# 第 3 章

# NumPy 数组计算

NumPy 是 Numerical Python 的简称，是在 1995 年诞生的 Python 库 Numeric 的基础上建立起来的，属于 Python 语言的一个开源数值计算的扩展程序库。NumPy 包含数组计算、数值积分、傅里叶变换和线性代数运算等功能。它的主要用途是以数组的形式进行数据操作和数学运算。

NumPy 作为高性能科学计算和数据分析的基础库，是众多数据分析、机器学习工具的基础架构，掌握 NumPy 的功能及用法将有助于后续学习和使用其他数据分析工具。本章将针对 NumPy 库的基础功能进行详细的讲解。

## 3.1 NumPy 与数组对象

### 3.1.1 NumPy 概述

NumPy(官网：https://numpy.org)是 Python 数组计算、矩阵运算和科学计算的核心库。NumPy 这个词来源于 Numerical 和 Python 两个单词的结合。NumPy 提供了一个高性能的数组对象，让我们轻松创建一维数组、二维数组和多维数组，以及大量的函数和方法，帮助我们轻松地进行数组计算，从而广泛地应用于数据分析、机器学习、图像处理和计算机图形学、数学任务等领域当中。

NumPy 是以数组的形式对数据进行操作。尤其是机器学习中有大量的数组运算，而 NumPy 使得这些操作变得简单。由于 NumPy 是通过 C 语言实现的，所以其运算速度非常快。具体功能如下。

(1) 有一个强大的 $N$ 维数组对象 ndarray。

(2) 广播功能函数。所谓广播是一种对数组执行数学运算的函数，其执行的是元素级计算。广播提供了算术运算期间处理不同形状的数组集工具。

(3) 具有线性代数、傅里叶变换、随机数生成、图形操作等功能。

(4) 整合了 C/C++ /FORTRAN 代码的工具。

标准的 Python 用 list(列表)保持值，可以当作数组使用，但因为列表中的元素可以是任何对象，所以浪费了 CPU 运算的时间和内存。NumPy 的诞生弥补了这些缺陷，它提供了以下两种基本的对象。

(1) ndarray(n-dimensional array object)：存储单一数据类型的多维数组。

(2) ufunc(universal function object)：一种能够对数组进行处理的函数。

在应用 NumPy 前，必须先安装 NumPy 模块，使用 pip 工具，安装命令为：pip install numpy。

### 3.1.2　NumPy 数组对象

数组可分为一维数组、二维数组、三维数组，其中，三维数组是常见的多维数组，所有元素必须是相同类型。其中，一维数组很简单，基本和 Python 列表一样，区别在于数组切片针对的是原始数组（这就意味着，如果对数组进行修改，原始数组也会跟着更改）。二维数组的本质是以数组作为数组元素。二维数组包括行和列，类似于表格形状，又称为矩阵。三维数组是指维数为三的数组结构，也称矩阵列表。三维数组是最常见的多维数组，由于其可以用来描述三维空间中的位置或状态而被广泛使用。

在 NumPy 模块里有 axis（轴），通常用于指定某个 axis，就是沿着这个 axis 做相关操作，其中，二维数组的两个 axis 的指向如图 3-1 所示。

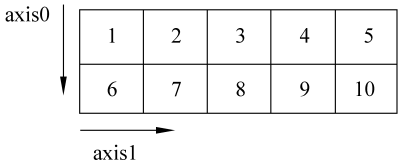

**图 3-1　二维数组两个轴**

对于一维数组，情况有点特殊，它不像二维数组从上向下地操作，而是水平的。

NumPy 中最重要的一个特点就是其 $N$ 维数组对象，即 ndarray（别名 array）对象，该对象具有向量算术能力和复杂的广播能力，可以执行一些科学计算。不同于 Python 标准库，ndarray 对象拥有对高维数组的处理能力，这也是数值计算中缺一不可的重要特性。ndarray 对象中定义了一些重要的属性，具体如表 3-1 所示。

**表 3-1　ndarray 对象的常用属性**

| 属　性 | 具 体 说 明 |
|---|---|
| ndarray.ndim | 维度个数，也就是数组轴的个数，如一维、二维、三维等 |
| ndarray.shape | 数组的维度。这是一个整数元组，表示每个维度上数组的大小。例如，一个 $n$ 行和 $m$ 列的数组，它的 shape 属性为（$n$，$m$） |
| ndarray.size | 数组元素的总个数，等于 shape 属性中元组元素的乘积 |
| ndarray.dtype | 描述数组中元素类型的对象，既可以使用标准的 Python 类型创建或指定，也可以使用 NumPy 特有的数据类型来指定，如 numpy.int32、numpy.float64 等 |
| ndarray.itemsize | 数组中每个元素的字节大小。例如，元素类型为 float64 的数组有 8（64/8）字节，这相当于 ndarray.dtype.itemsize |

值得一提的是，ndarray 对象中存储元素的类型必须是相同的。

**【例 3-1】**　查看 ndarray 对象的属性。

```
import numpy as np                    #使用 import…as 语句导入 NumPy 库，并将其取别名为 np
#创建一个 3 行 4 列的数组
ndata = np.arange(12).reshape(3, 4)
print("数组为:\n", ndata)
print("数组的类型为:", type(ndata))
#数组维度的个数，输出结果 2，表示二维数组
print("数组的维度个数:", ndata.ndim)
#数组的维度，输出结果(3,4)，表示 3 行 4 列
print("数组维度:", ndata.shape)
#数组元素的个数，输出结果 12，表示总共有 12 个元素
print("数组元素的总个数:", ndata.size)
#数组元素的类型，输出结果 dtype('int32')，表示元素类型都是 int32
print("数组元素的类型:", ndata.dtype)
```

