5.3.2 reduce()通用函数的聚合操作

任务：`reduce()`通用函数的聚合操作

> 知识点：
>
> - numpy二元通用函数提供了非常有用的聚合 `reduce()` 方法，通过聚合可以实现对数组元素的累积操作。
> - `reduce()` 聚合方法：对给定数组元素**累积**执行二元通用函数操作。即，将上次运算的结果和下个数组元素进行累积运算，直到遍历完所有元素。

例如，对`add`通用函数调用`reduce`方法会返回数组中所有元素的和。

```
x = np.arange(1, 6)
print(np.add.reduce(x))
```

例如，对`multiply`通用函数调用`reduce`方法会返回数组中所有元素的乘积。

```
x = np.arange(1, 6)
print(np.multiply.reduce(x))
```

> 知识点：如果需要存储每次计算的中间结果，可以使用 `accumulate()` 方法。

```
x = np.arange(1, 6)
print(np.add.accumulate(x))
print(np.multiply.accumulate(x))
```

习题：

- 随机一个自定义`size`的二维数组，选定4种通用函数，然后使用`reduce()`和`accumulate()`对数组实现聚合运算，并输出结果。

  代码：

```python
x2 = np.random.randint(10, size=(3, 3))
print(x2)

print(f'\n{np.add.reduce(x2)}')
print(f'\n{np.subtract.reduce(x2)}')
print(f'\n{np.power.reduce(x2)}')
print(f'\n{np.multiply.reduce(x2)}')

print(f'\n{np.add.accumulate(x2)}')
print(f'\n{np.subtract.accumulate(x2)}')
print(f'\n{np.power.accumulate(x2)}')
print(f'\n{np.multiply.accumulate(x2)}')
```

输出结果：

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

[15 14  9]

[-7  0  1]

[        0 956385313       625]

[120  84  20]

[[ 4  7  5]
 [ 9 11  7]
 [15 14  9]]

[[ 4  7  5]
 [-1  3  3]
 [-7  0  1]]

[[        4         7         5]
 [     1024      2401        25]
 [        0 956385313       625]]

[[  4   7   5]
 [ 20  28  10]
 [120  84  20]]
```

5.3.3 outer()通用函数的外积操作

任务：`outer`通用函数的外积操作

> 知识点：
>
> - 任何通用函数都可使用 `outer` 外积方法获得两个输入数组**所有元素对的函数运算结果**。
> - 联系线性代数的叉乘外积(**展成元素对**)，
> - `outer` 方法：首先将两个数组展成元素对形式，然后再对这些元素对实现通用函数运算。

```
x = np.arange(1, 6)
print(np.add.outer(x, x))
```

习题：

- 使用外积方式，实现99乘法表，并输出结果。

  代码：

