Out[111]: array([3, 4, 8, 7, 2, 9, 0, 5, 1, 6], dtype = int64)
```

此外，ndarray 对象也有 sort()和 argsort()方法，语法格式如下：

```
ndarray.sort(axis = -1, kind = 'quicksort')
ndarray.argsort(axis = -1, kind = 'quicksort')
```

这两个方法的参数含义与 sort()函数、argsort()函数相同。需要注意的是，sort()方法是原地工作模式，排序会改变原数组。

3.3.3 统计运算

ndarray 数组对象支持常见的统计运算，如表 3.8 所示。

表 3.8 ndarray 对象的统计运算方法

统计运算方法	功 能 说 明
ndarray.max(axis=None)	根据 axis 指定的轴，返回数组中的最大值
ndarray.argmax(axis=None)	根据 axis 指定的轴，返回数组中的最大值元素的索引
ndarray.min(axis=None)	根据 axis 指定的轴，返回数组中的最小值
ndarray.argmin(axis=None)	根据 axis 指定的轴，返回数组中的最小值元素的索引
ndarray.ptp(axis=None)	根据 axis 指定的轴，计算数组中最大值与最小值之差
ndarray.sum(axis=None)	根据 axis 指定的轴，计算数组元素的和
ndarray.cumsum(axis=None)	根据 axis 指定的轴，计算数组元素的累计和
ndarray.mean(axis=None)	根据 axis 指定的轴，计算数组元素的平均值
ndarray.var(axis=None)	根据 axis 指定的轴，计算数组元素的方差
ndarray.std(axis=None)	根据 axis 指定的轴，计算数组元素的标准差
ndarray.prod(axis=None)	根据 axis 指定的轴，计算数组元素的乘积
ndarray.cumprod(axis=None)	根据 axis 指定的轴，计算数组元素的累积
ndarray.trace(offset=0)	返回数组对角线元素之和，参数 offset 表示离开主对角线的偏移量

表 3.8 中大多函数都有 axis 参数，用于指定统计计算应用的轴，axis 取默认值 None 表示对数组中所有元素进行指定运算。例 3-36 选取表 3.8 中部分函数加以说明。

【例 3-36】 ndarray 对象的统计运算方法。

```
In[112]:
    arr = np.random.randint(20,size = (4,4))
    arr
```

```
Out[112]:
    array([[ 5, 18, 14, 3],
           [12, 15, 19, 8],
           [ 4, 6, 16, 7],
           [14, 11, 2, 2]])
In[113]: arr.max()                    #axis 参数为默认值,返回数组所有元素中的最大值
Out[113]: 19
In[114]: arr.argmin(axis = 0)         #按列返回最小值元素索引
Out[114]: array([2, 2, 3, 3], dtype = int64)
In[115]: arr.cumsum(axis = 1)         #按行计算累计和
Out[115]:
    array([[ 5, 23, 37, 40],
           [12, 27, 46, 54],
           [ 4, 10, 26, 33],
           [14, 25, 27, 29]], dtype = int32)
In[116]: arr.ptp(axis = 1)            #按行计算最大元素和最小元素之差
Out[116]: array([15, 11, 12, 12])
In[117]: arr.mean(axis = 0)           #按列计算元素平均值
Out[117]: array([ 8.75, 12.5 , 12.75, 5. ])
In[118]: arr.std(axis = 1)            #按行计算元素的标准差
Out[118]: array([6.20483682, 4.03112887, 4.60298816, 5.35607132])
In[119]: arr.trace()                  #计算数组主对角线元素之和
Out[119]: 38
```

NumPy 中同样也提供了实现表 3.8 中所列统计运算的函数,函数名称及功能与表 3.8 中的方法也相同,调用时将数组作为参数即可。例如,np.argmax(arr,axis=0)。此外,在统计运算中,中位数也是一个常用的统计量,NumPy 中求数组元素中位数的函数是 np.medain(),不过 ndarray 数组对象并没有这个方法,需要读者注意。

3.3.4 线性代数运算

NumPy 中的 linalg 模块提供了常用的线性代数运算函数,包括矩阵求逆、求特征值、求解线性方程组等。linalg 模块常用函数如表 3.9 所示。

表 3.9 linalg 模块常用函数

线性代数函数	功 能 说 明
det()	计算矩阵行列式
eig(A)	计算方阵 A 的特征值和特征向量
inv(A)	计算方阵 A 的逆矩阵
solve(A,b)	求解线性方程组 AX=b,其中 A 是一个 N 阶方阵,b 是长度为 N 的一维向量,X 为线性方程组的解

【例 3-37】 线性代数函数示例。