运行结果：

```
数组为:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
数组的类型为:<class 'numpy.ndarray'>
数组的维度个数: 2
数组维度: (3, 4)
数组元素的总个数: 12
数组元素的类型: int32
```

上述示例中，arange()函数的功能类似于 range()，只不过 arange()函数生成的是一系列数字元素的数组；reshape()函数的功能是重组数组的行数、列数和维度。

## 3.2 创建 NumPy 数组

### 3.2.1 利用 array 函数创建数组

创建 ndarray 对象的方式有若干种，其中最简单的方式就是使用 array()函数，在调用该函数时传入一个 Python 现有的类型即可，如列表、元组。

NumPy 创建简单的数组主要使用 array()函数，语法格式如下。

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

参数说明如下。

- object：数组。公开数组接口的任何对象，__array__方法返回数组的对象，或任何(嵌套)序列。
- dtype：数据类型，可选。数组所需的数据类型。如果没有给出，那么类型将被确定为保持序列中的对象所需的最小类型。此参数只能用于“upcast”数组。对于向下转换，请使用.astype(t)方法。
- copy：bool 型，可选。如果为 True(默认值)，则复制对象。否则，只有当__array__返回副本，obj 是嵌套序列，或者需要副本来满足任何其他要求(dtype，顺序等)时，才会进行复制。
- order：元素在内存中的出现顺序，值为 {'K','A','C','F'}，可选。如果 object 参数不是数组，则新创建的数组将按行排列(C)，如果值为 F，则按列排列；如果 object 参数是一个数组，则以下内容成立：C(按行)、F(按列)、A(原顺序)、K(元素在内存中的出现顺序)。
- subok：bool 型，可选。如果为 True，则将传递子类，否则返回的数组将被强制为基类数组(默认)。
- ndmin：int，可选。指定结果数组应具有的最小维数。将根据需要预先设置形状。

**【例 3-2】** 通过 array()函数分别创建一个一维数组和二维数组。

```
import numpy as np
ndata1 = np.array([1, 2, 3])                    #创建一个一维数组
print("创建一维数组:\n", ndata1)
ndata2 = np.array([[1, 2, 3], [4, 5, 6]])       #创建一个二维数组
```

```
print("创建二维数组:\n", ndata2)
```

运行结果：

```
创建一维数组:
 [1 2 3]
创建二维数组:
 [[1 2 3]
  [4 5 6]]
```

### 3.2.2　其他方式创建数组

除了可以使用 array()函数创建 ndarray 对象外，还有其他创建数组的方式，具体分为以下几种。

**1. 创建指定维度和数据类型未初始化的数组**

在创建指定维度和数据类型未初始化的数组时，主要使用 empty()函数。通过 empty()函数创建一个新的数组，该数组只分配了内存空间，它里面填充的元素都是随机的，且数据类型默认为 float64。

**【例 3-3】** 利用 empty()函数创建未初始化数组。

```
import numpy as np
data = np.empty((3, 4))
print(data)
data.dtype ='int'
print(data)
```

运行结果：

```
[[6.23042070e-307 4.67296746e-307 1.69121096e-306 1.29061074e-306]
 [1.69119873e-306 1.78019082e-306 3.56043054e-307 7.56595733e-307]
 [1.60216183e-306 8.45596650e-307 9.79094970e-307 1.11261842e-306]]
[[4128860 6029375 3801155 5570652 6619251 7536754 4259932 7143524]
 [7209065 7536745 7471220 7602273 7471215 5242972 6488185 6357096]
 [7143538 7471184 6946927 6488165 7536756 6684764 7209061 6881400]]
```

由运行结果来看，empty()函数创建数组元素为随机值，因为它们未被初始化。如果要改变数组类型，可以使用 dtype 参数，如整型，dtype＝'int'。

**2. 创建指定维度(以 0 填充)的数组**

在创建指定维度(以 0 填充)的数组时，主要使用 zeros()函数。

**【例 3-4】** 利用 zeros()函数创建元素值为 0 的数组。

```
import numpy as np
data = np.zeros((3, 4))
print(data)
```

运行结果：

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

**3. 创建指定维度(以 1 填充)的数组**

在创建指定维度(以 1 填充)的数组时，主要使用 ones()函数。

【例 3-5】 利用 ones()函数创建元素值为 1 的数组。

```
import numpy as np
data = np.ones((3, 4))
print(data)
```

运行结果：

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

**4. 创建指定维度和类型的数组并以指定值填充**

在创建指定维度和类型的数组并以指定值填充时，主要使用 full()函数。

【例 3-6】 利用 full()函数创建 3 行 4 列数组，以 6 填充。

```
import numpy as np
data = np.full((3, 4), 6)
print(data)
```

运行结果：

```
[[6 6 6 6]
 [6 6 6 6]
 [6 6 6 6]]
```

**5. 创建数值范围函数**

通过 arange()函数可以创建一个指定区间均匀分布数值范围的数组，arange()函数同 Python 内置的 range()函数相似，区别在于返回值，arange()函数的返回值是数组，而 range()函数的返回值是列表。

arange()函数的语法格式如下。

```
arange([start,] stop[,step,], dtype=None]
```

参数说明如下。

- start：起始值，默认为 0。
- stop：终止值(不包含)。
- step：步长，默认为 1。
- dtype：创建数组的数据类型，如果不设置数据类型，则使用输入数据的数据类型。

【例 3-7】 利用 arange()函数创建数值范围数组。

```
import numpy as np
data = np.arange(1,10,2)
print(data)
```

