第5章

NumPy基础与应用

NumPy是应用广泛的Python的一个基础科学计算包，是Python生态系统中数值计算的基石，目前许多主流的算法以及数据分析库，例如Pandas、sklearn(全称为scikit-learn)等都是基于NumPy发展而来的。在本章中主要介绍NumPy的核心——数组，包括数组的基础知识以及与数组相关的操作。

本章要点：

- NumPy数组的基本属性。
- NumPy数组的创建方法。
- NumPy数组的相关操作。
- NumPy数组的读/写。

5.1 NumPy简介

NumPy是Python的一个基础科学计算包，许多高级的第三方科学计算的模块都是基于NumPy所构建的，例如Pandas等。NumPy有以下几个特点：

(1) 强大的多维数组对象。

(2) 精细而复杂的功能。

(3) 用于集成C/C++和FORTRAN代码的工具。

(4) 实用的线性代数、傅里叶变换和随机数功能。

NumPy除了用于科学计算以外，还可以用作通用数据的高效多维容器。同时，NumPy支持自定义数组的数据类型，这使NumPy能够无缝、快速地与各种数据库集成，也使得NumPy在性能上比Python自身的嵌套列表结构高效很多。NumPy本身并没有提供多少高级的数据分析功能，但熟练掌握NumPy有助于更加高效地使用Pandas等诸多进行数据分析的库。

对于许多用户而言，尤其是Windows用户，学习Python的常用工具为Anaconda，其中包括所有数据分析的关键包。本书的相关练习操作均使用Anaconda中的Jupyter Notebook。在Anaconda环境中已经集成了NumPy模块，不需要再次安装，但可以在命令行中使用pip更新：

```
pip install numpy -U
```

用户可以在“https://numpy.org/doc/stable/”上查阅 NumPy 的使用文档。

5.2 NumPy 数组基础

数组是 NumPy 中的核心类型，全称为 **N 维数组**（N-dimensional Array，ndarray），整个 NumPy 模块都是围绕数组来构建的，它是一个固定大小和形状的大数据集容器，该对象由两部分组成，即实际的数据以及描述这些数据的元数据。大部分数组操作仅修改元数据部分，而不改变底层的实际数据。

在使用数组之前，首先需要导入 NumPy 模块。在本章例子中均默认已导入 NumPy 模块，导入语句为：

```
import numpy as np
```

5.2.1 数组的属性

ndarray 是 NumPy 中的数组类，也被称为 array，它是同构多维数组，可以存储相同类型、以多种形式组织的数据。NumPy.array 和标准的 Python 库中的 array.array 是不一样的，标准的 Python 库中的 array.array 只能处理一维数组，且所有元素的类型必须是一致的，支持的数据类型有整型、浮点型以及复数型，但这些类型不足以满足科学计算的需求，因此 NumPy 中添加了许多其他的数据类型，例如 bool、int、int64、float32、complex64 等。此外，NumPy 数组也有特有的属性，表 5-1 列出了几种常见的重要属性。

表 5-1 NumPy 数组的常见属性

属 性	说 明
.ndim	秩，即轴的数量或维度的数量
.shape	ndarray 对象的大小，对于矩阵，为 n 行 m 列
.size	ndarray 对象中元素的个数，相当于.shape 中 n×m 的值
.dtype	ndarray 对象的元素类型
.itemsize	ndarray 对象中每个元素的大小，以字节为单位
.nbytes	ndarray 对象中元素所占空间的大小

下面通过实例来介绍 NumPy 数组的常用属性。

例 5-1 构建一个二维数组 arr，并查看数组的属性。

首先构建数组 arr，代码及输出结果如下：

```
In:
    import numpy as np
    arr = np.array([[1,2,3],
                    [4,5,6]])
    print(arr)
```

```
Out:
    arr([[1, 2, 3],
    [4, 5, 6]])
```

然后使用.ndim 属性查看数组的维度,也称为轴,代码及输出结果如下:

```
In:
    print('数组的维度为: ',arr.ndim)
Out:
    数组的维度为: 2
```

除了秩属性以外,还可以使用.shape 属性查看数组的大小。对于一个具有 n 行、m 列的矩阵,形状为(n,m)。使用.shape 查看数组的大小,代码及输出结果如下:

```
In:
    print('数组的形状为: ',arr.shape)
Out:
    数组的形状为: (2, 3)
```

使用数组的.size 属性可以查看数组元素的个数,数值相当于.shape 中 n×m 的结果值,代码及输出结果如下:

```
In:
    print('数组元素的个数:',arr.size)
Out:
    数组元素的个数: 6
```

与列表不同,NumPy 数组要求所有元素都是同一类型,使用数组的.dtype 属性可以查看数组中保存的数据类型。此外,用户不仅可以使用标准的 Python 类型创建或指定 dtype,还可以使用 NumPy 提供的类型,例如 numpy.int32、numpy.int16、numpy.float64。使用.dtype 属性查看数组中保存的数据类型,代码及输出结果如下:

```
In:
    print('查看数组元素的类型:',arr.dtype)
Out:
    数组元素的类型: int32
```

用户还可以使用数组的.itemsize 属性查看数组中每个元素所占的字节数,即每个元素的大小,代码及输出结果如下:

```
In:
    print('查看元素所占的字节:',arr.itemsize)
Out:
    数组元素所占的字节: 4
```

数组需要开辟出一段连续的内存来存放数据,可以使用.nbytes 属性查看数组中所有元素所占的空间,在数值上等于属性.itemsize 和属性.size 的乘积。使用.nbytes 属性查看

数组中所有元素所占的空间，代码及输出结果如下：

```
In:
    print('数组所占内存的大小:',arr.nbytes)
Out:
    数组所占内存的大小: 24
```