```
In[120]:
    a = np.array([[1,2,3],[1,0, - 1],[0,1,1]])
    np.linalg.det(a)                  #计算矩阵的行列式
```

```
Out[120]: 2.0
In[121]: np.linalg.inv(a)          #求矩阵的逆矩阵
Out[121]:
    array([[ 0.5, 0.5, -1. ],
           [-0.5, 0.5, 2. ],
           [ 0.5, -0.5, -1. ]])
```

【例 3-38】 求解线性方程组$\begin{cases} x-2y+z=0 \\ 2y-8z=8 \\ -4x+5y+9z=-9 \end{cases}$。

```
In[122]:
    A = np.array([[1, -2,1],[0,2, -8],[ -4,5,9]])
    b = np.array([0,8, -9])
    x = np.linalg.solve(A,b)
    x
Out[122]: array([29., 16., 3.])
```

对于以上的求解结果，可以使用 dot()函数进行验证，代码如下：

```
In[123]: np.dot(A,x)
Out[123]: array([ 0., 8., -9.])
```

可以看到，A 和 x 点乘的结果就是一维数组 b。

3.4 一个有趣的数组应用实例

在本章前面几节中，初步学习了 NumPy 中 ndarray 数组使用，涉及了大量的运算和函数，读者可能会感到有些枯燥。实际上，除了枯燥的数组操作、矩阵运算之外，ndarray 对象还可以做很多有趣事情，本节介绍使用数组进行简单图像变换的实例。

RGB 模式是工业界通用的颜色标准，通过 R(红)、G(绿)、B(蓝)3 种颜色通道的变化以及它们相互之间的叠加可以得到几乎所有人类视力所能感知的颜色。一幅图像由若干像素点组成，每个像素点均由 R、G、B 这 3 种颜色组成，每种颜色的取值范围是 0～255。

在本节的实例中，会涉及用于图像处理的第三方库 PIL 以及用于数据可视化的第三方库 Matplotlib，这两个库在 Anaconda 中都已经默认安装。读者对这些内容不太了解也没有关系，这里更多关注如何将本章所学有关数组的操作和运算用于图像处理的过程。

首先，导入相关的库和模块；然后就可以使用 Image 对象的 open()函数打开图像文件。如果图像文件在 Jupyter Notebook 默认工作目录下，则直接给出文件名即可；否则需要给出绝对路径和文件名。下面假设当前工作目录下已存在一个文件名为 iris.jpg 的图像文件。

```
In[124]:
    import numpy as np
    from PIL import Image
    import matplotlib.pyplot as plt
    pic = Image.open('iris.jpg')          #读取图像文件
```

此时，如果直接输出变量 pic，则会在 Notebook 中显示原图，此处不再演示。接下来要做的是把图片的数据读入一个 ndarray 数组中，可以使用 array()函数直接完成，代码如下：