运行结果：

```
[1 3 5 7 9]
```

从运行结果来看，arange()函数创建了一个步长为 2 的等差数组。

**6. 使用 linspace()函数创建等差数列**

等差数列是指如果一个数列从第二项起，每一项与它的前一项的差等于同一个常数，那么这个数列就叫作等差数列。

在 Python 中，创建等差数列可以使用 NumPy 的 linspace()函数，该函数用于创建一个一维的等差数列的数组，它与 arange()函数不同，arange()函数是从开始值到结束值的左闭右开区间(即包括开始值，但不包括结束值)，第三个参数(如果存在)是步长；而 linespace()函数是从开始值到结束值的闭区间(可以通过参数 endpoint＝False 设置，使结束值不是闭区间)，并且第三个参数是值的个数。

linspace()函数的语法格式如下。

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

参数说明如下。

- start：序列的起始值。
- stop：序列的终止值，如果 endpoint 参数的值为 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：如果值为 True，数列中包含 stop 参数的值；反之则不包含，默认为 True。
- retstep：如果值为 True，则生成的数组中会显示间距，反之则不显示。
- dtype：数组的数据类型。

**【例 3-8】** 利用 linspace()函数创建等差数列。

```
import numpy as np
n1 = np.linspace(100,200,5)
print(n1)
```

运行结果：

```
[100. 125. 150. 175. 200.]
```

**7. 使用 logspace()函数创建等比数列**

等比数列是指从第二项起，每一项与它的前一项的比值等于同一个常数的一种数列。在 Python 中，创建等比数列可以使用 NumPy 的 logspace()函数，语法格式如下。

```
numpy.logspace(start, stop, num=50, endpoint, base=10, dtype)
```

参数说明如下。

- start：代表间隔的起始值。
- stop：序列的终止值。如果 endpoint 参数值为 True，则该值包含于数列中。
- num：要生成的等步长的数据样本数量，默认为 50。
- endpoint：如果值为 True，则数列中包含 stop 参数值；反之则不包含，默认为 True。
- base：对数 log 的底数。
- dtype：数组的数据类型。

**【例 3-9】** 利用 logspace()函数创建等比数列。

```
import numpy as np
n1 = np.logspace(0,63,64,base=2,dtype='uint64')
print(n1)
```

运行结果：

```
[                  1                   2                   4
                   8                  16                  32
                  64                 128                 256
```

```
                512              1024              2048
...
]
```

### 3.2.3 利用随机数模块生成随机数组

随机数组的生成主要使用 NumPy 中的 random 模块。与 Python 的 random 模块相比，NumPy 的 random 模块功能更多，它增加了一些可以高效生成多种概率分布的样本值的函数。

**1. rand()函数随机生成 0～1 的数组**

rand()函数用于生成(0,1)的随机数组，传入一个值随机生成一维数组，传入一对值随机生成二维数组，语法格式如下。

```
numpy.random.rand(d0, d1, d2, d3,…,dn)
```

参数 d0，d1，d2，d3,…,dn 为整数，表示维度，可以为空。

**【例 3-10】** 利用 rand()函数随机生成 0～1 的数组。

```
import numpy as np
n1 = np.random.rand(5)
print("随机生成 0~1 的一维数组:\n",n1)
n2 = np.random.rand(2,5)
print("随机生成 0~1 的二维数组:\n",n2)
```

运行结果：

```
随机生成 0~1 的一维数组:
 [0.9735713  0.6523465  0.20245525  0.68185661  0.50856046]
随机生成 0~1 的二维数组:
 [[0.73896671  0.19294721  0.75125821  0.67867033  0.84908786]
  [0.13372963  0.77202048  0.44768409  0.08503824  0.97737551]]
```

上述代码中，rand()函数隶属于 numpy.random 模块，它的作用是随机生成 $N$ 维浮点数组。

需要注意的是，每次运行代码后生成的随机数组都不一样。

**2. randn()函数随机生成满足正态分布的数组**

randn()函数用于从正态分布中返回随机生成的数组，语法格式如下。

```
numpy.random.randn(d0, d1, d2, d3,…,dn)
```

参数 d0，d1，d2，d3,…,dn 为整数，表示维度，可以为空。

**【例 3-11】** 利用 randn()函数随机生成满足正态分布的数组。

```
import numpy as np
n1 = np.random.randn(5)
print("随机生成满足正态分布的一维数组:\n",n1)
n2 = np.random.randn(2,5)
print("随机生成满足正态分布的二维数组:\n",n2)
```

运行结果：

```
随机生成满足正态分布的一维数组:
 [ 0.07955534  -1.34039841  1.08138942  -1.03197295  0.97758157]
```

```
随机生成满足正态分布的二维数组:
 [[-0.70451333  -1.50902486  -1.27487169  -1.41709166  -1.84922965]
  [0.73333752  -0.74482002  0.30568106  0.27679098  1.40427717]]
```

**3. randint()函数生成一定范围内的随机数组**

randint()函数与 NumPy 中的 arange()函数类似。randint()函数用于生成一定范围内的随机数组,左闭右开区间,语法格式如下。

```
numpy.random. randint(low, high=None, size=None)
```

参数说明如下。

- low：低值(起始值),整数,且当参数 high 不为空时,参数 low 应小于参数 high,否则程序会出现错误。
- high：高值(终止值),整数。
- size：数组维数,整数或者元组,整数表示一维数组,元组表示多维数组。默认值为空,如果为空,则仅返回一个整数。

**【例 3-12】** 利用 randint()函数生成一定范围内的随机数组。

```
import numpy as np
n1 = np.random.randint(1,5,10)
print("随机生成 10 个 1~5 不包括 5 的一维数组:\n",n1)
n2 = np.random.randint(5,10)
print("size 参数为空,随机返回一个整数:\n",n2)
n3=np.random.randint(5,size=(2,5))
print("随机生成 5 以内的二维数组:\n",n3)
```