以上是对数组的几种常见的重要属性的说明，对于其他属性，读者可根据实际开发需要查看 NumPy 的官方文档。

5.2.2 创建数组

用户可以采用多种方式来创建 ndarray 数组对象，例如结合 NumPy 中提供的多种函数生成数组、用随机数生成数组等。下面结合具体实例说明数组的生成方法，在实例中主要创建一维数组和二维数组。

1. 利用 np.array()函数生成数组

此方法是利用 np.array()函数来创建，函数的参数可以是元组、列表，也可以是另一个数组。其语法格式为：

```
X = np.array(list/tuple) 或者 X = np.array(list/tuple, dtype = np.dtype)
```

当 np.array()不指定 dtype 时，NumPy 将根据数据情况自动匹配一个 dtype 类型。

当 np.array()的参数是列表时，依据列表的维度分别创建一维数组或者二维数组。在例 5-2 中，可以看出当列表元素是整数型时，NumPy 为数组分配的类型为 int32。当创建二维数组时，只需要将列表以逗号隔开。

例 5-2 使用 np.array()函数创建一维数组和二维数组，参数为列表。

```
In:
    a = np.array([1,2,3,4])
    print('创建的一维数组为: ',a,'数组的元素类型为: ',a.dtype)
    d = np.array([[1,2,3],[4,5,6]])
    print('创建的二维数组为: \n',d)
Out:
    创建的一维数组为: [1 2 3 4] 数组的元素类型为: int32
    创建的二维数组为:
    [[1 2 3]
    [4 5 6]]
```

另外，可以用元组作为 np.array()的参数，创建一维数组或者二维数组。此时需要注意不能省略元组的括号，也可另外指定数组元素的数据类型。

例 5-3 使用 np.array()函数创建数组，参数为元组，并指定数据类型。

```
In:
    b = np.array((1,2,3,4),dtype = np.float32)
    print('创建的数组为{},参数为元组,数组元素的数据类型为:{}'.format(b,b.dtype))
Out:
    创建的数组为[1. 2. 3. 4.],参数为元组,数组元素的数据类型为:float32
```

2. 利用内置函数产生特定形式的数组

此方法是利用 NumPy 中内置的特殊函数来创建 ndarray 对象，例如 zeros()、ones()、empty()等。常用内置函数的功能见表 5-2。

表 5-2 NumPy 的常用内置函数

函　数	说　明
np.arange(n)	类似 range()函数，返回 ndarray 类型，元素为 0～n－1
np.ones(shape)	根据 shape 生成一个全 1 数组，shape 是元组类型，默认为浮点型
np.zeros(shape)	根据 shape 生成一个全 0 数组，shape 是元组类型，默认为浮点型
np.full(shape,val)	根据 shape 生成一个数组，每个元素值都是 val
np.eye(n)	创建一个正方的 n×n 单位矩阵，对角线为 1，其他为 0
np.linspace()	生成由等差数列构成的一维数组
np.empty()	生成一个指定形状和类型且全为空的数组

下面通过实例介绍上述函数的用法。

1）np.zeros()函数和 np.ones()函数

语法格式为：

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

例 5-4 使用 np.zeros()函数创建全 0 数组。

```
In:
    zero = np.zeros((2,3))              #生成 2 行 3 列全为 0 的矩阵
    display(zero)
Out:
    array([[0., 0., 0.],
[0., 0., 0.]])
```

np.zeros()函数用于生成一个数值全为 0 的数组，其中的行数和列数可以自定义。数值类型默认为浮点型。np.ones()函数的用法与 np.zeros()函数完全相同，只是生成的是全 1 数组。

2）np.arange()函数

np.arange()函数可以产生一个等距数组，其语法格式为：

```
X = np.arange([ start, ] stop[,step, ],dtype = None)
```

其中，[ ]中的内容可以省略，start 的默认值为 0，step 的默认值为 1，因此参数个数可以为 1～3 个。当只含一个参数时（即只有 stop 参数），将生成一个 0～stop－1、步长为 1 的整数数组。

例 5-5 使用 np.arange()创建数组，分别带有 1～3 个参数。

首先创建只带 1 个参数的数组，代码及输出结果如下：

```
In:
    arange1 = np.arange(10)
```

```
    display(arange1)
Out:
    array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

当含 2 个参数时，即指定起始位置，生成一个 start～stop－1、步长为 1 的整数数组，代码及输出结果如下：

```
In:
    arange2 = np.arange(4,10)
    display(arange2)
Out:
    array([4, 5, 6, 7, 8, 9])
```

当含 3 个参数时，即指定起始位置与步长，生成一个 start～stop－1、步长为定值的数组，指定的步长可以是浮点型。

```
In:
    arange3 = np.arange(4,8,0.5)
    display(arange3)
Out:
    array([4. , 4.5, 5. , 5.5, 6. , 6.5, 7. , 7.5])
```

3）np.linspace()函数

NumPy 中的 linspace()函数可以用来创建由等差数列构成的一维数组，该函数的格式如下：

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

该函数的参数及说明如表 5-3 所示。

表 5-3 np.linspace()函数的参数及说明

参　数	说　明
start	序列的起始值
stop	序列的终止值，如果 endpoint 为 True，则该值包含于数列中
num	要生成的等步长的样本数量，默认为 50
endpoint	当该值为 True 时，数列中包含 stop 值，否则不包含，默认为 True
retstep	如果为 True，生成的数组中会显示间距，不显示
dtype	ndarray 的数据类型

例 5-6 使用 linspace()函数生成均匀分布的数组。