```
In[125]:
    im = np.array(pic)                    #将图片数据读入数组 im
    im.shape
Out[125]: (694, 869, 3)
```

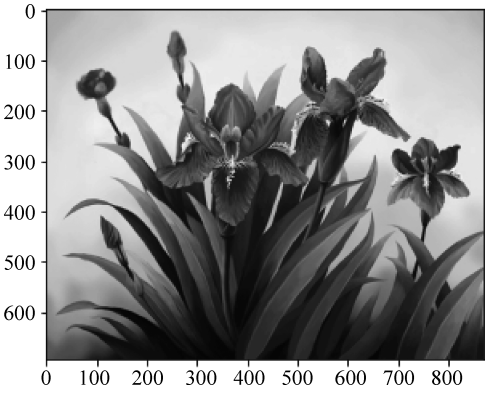
从上述结果中可以看出，im 是一个尺寸为 694×869×3 的三维数组，说明原图像文件的像素是 694×869，后面的 3 则表示每个像素点的 R、G、B 三种颜色值。可以输出数组 im 的值来观察，代码如下：

```
In[126]: im
Out[126]:                             #输出数组 im 的值，行数太多，只显示一小部分
    array([[[168, 208, 148],
        [170, 210, 150],
        [170, 210, 148],
        ...,
        [186, 208, 126],
        [189, 211, 129],
        [190, 212, 130]],
        ...,
        ...,
        [ 21, 40, 10],
        [ 19, 38, 8],
        [ 18, 37, 7]]], dtype = uint8)
```

其中，最内层括号中的 3 个数就是一个像素点的 R、G、B 三种颜色值。

下面使用 plt 模块显示图像：

```
In[127]:
    plt.imshow(pic)
    plt.show()
```

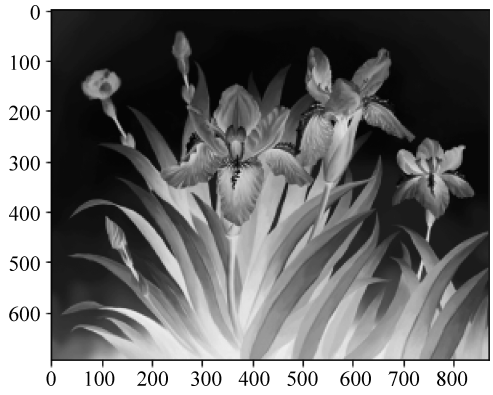


将图像文件读到数组中后，就可以通过对数组的操作实现图像变换的功能。

1. 改变像素点颜色值

下面通过简单的算术运算改变图像中像素点的颜色值。

```
In[128]:
    plt.imshow([255,255,255] - im)
    plt.show()
```

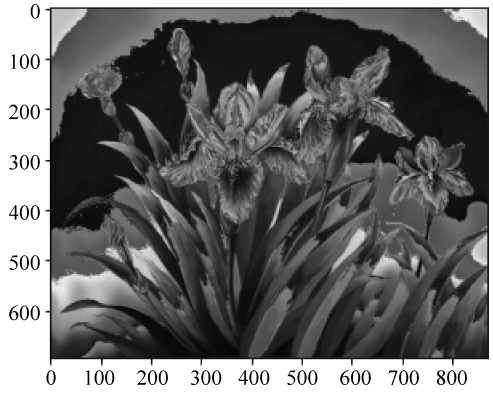


上面的操作中，[255,255,255]-im 是一个简单的数组算术运算，两个运算对象一个是一维的，一个是三维的，那么必然会涉及广播机制，操作结果就是用[255,255,255]分别减每个像素点的 R、G、B 三种颜色的数值，再输出每个点的颜色都改变了的图像。

2. 灰度变换

灰度变换是图像处理中常见的操作，使用数组同样可以很轻松实现，代码如下：