运行结果:

```
随机生成 10 个 1~5 不包括 5 的一维数组:
 [4 4 1 1 2 2 3 2 3 4]
size 参数为空,随机返回一个整数:
 7
随机生成 5 以内的二维数组:
 [[1 0 1 0 1]
  [1 1 1 4 4]]
```

**4. normal()函数生成正态分布的随机数组**

normal()函数用于生成正态分布的随机数组,语法格式如下。

```
numpy.random.normal(loc, scale, size)
```

参数说明如下。

- loc：正态分布的均值,对应正态分布的中心。'loc=0'说明是一个以 y 轴为对称轴的正态分布。
- scale：正态分布的标准差,对应正态分布的宽度,scale 值越大,正态分布的曲线越矮胖;scale 值越小,曲线越高瘦。
- size：表示数组维数。

**【例 3-13】** 利用 normal()函数生成正态分布的随机数组。

```
import numpy as np
n1 = np.random.normal(0,10,10)
```

```
print("生成正态分布的随机数组:\n",n1)
```

运行结果：

```
生成正态分布的随机数组:
 [13.43474667  -15.5728682  -10.10326793  -2.94301624  13.22621808
  7.95170908  15.77296567  -7.81340458   8.631446     8.91274957]
```

### 3.2.4 从已有的数组中创建数组

**1. asarray()函数创建数组**

asarray()函数用于创建数组，其与 array()函数类似，语法格式如下。

```
numpy.asarray(a, dtype=None, order=None)
```

参数说明如下。

- a：可以是列表、列表的元组、元组、元组的元组、元组的列表或多维数组。
- dtype：数组的数据类型。
- order：值为“C”和“F”，分别代表按行排列和按列排列，即数组元素在内存中的出现顺序。

**【例 3-14】** 利用 asarray()函数创建数组。

```
import numpy as np
n1 = np.asarray([1,4,7])
print("通过列表创建数组:\n",n1)
n2 = np.asarray([(1,4,7),(3,6,9)])
print("通过列表的元组创建数组:\n",n2)
n3 = np.asarray((1,4,7))
print("通过元组创建数组:\n",n3)
n4 = np.asarray(((1,4,7),(3,6,9)))
print("通过元组的元组创建数组:\n",n4)
```

运行结果：

```
通过列表创建数组:
 [1 4 7]
通过列表的元组创建数组:
 [[1 4 7]
  [3 6 9]]
通过元组创建数组:
 [1 4 7]
通过元组的元组创建数组:
 [[1 4 7]
  [3 6 9]]
```

**2. frombuffer()函数实现动态数组**

NumPy 中的 ndarray 数组对象不能像 Python 列表一样动态地改变其大小，因为在做数据采集时很不方便。通过 frombuffer()函数可以实现动态数组。frombuffer()函数接受 buffer 输入参数，以流的形式将读入的数据转换为数组。

frombuffer()函数的语法格式如下。

```
numpy .frombuffer(buffer, dtype=float, count=-1, offset=0)
```

参数说明如下。

- buffer：实现了_buffer_()方法的对象。
- dtype：数组的数据类型。
- count：读取的数据数量，默认为－1，表示读取段所有数据。
- offset：读取的起始位置，默认为 0。

**【例 3-15】** 利用 frombuffer()函数实现动态数组(字符串转换为数组)。

```
import numpy as np
l = b'Ilove Python'
print(type(l))
ar = np.frombuffer(l, dtype = "S1")
print(ar)
print(type(ar))
```

运行结果：

```
<class 'bytes'>
[b'I' b'l' b'o' b'v' b'e' b' ' b'P' b'y' b't' b'h' b'o' b'n']
<class 'numpy.ndarray'>
```

上述代码中，buffer 参数值为字符串时，Python 3 版本默认字符串是 Unicode 类型，所以要转换成 Byte string 类型，需要在原字符串前加上 b。

**3. fromiter()函数从可迭代对象中建立数组对象**

fromiter()函数用于从可迭代对象中建立数组对象。

语法格式如下。

```
numpy.fromiter(iterable, dtype, count=-1)
```

参数说明如下。

- iterable：可迭代对象。
- type：数组的数据类型。
- count：读取的数据数量，默认为－1，表示读取所有数。

**【例 3-16】** 利用 fromiter()函数从可迭代对象中建立数组对象。

```
import numpy as np
iterable = (x * 2 for x in range(6))
ar = np.fromiter(iterable,dtype='int')
print(ar)
```

运行结果：

```
[ 0  2  4  6  8  10]
```

**4. empty_like()函数创建未初始化的数组**

empty_like()函数用于创建一个与给定数组具有相同维度和数据类型且未初始化的数组，语法格式如下。

```
numpy.empty_like(prototype, dtype=None,order='K', subok=True)
```

参数说明如下。

- prototype：给定的数组。
- type：覆盖结果的数据类型。

- order：指定数组的内存布局，可取值 C(按行)、F(按列)、A(原顺序)、K(数据元素在内存中的出现顺序)。
- subok：默认情况下，返回的数组被强制为基类数组。如果值为 True，则返回子类。

【例 3-17】 利用 empty_like()函数创建未初始化的数组。

```
import numpy as np
ar = np.empty_like([[1,4,7],[2,5,8]])
print(ar)
```

运行结果：

```
[[376700994  26217067     90226]
 [    65537  40763481  24903768]]
```