```
In:
    arr1 = np.linspace(1,10,10)
    print('设置起始值为 1、终止值为 10、元素个数为 10 的数组:\n',arr1)
    arr2 = np.linspace(1,1,10)
    print('数组元素全部为 1 的数列:\n',arr2)
    arr3 = np.linspace(10,20,5, endpoint = False)
    print('生成 10 到 20,元素个数为 5 的数组,设置 endpoint 为 False,则数组中不包含终止值 20:
\n',arr3)
```

```
    arr4 = np.linspace(1,10,10).reshape([2,5])
    print('将数组形状设置为 2 行 5 列:\n',arr4)
Out:
    设置起始值为 1、终止值为 10、元素个数为 10 的数组:
    [ 1. 2. 3. 4. 5. 6. 7. 8. 9. 10.]
    数组元素全部为 1 的数列:
    [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
    生成 10 到 20,元素个数为 5 的数组,设置 endpoint 为 False,则数组中不包含终止值 20:
    [10. 12. 14. 16. 18.]
    将数组形状设置为 2 行 5 列:
    [[ 1. 2. 3. 4. 5.]
    [ 6. 7. 8. 9. 10.]]
```

4) np.empty()函数

np.empty()函数用于生成一个指定形状和类型全为空的数组,该函数只是让系统分配指定大小的内容,并没有初始化,里面的值是随机的。

例 5-7 使用 np.empty()生成一个指定形状和类型全为空的数组。

```
In:
    empty = np.empty(2)
    display(empty)
Out:
    array([ 2.00000047, 512.00012255])
```

3. 生成随机数组

在 NumPy 的 random 模块中包含了很多生成随机数字的函数,例如 random.rand()、random.randint()、random.randn()等。这些函数的用法相近,均是返回指定范围内的一个或一组随机整数或浮点数。对这些函数的简要说明如下:

1) numpy.random.rand(d0,d1,…,dn)函数

该函数创建给定形状的数组,并使用[0.0, 1.0)上均匀分布的随机浮点数填充数组。当函数没有参数时,返回一个随机浮点数;当函数有一个参数时,返回该参数长度大小的一维随机浮点数数组;当函数有两个或两个以上参数时,返回对应维度的数组。

2) numpy.random.randint(low, high=None, size=None, dtype=int)函数

该函数返回随机整数数组,数据值位于半开区间[low, high)。用户可以使用 size 设置数组的形状,例如,若 size 为(m,n,k),则绘制 m×n×k 形状的样本。size 的默认值为"None",在这种情况下将返回单个值。

3) numpy.random.randn(d0,d1,…,dn)函数

该函数返回一个或一组符合标准正态分布的随机样本值。当函数没有参数时,返回一个浮点数;当函数有一个参数时,返回秩为 1 的数组,不能表示向量和矩阵;当函数有两个或两个以上参数时,返回对应维度的数组,能表示向量或矩阵。此外,np.random.standard_normal()函数与 np.random.randn()函数类似,但是 np.random.standard_normal()函数的输入参数为元组(tuple)。

4）numpy.random.shuffle(x)函数

类似洗牌操作，修改参数 x。参数 x 为要洗牌的数组、列表或可变序列。

5）numpy.random.permutation(x)函数

该函数随机排列一个序列，或返回一个排列的范围。如果 x 是一个多维数组，则它只会沿着其第一个索引移动。

基于以上这些方法可以产生随机数组，下面通过实例进行介绍。对于更加详细的使用 NumPy 产生随机数组的方法，读者可以参考 NumPy 的官方文档。

例 5-8 使用 random 模块生成随机数组。

```
In:
    import random
    arr1 = np.random.rand(5)
    print('rand()生成长度为 5 的一维随机浮点数数组：\n',arr1)
    arr2 = np.random.rand(3,2)
    print('rand()生成 3 行 2 列的随机小数数组：\n',arr2)
    arr3 = np.random.randint(5,10,size = (2,3))
    print('randint()生成 2 行 3 列的随机整数数组，整数在[5,10)内：\n',arr3)
    arr4 = np.random.randn(10)
    print('randn()生成一组符合标准正态分布的随机样本值：\n',arr4)
    arr5 = np.arange(10)
    np.random.shuffle(arr5)
    print('shuffle()打乱数组的顺序：\n',arr5)
    listA = [0,1,2,3,4,5]
    arr6 = np.random.permutation(listA)
    print('permutation()返回对 listA 随机排序的结果：\n',arr6)
Out:
    rand()生成长度为 5 的一维随机浮点数数组：
    [0.09306943 0.62932793 0.14739025 0.1183968 0.64784539]
    rand()生成 3 行 2 列的随机小数数组：
    [[0.5400546 0.09572838]
    [0.06671314 0.29834428]
    [0.93913566 0.44298513]]
    randint()生成 2 行 3 列的随机整数数组，整数在[5,10)内：
    [[9 6 6]
    [6 5 6]]
    randn()生成一组符合标准正态分布的随机样本值：
    [-1.1437724 0.71268671 -0.47366688 0.98353624 0.56187371 -1.35199912
    -1.89082035 -0.30280619 -1.18260278 -0.13149373]
    shuffle()打乱数组的顺序：
    [2 7 6 0 1 4 5 9 3 8]
    permutation()返回对 listA 随机排序的结果：
    [1 4 5 3 2 0]
```

5.2.3 数组的数据类型

本章前面已经提到数组元素的类型，ndarray 数组的元素类型在 Python 列表元素类型的基础上进行了补充与扩展，从而使 ndarray 的使用范围更加广泛。NumPy 数组的数据类型如表 5-4 所示。

表 5-4 NumPy 数组的数据类型

数 据 类 型	说 明
bool	布尔类型，True 或 False
intc	与 C 语言中的 int 类型一致，一般是 int32 或 int64
intp	用于索引的整数，为 int32 或 int64
int8	字节长度的整数，取值为[－128，127]
int16	16 位长度的整数，取值为$[-2^{15},2^{15}-1]$
int32	32 位长度的整数，取值为$[-2^{31},2^{31}-1]$
int64	64 位长度的整数，取值为$[-2^{63},2^{63}-1]$
uint8	8 位无符号整数，取值为$[0,2^{8}-1]$
uint16	16 位无符号整数，取值为$[0,2^{16}-1]$
uint32	32 位无符号整数，取值为$[0,2^{32}-1]$
uint64	64 位无符号整数，取值为$[0,2^{64}-1]$
float16	16 位半精度浮点数：1 位符号位，5 位指数，10 位尾数
float32	32 位半精度浮点数：1 位符号位，8 位指数，23 位尾数
float64	64 位半精度浮点数：1 位符号位，11 位指数，52 位尾数
complex64	复数类型，实部和虚部都是 32 位浮点数
complex128	复数类型，实部和虚部都是 64 位浮点数

当使用元组或列表作为创建数组的参数时，若数据类型不同，会将其类型统一为类型范围较大的一种，即当传入的数据有多种类型时，NumPy 会自动进行判断，将数组转化为最通用的类型。

例 5-9 创建整型和浮点型的混合类型数组，NumPy 将其统一改为浮点型。