```
In[129]:
    R = im[:,:,0]
    G = im[:,:,1]
    B = im[:,:,2]
    L = R * 299 / 1000 + G * 587 / 1000 + B * 114 / 1000          #灰度变换公式
    plt.imshow(L, cmap = "gray")
    plt.show()
```

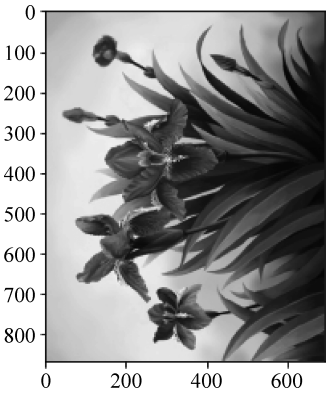


上述操作中，先通过索引操作，分别取出所有像素点在 R、G、B 三个通道上的颜色值，然后用灰度变换公式进行计算即可。

3. 改变图像纵横方向

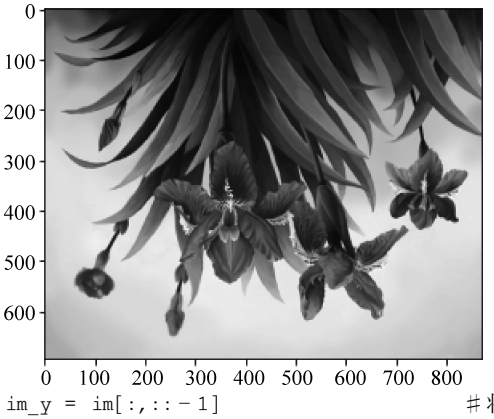
改变图像的纵横方向，实际上就是对数组进行轴对换操作，代码如下：

```
In[130]:
    plt.imshow(im.transpose(1,0,2))          #交换数组的 0 轴和 1 轴
    plt.show()
```

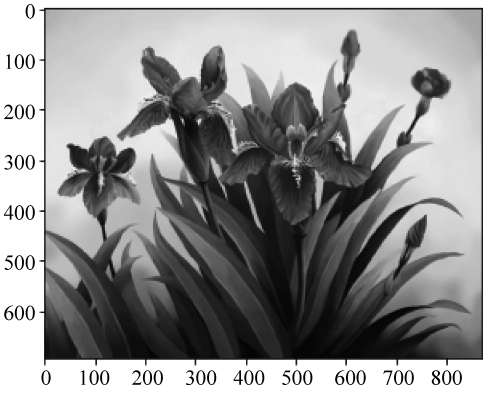


4. 图像翻转

```
In[131]:
    im_x = im[::-1]                          #将数组 0 轴整体翻转,实现图像纵向翻转
    plt.imshow(im_x)
    plt.show()
```




```
    im_y = im[:,::-1]                        #将数组 1 轴整体翻转,实现图像横向翻转
    plt.imshow(im_y)
    plt.show()
```

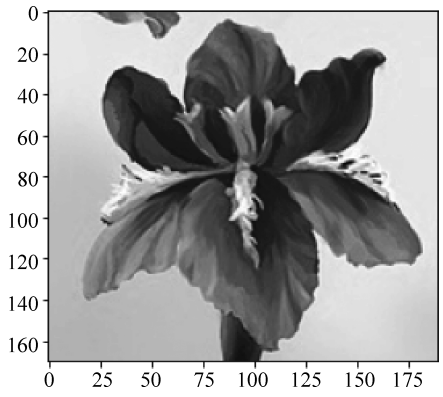


上述操作中，使用数组切片操作翻转数组的某一轴，实际上就是沿着横向或纵向将图像的像素点翻转位置，以此实现图像的翻转。

5. 图像裁剪

图像裁剪实现同样非常简单，只需在0轴和1轴上同时对数组进行切片操作即可，代码如下：

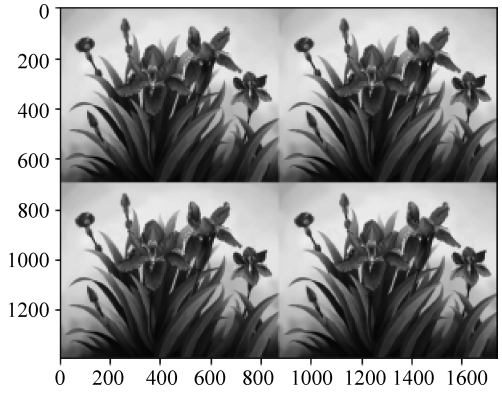
```
In[132]:
    plt.imshow(im[250:420,660:850])
    plt.show()
```



6. 图像拼接

图像拼接可以通过数组的合并轻松实现，代码如下：