**5. zeros_like()函数创建以 0 填充的数组**

zeros_like()函数用于创建一个与给定数组维度和数据类型相同，并以 0 填充的数组，语法格式如下。

```
numpy.zeros_like(a):
```

参数说明如下。

a 是一个 ndarray，即产生一个维度和 a 一样大小的全 0 数组。

【例 3-18】 利用 zeros_like()函数创建以 0 填充的数组。

```
import numpy as np
ar = np.zeros_like([[1,4,7],[2,5,8]])
print(ar)
```

运行结果：

```
[[0 0 0]
 [0 0 0]]
```

**6. ones_like()函数创建以 1 填充的数组**

ones_like()函数用于创建一个与给定数组维度和数据类型相同，并以 1 填充的数组，语法格式如下。

```
numpy.ones_like(a):
```

参数说明如下。

a 是一个 ndarray，即产生一个维度和 a 一样大小的全 1 数组。

【例 3-19】 利用 ones_like()函数创建以 1 填充的数组。

```
import numpy as np
ar = np.ones_like([[1,4,7],[2,5,8]])
print(ar)
```

运行结果：

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

**7. full_like()函数创建以指定值“6”填充的数组**

full_like()函数用于创建一个与给定数组维度和数据类型相同，并以指定值填充的数组，语法格式如下。

```
numpy.full_like(a, fill value, dtype=None, order='K', subok=True)
```

参数说明如下。

- a：给定的数组。
- fill value：填充值。
- dtype：数组的数据类型，默认为 None，则使用给定数组的数据类型。
- order：指定数组的内存布局，可取值 C(按行)、F(按列)、A(原顺序)、K(数组元素在内存中的出现顺序)。
- subok：默认情况下，返回的数组被强制为基类数组。如果值为 True，则返回子类。

**【例 3-20】** 利用 full_like()函数创建以指定值“6”填充的数组。

```
import numpy as np
ar = np.arange(5)
print('生成 5 个元素的数组:\n',ar)
n = np.full_like(ar,6)
print('用 6 填充后的同型数组:\n',n)
```

运行结果：

```
生成 5 个元素的数组:
 [0 1 2 3 4]
用 6 填充后的同型数组:
 [6 6 6 6 6]
```

## 3.3　数组对象的数据类型

在对数组进行基本操作前，首先了解一下 NumPy 的数据类型。NumPy 的数据类型比 Python 数据类型增加了更多种类的数值类型，如表 3-2 所示，为了区别于 Python 的数据类型，像 bool、int、float、complex 等数据类型的名称末尾加了短下画线“_”。

### 3.3.1　查看数据类型

如前面所述，通过 ndarray.dtype 可以创建一个表示数据类型的对象。要想获取数据类型的名称，则需要访问 name 属性进行获取。

**【例 3-21】** 获取数据类型。

```
import numpy as np
data_one = np.array([[1, 4, 7], [2, 5, 8]])
print(data_one.dtype.name)
```

运行结果：

```
'int32'
```

**注意**：在默认情况下，64 位 Windows 系统输出的结果为 int32，64 位 Linux 或 macOS 系统输出的结果为 int64，当然也可以通过 dtype 来指定数据类型的长度。

上述代码中，使用 dtype 属性查看 data_one 对象的类型，输出结果是 int32。从数据类型的命名方式上可以看出，NumPy 的数据类型是由一个类型名(如 int. float)和元素位长的数字组成。如果在创建数组时，没有显式地指明数据的类型，则可以根据列表或元组中的元

素类型推导出来。默认情况下，通过 zeros()、ones()、empty()函数创建的数组中数据类型为 float64。

表 3-2 罗列了 NumPy 中常用的数据类型。

**表 3-2 NumPy 中常用的数据类型**

| 数 据 类 型 | 含 义 |
|---|---|
| bool | 布尔类型，值为 Tue 或 False |
| int8、uint8 | 有符号和无符号的 8 位整数 |
| int16、uint16 | 有符号和无符号的 16 位整数 |
| int32、uint32 | 有符号和无符号的 32 位整数 |
| int64、uint64 | 有符号和无符号的 64 位整数 |
| float16 | 半精度浮点数(16 位) |
| float32 | 半精度浮点数(32 位) |
| float64 | 半精度浮点数(64 位) |
| complex64 | 复数，分别用两个 32 位浮点数表示实部和虚部 |
| complex128 | 复数，分别用两个 64 位浮点数表示实部和虚部 |
| object | Python 对象 |
| string_ | 固定长度的字符串类型 |
| unicode | 固定长度的 Unicode 类型 |

每一个 NumPy 内置的数据类型都有一个特征码，它能唯一标识一种数据类型，具体如表 3-3 所示。

**表 3-3 NumPy 内置特征码**

| 特 征 码 | 含 义 | 特 征 码 | 含 义 |
|---|---|---|---|
| b | 布尔型 | i | 有符号整型 |
| u | 无符号整型 | f | 浮点型 |
| c | 复数类型 | O | Python 对象 |
| S，a | 字节字符串 | U | Unicode 字符串 |
| V | 原始数据 | | |

## 3.3.2 转换数据类型

**1. 借助类型函数转换数据类型**

NumPy 的数据类型比 Python 数据类型增加了更多种类的数值类型。每一种数据类型都有相应的数据转换函数。在创建 ndarray 数组时，也可以直接指定数值类型。但是复数不能转换成为整数类型或者浮点数。

**【例 3-22】** 利用类型函数和创建 NumPy 数据时，进行数据类型转换。

```
import numpy as np
print(np.int8(3.69))
print(np.float32(6))
print(float(True))
```

```
ar = np.arange(5,dtype=float)
print(ar)
```