```
In:
    c = np.array([[1,2],[3,4],(0.1,0.8)])
    print('数组的数据类型为：',c.dtype)
Out:
    数组的数据类型为：float64
```

5.2.4 数组的迭代

数组是一种支持迭代的对象，可以灵活地访问一个或者多个数组元素。对于一维数组而言，迭代返回对应数组中的每一个元素；对于多维数组，迭代只针对数组的第一维进行迭代，故每次的返回值是一个维度减 1 的数组。

例 5-10 一维和多维数组迭代。

```
In:
    print('对于一维数组而言，迭代返回对应数组中的每一个元素：')
    mm = np.arange(1,4)
    for i in mm:
        print(i)
    print('对于多维数组，迭代只针对数组的第一维进行迭代：')
```

```
    nn = np.array([[1,2,3],[4,5,6]])
    for i in nn:
        print(i)
Out:
    对于一维数组而言,迭代返回对应数组中的每一个元素:
    1
    2
    3
    对于多维数组,迭代只针对数组的第一维进行迭代:
    [1 2 3]
    [4 5 6]
```

在上述实例中也可以理解为按行迭代。此外还可以进行列迭代和逐个元素迭代,如例 5-11 所示。

例 5-11 多维数组的列迭代和逐个元素迭代。

```
In:
    print('多维数组的列迭代: ')
    nn = np.array([[1,2,3],[4,5,6]])
    for i in nn.T:
        print(i)
    print('多维数组的逐个元素迭代: ')
    nn = np.array([[1,2],[4,5]])
    for i in nn.flat:
        print(i)
Out:
    多维数组的列迭代:
    [1 4]
    [2 5]
    [3 6]
    多维数组的逐个元素迭代:
    1
    2
    4
    5
```

5.2.5 数组的索引和切片

NumPy 的 ndarray 数组对象的内容可以通过索引和切片来访问和修改,与 Python 中 list 的切片操作一样。ndarray 数组可以基于 0～n 的下标进行索引,切片对象可以通过内置的 slice()函数,并设置 start、stop 及 step 参数进行,目的是从原数组中切割出一个新数组。另外,也可以通过冒号分隔切片参数(start:stop:step)来进行切片操作。

1. 一维数组的索引和切片

一维数组的索引与列表类似,不仅支持单个元素的索引,还支持负数的索引与切片。在例 5-12 中,首先创建一个一维数组 arr1。

例 5-12 一维数组的索引。

```
In:
    arr1 = np.arange(1,10)
    print('一维数组 arr1: ',arr1)
    print('单个元素的索引,提取第 2 个位置的数: ',arr1[2])
    print('从第 2 到倒数第 1 个位置的数据,不包括最后一个: ',arr1[2:-1])
    print('从索引 2 开始到索引 7 停止,间隔为 2: ',arr1[2:7:2])
    s = slice(2,7,2)
    print('用 slice()函数切片: ',arr1[s])
    print('取第 1～4 个位置上的数据(包前不包后): ',arr1[1:4])
    print('取前 5 个数据: ',arr1[:5])
    print('取最后两个数据: ',arr1[-2:])
Out:
    一维数组 arr1: [1 2 3 4 5 6 7 8 9]
    单个元素的索引,提取第 2 个位置的数: 3
    从第 2 到倒数第 1 个位置的数据,不包括最后一个: [3 4 5 6 7 8]
    从索引 2 开始到索引 7 停止,间隔为 2: [3 5 7]
    用 slice()函数切片: [3 5 7]
    取第 1～4 个位置上的数据(包前不包后): [2 3 4]
    取前 5 个数据: [1 2 3 4 5]
    取最后两个数据: [8 9]
```

如该例所示,在索引时只放置一个参数,例如 arr1[2],将返回与该索引相对应的单个元素。如果为 arr1[2:],表示从该索引开始以后的所有项都将被提取。如果使用了两个参数,例如 arr1[2:7],那么将提取两个索引(不包括停止索引)之间的项。同时,负索引或切片后得到的结果仍是一个数组,可以进行数组的相关操作。

2. 多维数组的索引和切片

多维数组的索引与多维列表不同,只能通过多重索引的方法得到其中的元素。对于单个元素的索引,采用 array[行数][列数]的方法。在例 5-13 中,首先创建一个二维数组 arr2。

例 5-13 多维数组的索引。