```
In[133]:
    t1 = np.concatenate((im, im), axis = 1)          # 横向拼接
    t2 = np.concatenate((t1, t1), axis = 0)          # 横向拼接
    plt.imshow(t2)
    plt.show()
```

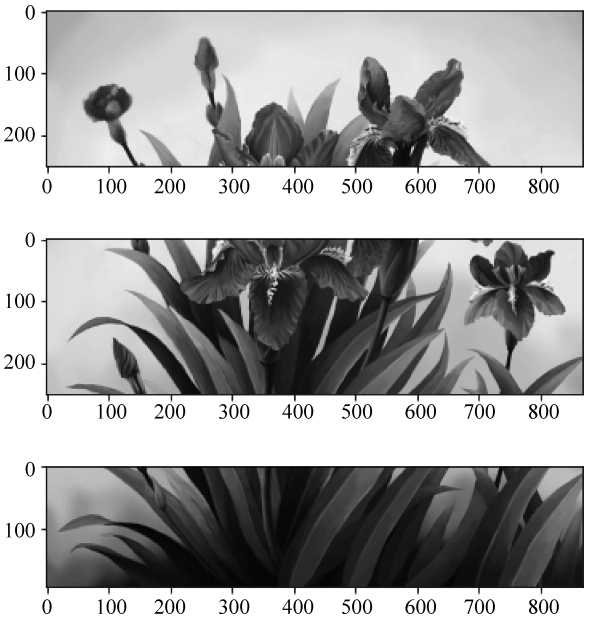


7. 图像分割

图像分割可以通过数组的拆分来实现，代码如下：

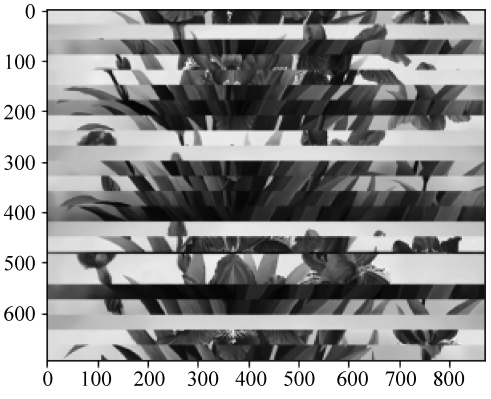
```
In[134]:
    t1,t2,t3 = np.vsplit(im,[250,500])
    plt.imshow(t1)
```

```
plt.show()
plt.imshow(t2)
plt.show()
plt.imshow(t3)
plt.show()
```



8. 随机打乱图片

```
In[135]:
    height = im.shape[0]
    li = np.split(im.copy(), range(30, height, 30), axis = 0)
    np.random.shuffle(li)
    t = np.concatenate(li, axis = 0)
    plt.imshow(t)
    plt.show()
```



以上操作中，沿 0 轴将数组拆分成多个部分，用 shuffle()函数随机排序后再合并在一起，达到了将图像随机打乱的效果。

简单总结上述这些图像变换，用到了 ndarray 数组的算术运算、随机函数，以及索引、切片、拆分、合并、轴对换等操作。更多的转换方式和操作技巧，读者可以进一步自行探索研究。

3.5 本章小结

NumPy 是 Python 科学计算及数据分析领域最基础的库，本章主要介绍了 NumPy 的基础知识，包括 NumPy 核心对象 ndarray 数组的各种创建方法和属性；数组的索引、切片、形状变换、合并拆分等基本操作；ndarray 数组的各种运算方法及广播机制、常用的排序函数、统计方法以及线性代数运算函数等。