运行结果：

```
3
6.0
1.0
[0. 1. 2. 3. 4.]
```

**2. 借助 astype()函数转换数据类型**

ndarray 对象的数据类型可以通过 astype()函数进行转换。

**【例 3-23】** 利用 astype()函数转换数据类型。

```
import numpy as np
data = np.array([[1, 4, 7], [2, 5, 8]])
print(data.dtype)
float_data = data.astype(np.float64)          #数据类型转换为 float64
print(float_data.dtype)
float_data_new = np.array([1.2, 2.3, 3.5])
print(float_data_new)
str_data = np.array(['1', '2', '3'])
int_data = str_data.astype(np.int64)
print(int_data)
```

运行结果：

```
int32
float64
[1.2 2.3 3.5]
[1 2 3]
```

上述示例中，将数据类型 int64 转换为 float64，即整型转换为浮点型。若希望将数据的类型由浮点型转换为整型，则需要将小数点后面的部分截掉。

如果数组中的元素是字符串类型的，且字符串中的每个字符都是数字，则也可以使用 astype()方法将字符串转换为数值类型。

## 3.4　数 组 运 算

无论是形状相同的数组，还是形状不同的数组，它们之间都可以执行算术运算。与 Python 列表不同，数组在参与算术运算时无须遍历每个元素，便可以对每个元素执行批量运算，效率更高。

NumPy 数组不需要循环遍历，即可对每个元素执行批量的算术运算操作，这个过程叫做向量化运算。不过，如果两个数组的大小(ndarray.shape)不同，则它们进行算术运算时会出现广播机制。除此之外，数组还支持使用算术运算符与标量进行运算，本节将针对数组运算的内容进行详细的介绍。

### 3.4.1　形状相同的数组间运算

在 NumPy 中，大小相等的数组之间的任何算术运算都会应用到元素级，即只用于位置

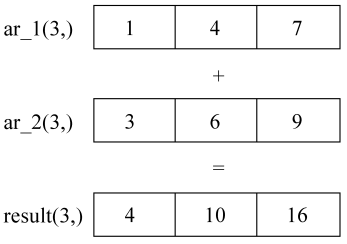


**图 3-2　形状相同的数组运算**

相同的元素之间，所得的运算结果组成一个新的数组。接下来，通过一张示意图来描述什么是向量化运算，具体如图 3-2 所示。

由图 3-2 可知，数组 ar_1 与 ar_2 对齐以后，会让相同位置的元素相加得到一个新的数组 result。其中，result 数组中的每个元素为操作数相加的结果，并且结果的位置跟操作数的位置是相同的。

**【例 3-24】** 大小相等的数组之间的算术运算。

```
import numpy as np
data1 = np.array([[3, 6, 9], [2, 5, 6]])
data2 = np.array([[1, 2, 3], [1, 2, 3]])
print('两个数组相加的结果:\n',data1+data2)
print('两个数组相乘的结果:\n',data1 * data2)
print('两个数组相减的结果:\n',data1-data2)
print('两个数组相除的结果:\n',data1/data2)
print('两个数组幂运算的结果:\n',data1**data2)
print('两个数组比较运算的结果:\n',data1>data2)
```

运行结果：

```
两个数组相加的结果:
 [[4  8  12]
  [3  7  9]]
两个数组相乘的结果:
 [[3  12  27]
  [2  10  18]]
两个数组相减的结果:
 [[2  4  6]
  [1  3  3]]
两个数组相除的结果:
 [[3.  3.  3.]
  [3.  2.5  2.]]
两个数组幂运算的结果:
 [[3  36  729]
  [2  25  216]]
两个数组比较运算的结果:
 [[True  True  True]
  [True  True  True]]
```

从数组比较运算的结果来看，组的比较运算是数组中对应位置元素的比较运算，比较后的结果是布尔值数组。

### 3.4.2　形状不同的数组间运算

当形状不相等的数组执行算术运算的时候，就会出现广播机制，该机制会对数组进行扩展，使数组的 shape 属性值一样，这样就可以进行向量化运算了。

所谓广播是指不同形状的数组之间执行算术运算的方式。

广播机制需要遵循以下 4 个原则。

(1) 让所有输入数组都向其中 shape 最长的数组看齐，shape 中不足的部分都通过在前面加 1 补齐。

(2) 输出数组的 shape 是输入数组 shape 的各个轴上的最大值。

(3) 如果输入数组的某个轴和输出数组的对应轴的长度相同或者其长度为 1 时,这个数组能够用来计算,否则出错。

(4) 当输入数组的某个轴的长度为 1 时,沿着此轴运算时都用此轴上的第一组值。

**【例 3-25】** 形状不相等的数组向量化运算。

```
import numpy as np
arr1 = np.array([[0], [1], [2], [3]])
print('数组 arr1 的形状:\n',arr1.shape)
arr2 = np.array([1, 2, 3])
print('数组 arr2 的形状:\n',arr2.shape)
print('arr1 与 arr2 相加的结果为:\n',arr1 + arr2)
```

运行结果:

```
数组 arr1 的形状:
 (4, 1)
数组 arr2 的形状:
 (3,)
arr1 与 arr2 相加的结果为:
 [[1 2 3]
  [2 3 4]
  [3 4 5]
  [4 5 6]]
```

上述代码中,数组 arr1 的 shape 是(4,1),arr2 的 shape 是(3,),这两个数组要是进行相加,按照广播机制会对数组 arrl 和 arr2 都进行扩展,使得数组 arrl 和 arr2 的 shape 都变成(4,3)。