```
In:
    arr2 = np.arange(12).reshape(3,4)
    print('二维数组 arr2: \n',arr2)
    print('取二维数组中第 1 行第 2 列的数据: ',arr2[1][2])
    print('取二维数组中第 1 行的数据: ',arr2[1])
    print('取二维数组中所有行的第 2 列数据: ',arr2[:,2])
    print('取二维数组中第 0～1 行第 1～2 列的数据: \n',arr2[0:2,1:3])
Out:
    二维数组 arr2:
    [[ 0 1 2 3]
    [ 4 5 6 7]
    [ 8 9 10 11]]
    取二维数组中第 1 行第 2 列的数据: 6
```

```
    取二维数组中第 1 行的数据: [4 5 6 7]
    取二维数组中所有行的第 2 列数据: [ 2 6 10]
    取二维数组中第 0～1 行第 1～2 列的数据:
    [[1 2]
    [5 6]]
```

如该例所示,对于某一行元素的索引,采用 array[行数]或者 array[行数,:]的方法;对于某一列元素的索引,采用 array[:,列数]的方法;对于某几行和某几列的索引,采用[起始行:终止行,起始列:终止列]的方法。

5.2.6 数组的合并与拆分

NumPy 数组支持水平、竖向和深向的合并与拆分。

1. 数组的合并

数组的合并支持 3 个维度上的操作,即水平、竖向和深向,使用的函数分别是 hstack()、vstack()、dstack()、column_stack()、row_stack()和 concatenate()等。

在水平方向上,一般采用 hstack()函数实现两个数组的合并,具体语法为 np.hstack((a,b))。此外,concatenate()函数和 column_stack()函数可以实现类似的效果。在竖向方向上,一般采用 vstack()函数实现两个数组的合并,具体语法为 np.vstack((a,b))。当设置参数 axis 为 0 时,使用 concatenate()函数可以得到同样的结果。row_stack()函数也是如此。另外,还有一种深向合并的操作,它是沿着第 3 个坐标轴的方向来合并,使用到的函数是 dstack()。

注意,在 NumPy 中关键字轴(axis)是为超过一维的数组定义的属性。一般而言,二维数据拥有两个轴——第 0 轴沿着行的方向垂直往下,第 1 轴沿着列的方向水平延伸,即使用 axis=0 表示沿着每一列或行标签/索引值向下执行方法,使用 axis=1 表示沿着每一行或者列标签/索引值水平执行方法。

为解释对应函数的意义,首先给定需要的数组。为了方便,下面直接以二维数组为例,一维数组的使用与之相同。

例 5-14 数组的水平方向上的合并。

```
In:
    a = np.arange(6).reshape(2,3)
    b = a+10
    print('数组 a: \n',a)
    print('数组 b: \n',b)
    print('采用 hstack()函数实现两个数组的水平方向上的合并: \n',np.hstack((a,b)))
    print('采用 concatenate()函数实现两个数组的水平方向上的合并: \n',np.concatenate((a,b),
axis=1))
    print('采用 column_stack()函数实现两个数组的水平方向上的合并: \n',np.column_stack((a,b)))
Out:
    数组 a:
    [[0 1 2]
    [3 4 5]]
```

```
    数组 b:
    [[10 11 12]
    [13 14 15]]
    采用 hstack()函数实现两个数组的水平方向上的合并:
    [[ 0 1 2 10 11 12]
    [ 3 4 5 13 14 15]]
    采用 concatenate()函数实现两个数组的水平方向上的合并:
    [[ 0 1 2 10 11 12]
    [ 3 4 5 13 14 15]]
    采用 column_stack()函数实现两个数组的水平方向上的合并:
    [[ 0 1 2 10 11 12]
    [ 3 4 5 13 14 15]]
```

例 5-15 数组的竖向及深向合并。

```
In:
    print('采用 vstack()函数实现两个数组的竖向合并: \n',np.vstack((a,b)))
    print('采用 dstack()函数实现两个数组的深向合并: \n',np.dstack((a,b)))
Out:
    采用 vstack()函数实现两个数组的竖向合并:
    [[ 0 1 2]
    [ 3 4 5]
    [10 11 12]
    [13 14 15]]
    采用 dstack()函数实现两个数组的深向合并:
    [[[ 0 10]
      [ 1 11]
      [ 2 12]]
    [[ 3 13]
      [ 4 14]
      [ 5 15]]]
```

注意,本例中进行合并的两个数组与例 5-14 相同。

2. 数组的拆分

用户还可以在水平、竖向和深向 3 个方向上拆分数组,相关函数有 hsplit()、vsplit()、dsplit()和 split()等。

对于一个 n×m 的数组,可以沿着水平方向(按列)将其拆分为 m 部分,并且各部分的大小和形状完全相同,可以用 hsplit()函数实现水平拆分。此外,使用 split()函数也可以实现相同的结果,需要把 axis 设置为 1。对于一个 n×m 的数组,也可以沿着竖向(按行)将其拆分为 n 部分,并且各部分的大小和形状完全相同,可以用 vsplit()函数实现竖向拆分。另外,调用参数 axis 为 0 的 split()函数也可以实现相同的效果。使用 dsplit()函数可以实现沿着深向方向分解数组,但它必须作用在三维及三维以上的数组,此处不给出实例说明。

例 5-16 数组的拆分。

```
In:
    print('采用 hsplit()函数实现水平拆分(按列拆分): \n',np.hsplit(a,3))
```

```
    print('采用 vsplit()函数实现竖向拆分(按行拆分): \n',np.vsplit(a,2))
    print('采用参数 axis 为 0 的 split()函数实现竖向拆分(按行拆分): \n',np.split(a,2,axis = 0))
Out:
    采用 hsplit()函数实现水平拆分(按列拆分):
    [array([[0],
            [3]]), array([[1],
            [4]]), array([[2],
            [5]])]
    采用 vsplit()函数实现竖向拆分(按行拆分):
    [array([[0, 1, 2]]), array([[3, 4, 5]])]
    采用参数 axis 为 0 的 split()函数实现竖向拆分(按行拆分):
    [array([[0, 1, 2]]), array([[3, 4, 5]])]
```

5.3 数组的相关操作

5.3.1 统计相关操作

NumPy 提供了很多统计函数,可以支持对数组的多种统计操作。例如从数组中查找最小元素、最大元素,计算均值、标准差和方差等。常见的统计函数及说明见表 5-5。

表 5-5 常见的统计函数及说明

函 数	说 明
.sum()	数组求和,默认对所有元素求和
.prod()	数组求积,默认对所有元素求积
.max()	求数组元素中的最大值
.argmax()	求数组元素中最大值的位置
.average()	根据在另一个数组中给出的各自的权重,计算数组中元素的加权平均值
.amax()	计算数组中的元素沿指定轴的最大值
.std()	求数组元素的标准差
.median()	计算数组 a 中元素的中位数(中值)
.ptp()	计算数组中元素的最大值与最小值的差(最大值-最小值)
.min()	求数组元素中的最小值
.argmin()	求数组中元素中最小值的位置
.mean()	求数组中所有元素的均值
.amin()	计算数组中的元素沿指定轴的最小值
.var()	求数组元素的方差

下面以求和等函数为例说明 NumPy 中数组的统计相关操作。

例 5-17 数组的统计相关操作实例。

```
In:
    a = np.arange(6).reshape(2,3)
    print('原数组为: \n',a)
    print('对数组的所有元素求和: ',a.sum())
    print('指定维数求和,返回一个数组: ',a.sum(axis = 1))
```

```
    print('计算数组中的元素沿轴 axis = 1 方向的最大值: ',np.amax(a,axis = 1))
    print('沿轴 axis = 0 调用 ptp()函数,计算数组中元素的最大值与最小值的差: ',np.ptp(a,
axis = 0))
    wt = np.array([3,2,1])
    print('指定权重数组,调用 average()函数,沿轴 axis = 1 计算数组中元素的加权平均值: ',
np.average(a, axis = 1, weights = wt))
    print('调用 std()函数,计算数组的标准差: ',np.std(a))
Out:
    原数组为:
    [[0 1 2]
    [3 4 5]]
    对数组的所有元素求和: 15
    指定维数求和,返回一个数组: [ 3 12]
    计算数组中的元素沿轴 axis = 1 方向的最大值 : [2 5]
    沿轴 axis = 0 调用 ptp()函数,计算数组中元素的最大值与最小值的差: [3 3 3]
    指定权重数组,调用 average()函数,沿轴 axis = 1 计算数组中元素的加权平均值: [0.66666667
3.66666667]
    调用 std()函数,计算数组的标准差: 1.707825127659933
```

注意,在用 average()计算加权平均值时,参数 weights 是与数组 a 中值关联的权重数组。a 中的每个值根据其关联的权重对平均值做出贡献。权重数组可以是一维(在这种情况下,其长度必须是沿给定轴 a 的大小),也可以是与 a 相同的形状。如果权重为 None,则假定 a 中的所有数据权重为 1。一维数组求平均值的方法为 avg=sum(a * weights)/sum(weights)。

对于其他更多的方法,请读者参考 NumPy 官方文档并自行实践。

5.3.2 形状相关操作

1. 使用.shape 属性修改数组的形状

修改数组形状最常见的方法就是使用.shape 属性。假设有一个一维数组,将其变为一个二维数组,见例 5-18。

例 5-18 使用.shape 属性修改数组的形状。

```
In:
    a = np.arange(8)
    print('原数组为一维数组: ',a)
    a.shape = 2,4
    print('通过.shape 属性修改数组的形状: \n',a)
Out:
    原数组为一维数组: [0 1 2 3 4 5 6 7]
    通过.shape 属性修改数组的形状:
    [[0 1 2 3]
    [4 5 6 7]]
```

2. 不改变数组自身形状的方法:.reshape()方法

在对数组形状进行的操作中,使用.reshape()方法也可以实现类似.shape 属性的效果,

但是它不改变原数组的形状，并返回一个新的数组。注意，形状参数必须与数组的大小一致，否则会抛出异常。

例 5-19 使用.reshape()方法不改变原数组的形状。

```
In:
    a = np.arange(8)
    print('原数组 a 为一维数组: ',a)
    a.reshape(4,2)
    print('使用.reshape()方法不改变原数组的形状: ',a)
Out:
    原数组 a 为一维数组: [0 1 2 3 4 5 6 7]
    使用.reshape()方法不改变原数组的形状: [0 1 2 3 4 5 6 7]
```

此外，.reshape()方法可以接受一个“－1”作为参数，当某一维度是－1 时，NumPy 会自动根据其他维度来计算这一维度的大小，如例 5-20 所示。

例 5-20 使用“－1”作为.reshape()的参数。