下面通过一张图来描述广播机制扩展数组的过程,具体如图 3-3 所示。

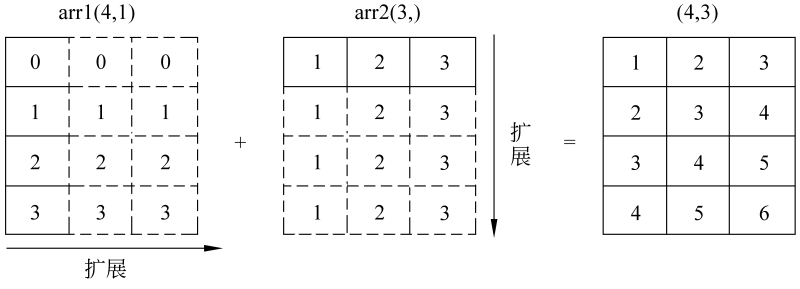


**图 3-3　数组广播机制**

广播机制实现了对两个或两个以上数组的运算,即使这些数组的 shape 不是完全相同的,只需要满足如下任意一个条件即可。

(1) 数组的某一维度等长。

(2) 其中一个数组的某一维度为 1。

广播机制需要扩展维度小的数组,使得它与维度最大的数组的 shape 值相同,以便使用元素级函数或者运算符进行运算。

### 3.4.3　数组与标量间的运算

标量其实就是一个单独的数;而向量是一组数,这组数是顺序排列的,这里我们理解为数组。那么,数组的标量运算也可以理解为向量与标量之间的运算。

大小相等的数组之间的任何算术运算都会将运算应用到元素级，同样，数组与标量的算术运算也会将那个标量值传播到各个元素。当数组进行相加、相减、乘以或除以一个数字时，这些称为标量运算，如图 3-4 所示。标量运算会产生一个与数组具有相同数量的行和列的新矩阵，其原始矩阵的每个元素都被相加、相减、相乘或者相除。

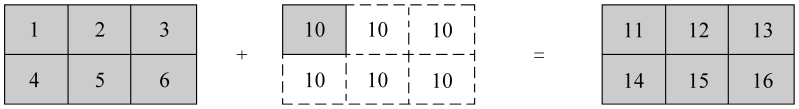


**图 3-4　数组和标量之间的运算示意图**

**【例 3-26】** 数组和标量之间的运算。

```
import numpy as np
data1 = np.array([[1, 4, 7], [2, 5, 8]])
data2 = 10
print('数组与 10 相加的结果:\n',data1+data2)
print('数组与 10 相乘的结果:\n',data1 * data2)
print('数组与 10 相减的结果:\n',data1-data2)
print('数组与 10 相除的结果:\n',data1/data2)
```

运行结果：

```
数组与 10 相加的结果:
 [[11 14 17]
  [12 15 18]]
数组与 10 相乘的结果:
 [[10 40 70]
  [20 50 80]]
数组与 10 相减的结果:
 [[-9 -6 -3]
  [-8 -5 -2]]
数组与 10 相除的结果:
 [[0.1 0.4 0.7]
  [0.2 0.5 0.8]]
```

## 3.5　数组元素的操作

NumPy 数组元素是通过数组的索引和切片来访问和修改的，因此索引和切片是 NumPy 中最重要、最常用的操作。NumPy 中提供了多种形式的索引：整数索引、花式索引和布尔索引，通过这些索引可以访问数组的单个、多个或一行元素。

数组元素也可以进行增加、删除、修改和查询。

### 3.5.1　整数索引和切片的基本使用

数组的索引，即用于标记数组当中对应元素的唯一数字，从 0 开始，即数组中的第一个元素的索引是 0，以此类推。NumPy 数组可以使用标准 Python 语法 x[obj]对数组进行索引，其中，x 是数组，obj 是索引。

NumPy 中可以使用整数索引访问数组，以获取该数组中的单个元素或一行元素。

因此，ndarray 对象的元素可以通过索引和切片来访问和修改，就像 Python 内置的容

器对象一样。

数组的切片可以理解为对数组的分割，按照等分或者不等分，将一个数组切割为多个片段，它与 Python 中列表的切片操作一样。数组的切片返回的是原始数组的视图，并会产生新的数据，即在视图上的操作会使原数组发生改变。如果需要的并非视图而是要复制数据，则可以通过 copy()方法实现。

NumPy 中的切片用冒号分隔切片，参数用来进行切片操作，语法格式如下。

```
[start : stop: step]
```

参数说明如下。

- start：起始索引。
- stop：终止索引。
- step：步长。

**【例 3-27】** 使用索引和切片的方式获取一维数组元素。

```
import numpy as np
ar = np.arange(6)                                    #创建一个一维数组
print('输出全部数组元素:\n',ar)
print('获取索引为 5 的元素:\n',ar[5])
print('获取索引为 3~5 的元素,但不包括 5:\n',ar[3:5] )
print('获取索引为 1~6 的元素,步长为 2:\n',ar[1:6:2])
```

运行结果：

```
输出全部数组元素:
 [0 1 2 3 4 5]
获取索引为 5 的元素:
 5
获取索引为 3~5 的元素,但不包括 5:
 [3 4]
获取索引为 1~6 的元素,步长为 2:
 [1 3 5]
```

切片式索引操作需要注意以下几点。

(1) 索引是左闭右开区间，如上述代码中的 ar[3：5]，只能获取到索引从 3 到 5 的元素，而获取不到索引为 5 的元素。

(2) 当没有 start 参数时，代表从索引 0 开始取数。

(3) start、stop 和 step 3 个参数都可以是负数，代表反向索引。

对于多维数组来说，索引和切片的使用方式与列表就大不一样了，它的每一个维度都有一个索引，各个维度的索引之间用逗号隔开。在二维数组中，每个索引位置上的元素不再是一个标量，而是一个一维数组。