```
In:
    a = np.arange(6)
    print('原数组 a 为一维数组: ',a)
    b = a.reshape( - 1,2)
    print('使用 - 1 作为.reshape()的参数,修改数组的形状,放到新数组 b 中: \n',b)
Out:
    原数组 a 为一维数组: [0 1 2 3 4 5]
    使用 - 1 作为.reshape()的参数,修改数组的形状,放到新数组 b 中:
    [[0 1]
    [2 3]
    [4 5]]
```

3. 改变数组自身形状的方法：.resize()方法

.resize()方法的作用与.reshape()方法的作用一致，但改变原数组的形状。其原理是首先将原数组转变为一维数组，再判断元素的个数，若缺失，则缺失元素以 0 补全；若过满，则只截取所需元素，从而实现对数组形状的修改。注意，.resize()方法不支持－1 作为参数。

例 5-21 使用.resize()方法改变数组自身的形状。

```
In:
    a = np.arange(8)
    print('原数组 a 为一维数组: ',a)
    a.resize(2,5)
    print('使用.resize()方法改变数组自身的形状: \n',a)
    a.resize(2,3)
    print('若过满,则只截取所需元素: \n',a)
Out:
    原数组 a 为一维数组: [0 1 2 3 4 5 6 7]
    使用.resize()方法改变数组自身的形状:
```

```
    [[0 1 2 3 4]
    [5 6 7 0 0]]
    若过满,则只截取所需元素:
    [[0 1 2]
    [3 4 5]]
```

4. 数组的转置：.T 属性和.transpose()方法

对于数组而言,转置可以通过.T 属性和.transpose()方法实现。在一般情况下,这两种方法是等价的。对于一维数组而言,转置返回其本身；对于二维数组而言,转置相当于将其行列互换；对于多维数组而言,转置是将所有的维度反向,即原来的第一维变成最后一维,原来的最后一维变成第一维。

例 5-22 数组的转置。

```
In:
    a = np.arange(6).reshape(3,2)
    print('原数组 a: \n',a)
    print('通过.T 属性实现数组的转置: \n',a.T)
    b = a.transpose()
    print('通过.transpose()方法实现数组的转置: \n',b)
Out:
    原数组 a:
    [[0 1]
    [2 3]
    [4 5]]
    通过.T 属性实现数组的转置:
    [[0 2 4]
    [1 3 5]]
    通过.transpose()方法实现数组的转置:
    [[0 2 4]
    [1 3 5]]
```

5. 数组的降维：.flat 属性、.flatten()方法和.revel()方法

数组的降维是针对多维数组而言的,可以利用.flat 属性得到数组中的一个一维引用数组,并且此属性支持修改对应数值,从而使原数组的对应位置有相应的改变。此外,使用.flatten()方法和.revel()方法也可以实现数组的一维化,但是.flatten()方法返回的是原数组的一个副本,修改它对原数组不产生影响。.revel()方法是返回引用,只有在必要时返回副本。

例 5-23 数组的降维。

```
In:
    a = np.arange(8).reshape(2,4)
    print('原数组 a: \n',a)
    print('使用.flat 属性得到数组中的一个一维引用数组: ',a.flat[-2])
    a.flat[3] = 100
```

```
    b = a.flatten()
    print('使用.flatten()方法实现数组的一维化: ',b)
Out:
    原数组 a:
    [[0 1 2 3]
    [4 5 6 7]]
    使用.flat 属性得到数组中的一个一维引用数组: 6
    使用.flatten()方法实现数组的一维化: [ 0 1 2 100 4 5 6 7]
```

5.3.3 数组的四则运算、点乘与比较操作

数组的四则运算是对数组的对应元素进行数值运算,可以是数组与数字、数组与数组之间的四则运算。

例 5-24 数组的四则运算、点乘与比较操作。

首先定义两个二维数组。对于单个数组,可以进行数组与数字之间的四则运算,规则是数组元素分别对应进行运算。

```
In:
    arr1 = np.array([[1,2,4],[4,6,8]])
    arr2 = np.array([[1,2,2],[2,3,4]])
    print('数组 arr1 为: \n',arr1)
    print('数组 arr2 为: \n',arr2)
    #所有元素 + 2,乘、除、减与之同理
    print('数组 arr1 的所有元素 + 2,结果为: \n',arr1 + 2)
Out:
    数组 arr1 为:
    [[1 2 4]
    [4 6 8]]
    数组 arr2 为:
    [[1 2 2]
    [2 3 4]]
    数组 arr1 的所有元素 + 2,结果为:
    [[ 3 4 6]
    [ 6 8 10]]
```

数组与数组之间也可以进行四则运算,前提是参与运算的数组具有相同的行列。运算规则是按位运算,对应位置进行运算。此外,数组还支持幂运算、取余和取整运算。

```
In:
    print('arr1 + arr2 的结果: \n',arr1 + arr2)
    print('arr1 - arr2 的结果: \n',arr1 - arr2)
    print('arr1 * arr2 的结果: \n',arr1 * arr2)
    print('arr1 ** arr2 (arr1 对应位置上 arr2 次幂)的结果: \n',arr1 ** arr2)
    print('arr1 / arr2 的结果: \n',arr1 / arr2)
    print('arr1 % arr2 取余运算的结果: \n',arr1 % arr2)
    print('arr1 // arr2 取整运算的结果: \n',arr1 // arr2)
Out:
    arr1 + arr2 的结果:
```

```
    [[ 2 4 6]
    [ 6 9 12]]
    arr1 - arr2 的结果:
    [[0 0 2]
    [2 3 4]]
    arr1 * arr2 的结果:
    [[ 1 4 8]
    [ 8 18 32]]
    arr1 ** arr2 (arr1 对应位置上 arr2 次幂)的结果:
    [[  1  4  16]
    [ 16 216 4096]]
    arr1 / arr2 的结果:
    [[1. 1. 2.]
    [2. 2. 2.]]
    arr1 % arr2 取余运算的结果:
    [[0 0 0]
    [0 0 0]]
    arr1 // arr2 取整运算的结果:
    [[1 1 2]
    [2 2 2]]
```

在数学上,二维数组可以看成矩阵,一维数组可以看成向量,它们还支持点乘运算,用.dot()函数实现。