**【例 3-28】** 使用索引和切片的方式获取二维数组元素。

```
import numpy as np
arr2d = np.array([[1, 4, 7],[2, 5, 8],[3, 6, 9]])          #创建二维数组
print('输出全部二维数组元素:\n',arr2d)
print('获取索引为 1 的元素:\n',arr2d[1])
print('获取位于第 0 行第 1 列的元素:\n',arr2d[0,1] )
print('传入一个切片获取数组元素:\n',arr2d[:2])
print('传入两个切片获取数组元素:\n',arr2d[0:2, 0:2])
```

```
print('切片与整数索引混合使用获取数组元素:\n',arr2d[1, :2])
```

运行结果：

```
输出全部二维数组元素:
 [[1 4 7]
  [2 5 8]
  [3 6 9]]
获取索引为 1 的元素:
 [2 5 8]
获取位于第 0 行第 1 列的元素:
 4
传入一个切片获取数组元素:
 [[1 4 7]
  [2 5 8]]
传入两个切片获取数组元素:
 [[1 4]
  [2 5]]
切片与整数索引混合使用获取数组元素:
 [2 5]
```

从上述运行结果来看，如果想通过索引的方式来获取二维数组的单个元素，就需要通过形如“arr[x,y]”以逗号分隔的索引来实现。其中，x 表示行号，y 表示列号。例如，上述例子中的 arr2d[0,1]。

如果想获取数组中的单个元素，必须同时指定这个元素的行索引和列索引。例如，获取索引位置为第 1 行第 1 列的元素，可以通过 arr2d[1,1]来实现。

相比一维数组，多维数组的切片方式花样更多，多维数组的切片是沿着行或列的方向选取元素的，可以传入一个切片，也可以传入多个切片，还可以将切片与整数索引混合使用。上述多维数组切片操作的相关示意图，如图 3-5 所示。

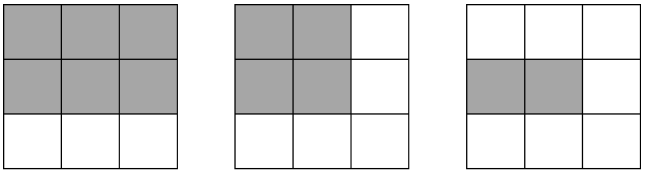


**图 3-5　多维数组切片图示**

## 3.5.2　花式(数组)索引的基本使用

花式索引是 NumPy 的一个术语，是指将整数数组或列表作为索引，然后根据索引数组或索引列表的每个元素作为目标数组的下标再进行取值。

当使用花式索引访问一维数组时，会将花式索引对应的数组或列表的元素作为索引，依次根据各个索引获取对应位置的元素，并将这些元素以数组的形式进行返回；当使用花式索引访问二维数组时，会将花式索引对应的数组或列表的元素作为索引，依次根据各个索引获取对应位置的一行元素，并将这些行元素以数组的形式进行返回。

**【例 3-29】**　创建一个 4 行 4 列的二维数组，按要求获取元素。

```
import numpy as np
```

```
demo_arr = np.empty((4, 4))                    #创建一个空数组
for i in range(4):
    demo_arr[i] = np.arange(i, i + 4)          #动态地为数组添加元素
print("输出全部数组元素:\n",demo_arr)
print("获取索引为[0,2]的元素:\n",demo_arr[[0, 2]])
print("获取索引为(1,2)和(2,3)的元素:\n",demo_arr [[1, 2], [1, 3]])
```

运行结果：

```
输出全部数组元素:
 [[0. 1. 2. 3.]
  [1. 2. 3. 4.]
  [2. 3. 4. 5.]
  [3. 4. 5. 6.]]
获取索引为[0,2]的元素:
 [[0. 1. 2. 3.]
  [2. 3. 4. 5.]]
获取索引为(1,1)和(2,3)的元素:
 [2. 5.]
```

在上述运行结果中，将(0,2)作为索引，分别获取 demo_arr 中索引 0 对应的一行数据以及索引 2 对应的一行数据。

如果使用两个花式索引操作数组时，即两个列表或数组，则会将第 1 个作为行索引，第 2 个作为列索引，通过二维数组索引的方式，选取其对应位置的元素，如 demo_arr [[1, 1], [2, 3]]。

### 3.5.3　布尔型索引的基本使用

布尔索引指以布尔值组成的数组或列表为索引。当使用布尔索引访问数组时，会将布尔索引对应的数组或列表的元素作为索引，以获取索引为 True 时对应位置的元素，即返回的数据是布尔数组中 True 对应位置的值。

**【例 3-30】** 假设现在有一组存储了学生姓名的数组，以及一组存储了学生各科成绩的数组，存储学生成绩的数组中，每一行成绩对应的是一个学生的成绩。如果想筛选某个学生对应的成绩，可以通过比较运算符，先产生一个布尔型数组，然后利用布尔型数组作为索引，返回布尔值 True 对应位置的数据。

```
import numpy as np
#存储学生姓名的数组
student_name = np.array(['申凡', '石英', '史伯', '王骏'])
#存储学生成绩的数组
student_score = np.array([[75, 98, 56], [79, 86, 78], [76, 89, 90], [84, 87, 76]])
#对 student_name 和字符串"王骏"通过运算符产生一个布尔型数组
student_name == '王骏'
#将布尔数组作为索引应用于存储成绩的数组 student_score,
#返回的数据是 True 值对应的行
print(student_score[student_name=='王骏'])
print(student_score[student_name=='王骏', :1])
```

运行结果：

```
[[84 87 76]]
[[84]]
```