```
In:
    print('用.dot()函数实现矩阵的乘法: \n',arr1.dot(arr2.T))
Out:
    用.dot()函数实现矩阵的乘法:
    [[13 24]
    [32 58]]
```

在上述实例中,用.dot()函数实现两个矩阵相乘,与 np.dot(arr1,arr2.T)的结果相同:

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}\begin{bmatrix}1 & 2\\1 & 3\\2 & 3\end{bmatrix}=\begin{bmatrix}9 & 17\\21 & 41\end{bmatrix}$$

数组还支持其他运算,例如比较和逻辑运算。在比较运算中,数组可以与一个数进行比较,返回一个布尔型数组;数组与数组之间也可以进行比较,返回一个布尔型数组。

```
In:
    arr3 = arr1 > 3
    print('数组的比较运算结果,返回布尔型数组: \n',arr3)
Out:
    数组的比较运算结果,返回布尔型数组:
    [[False False True]
    [ True True True]]
```

在上述实例中,比较 arr1 中的每一个元素与 3 的关系,若大于,返回 True,否则返回 False。此外,除了“>”以外,其他的比较符有“<”“>=”“<=”“==”和“!=”。

5.4 数组的读/写

在实际操作中使用的数据是很多的，那么从存放这些数据的文件中读/写数据就显得尤为重要。文件的格式主要有 CSV、TXT 等。

5.4.1 数组的读取

用户可以使用 np.loadtxt()函数从文本文件中读取数据，要求文件中的每一行必须具有相同数量的数据。该函数的语法格式如下：

```
    np.loadtxt(fname, dtype = <class 'float'>, comments = '#', delimiter = None, converters =
None, skiprows = 0, usecols = None, unpack = False, ndmin = 0, encoding = 'bytes', max_rows =
None, *, like = None)
```

np.loadtxt()函数的主要参数及说明如表 5-6 所示。

表 5-6 np.loadtxt()函数的主要参数及说明

主要参数	说　明
fname	文件、字符串或产生器，可以是.gz 或.bz2 格式的压缩文件
dtype	结果数组的数据类型，默认值为 float
delimiter	分隔字符串，默认是空格
skiprows	跳过前 n 行，包括注释，默认值为 0
usecols	要读取的列，0 是第 1 列。例如，usecols=(1,4,5)将提取第 2、5、6 列。默认值“None”将读取所有列
unpack	如果为 True，读入属性将分别写入不同变量

下面通过实例介绍如何从文件读取数据。

例 5-25 从文本文件中读取数据，读出的数据为数组类型。

首先读取 data 目录下以空格分隔数据的 test.txt 文件。

```
In:
    arr1 = np.loadtxt("data/test.txt")
    print('读出的数据类型为：',type(arr1))
    print('读出的数据为：\n',arr1)
Out:
    读出的数据类型为：<class 'numpy.ndarray'>
    读出的数据为：
    [[ 1. 2. 3. 4. 5. 6. 7.]
    [ 8. 9. 10. 11. 12. 13. 14.]]
```

如果文件中的数据不是以空格分隔，可以设置参数 delimiter 指定分隔符，例如读取以逗号分隔数据的 test1.txt 文件。

```
In:
    arr2 = np.loadtxt("data/test1.txt",delimiter = ',')
    print(arr2)
```

```
Out:
    [[ 1. 2. 3. 4. 5. 6. 7.]
    [ 8. 9. 10. 11. 12. 13. 14.]]
```

如果文件中包含字符串和数值型的混合数据(例如 test2.txt 中有'gender'、'age'、'weight'3 列数据),可以如下指定数据类型进行读取。

```
In:
    arr3 = np.loadtxt("data/test2.txt", dtype = {'names': ('gender', 'age', 'weight'),
                                                 'formats': ('S10', 'i4', 'f4')})
    print(arr3)
Out:
    [(b'Male', 21, 50.5) (b'Female', 22, 45.2)]
```

5.4.2 数组的写入

与读取文件类似,用户可以使用 np.savetxt()函数将数组写入文件,在写入时默认使用科学记数法的形式保存数字。该函数的语法格式如下:

```
    np.savetxt(fname, x, fmt = '%.18e', delimiter = ' ', newline = '\n', header = '', footer = '',
comments = '#', encoding = None)
```

np.savetxt()函数的主要参数及说明如表 5-7 所示。

表 5-7 np.savetxt()函数的主要参数及说明

主要参数	说 明
fname	文件、字符串或产生器,可以是.gz 或.bz2 格式的压缩文件
x	一维或二维数组,即要保存到文本文件的数据
fmt	单一格式(例如 10.5f)、格式序列或多格式字符串
delimiter	分隔字符串,默认是空格

例 5-26 数组的写入。

首先使用默认参数将一个二维数组写入文件,代码如下:

```
In:
    arr1 = np.array([[1,2,3],[4,5,6]])
    np.savetxt("data/write1.txt",arr1)
```

文件中存入的数据如下:

```
1.00000000000000000e+00 2.00000000000000000e+00 3.00000000000000000e+00
4.00000000000000000e+00 5.00000000000000000e+00 6.00000000000000000e+00
```

可以使用 fmt 参数设置数据写入的格式:

```
In:
    arr2 = np.array([[11.1,22.2,33.3],[44.4,55.5,66.6]])
    np.savetxt("data/write2.txt",arr2,fmt = "%d")
```