  ```
  x = np.arange(1, 10)
  print(np.multiply.outer(x, x))
  ```

  输出结果：

  ```
  [[ 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]]
  ```

第六章 Numpy计算科学模块(五)

6.1 Numpy数组的聚合统计分析

当面对大量的数据时，首先需要计算数据的**相关统计值**。

常见统计值有均值和标准差，通过这两个值可快速了解数据的基本特征，但是其他一些形式的聚合统计方式：求和、乘积、中位数、最小值和最大值、分位数等等。

任务：认识numpy数组的聚合

> 知识点：经常会在numpy中听到**聚合**术语，那么何谓聚合?
>
> - **聚合**：指对数组进行某种操作或运算(求和、极值等)，得到了一个**对原数组信息概括的结果数组**。而结果数组的维度 `ndim` 被压缩了。换言之，**原数组的维度 `ndim` 大于结果数组的维度 `ndim` 。**
> - 聚合作用：结果数组**按某种操作聚合(提炼)了**原数组的信息；结果数组概括了原数组某些方面的统计信息，比如求和、均值、方差等。
> - 原数组：numpy聚合函数的输入数组。
> - 结果数组：numpy聚合函数的返回数组。

习题:

- 了解术语：原数组、结果数组、聚合的概念。

6.1.1 数组聚合之求和

任务：`np.sum()` 数组的求和

为实现数组的求和，numpy提供了比python内置 `sum()` 函数功能更强大的求和方法 `np.sum()` 。请耐心理解 `np.sum()` 的细节功能。

> 知识点： `numpy.sum(a, axis=None, dtype=None, out=None)`
>
> - 描述：按某种规则，求和数组元素。

- 参数：numpy数组 `a` ，包括一维数组和多维数组。
- 参数： `axis` 指定沿第 `axis` 轴对数组进行求和。默认 `axis=None` ，此时对数组所有元素进行求和。(回顾 `axis` **参数的解析**)
- 参数： `dtype` 指定返回数据类型，默认不指定类型。
- 参数： `out` 指定预先分配的用于保存结果的空间，默认不指定空间。(回顾**通用函数的指定输出**)
- 返回：返回求和结果。标量或数组。
-
  - 如果不指定 `axis` 参数，返回所有数组元素之和，此时为一个标量。
  - 如果指定 `axis` 参数，返回**比原数组低一维的结果数组**。比如三维数组，那么结果结果是二维数组。
- 注意： `np.sum()` 通过设置不同的参数，其功能涵盖了python内置 `sum()` 函数和 `np.add.reduce()` 。

任务：一维数组的求和

下面例子分别调用python和numpy的求和函数对数组进行求和，并给出时间比较。

```
import numpy as np
# 对小数组进行运算
x = np.random.rand(100)
print(sum(x))
print(np.sum(x))
```

结果可知， `np.sum()` 比python的 `sum()` 快很多。

```
# 对大数组进行运算
big_array = np.random.rand(1000000)
%time sum(big_array)
%time np.sum(big_array)
```

任务：进阶 `np.sum()` 之 `axis` 参数分析

知识点： `axis` 参数的深入。

- 为 `np.sum()` 指定 `axis` 参数后，返回的**结果数组 `shape` 为原数组的非 `axis` 轴 `shape`** 。
- 例如，三维数组 `x` 的 `shape=(4,8,10)` 。如果沿 `axis=0` ，那么结果数组 `shape=(8,10)` ；如果沿 `axis=2` ，那么结果数组 `shape=(4,8)` 。
- 由此可知， `np.sum()` 的**结果数组是比原数组低一维的数组**。比如三维数组，那么结果数组是二维数组。

```python
x = np.random.randint(10, size=(4,8,10))
print('x', x.shape, x.ndim)

# 使用python的sum()
py_sum = sum(x)
print('py_sum', py_sum.shape, py_sum.ndim)

# 沿0轴对数组求和，等价于使用python的sum()
sum1 = np.sum(x, axis=0)
print('axis=0', sum1.shape, sum1.ndim)

# 沿1轴对数组求和
sum2 = np.sum(x, axis=1)
print('axis=1', sum2.shape, sum2.ndim)
```

任务：高维数组的求和

> 知识点：
>
> - python内置 `sum(x)` 函数等价于 `np.sum(x, axis=0)` 。
> - `np.add.reduce()` 通用函数的聚合等价于 `np.sum(x, axis=0)` 。

```python
x = np.random.randint(10, size=(3,4))
print(x)

print('sum(x)', sum(x))
print('np.add.reduce()', np.add.reduce(x))
print('np.sum(x)', np.sum(x))
# 对二维数组，axis=0，沿列求和
print('np.sum(x, axis=0)', np.sum(x, axis=0))
# 对二维数组，axis=1，沿行求和
print('np.sum(x, axis=1)', np.sum(x, axis=1))
```

习题：

- 1.(概念题) 结合原数组和结果数组的 `shape` ，分析 `np.sum()` 的 `axis` 参数的含义。

- 2.随机生成一个 `size=(4,7)` 的二维整形数组。
-
    - 使用 `np.sum()` ，设置不同的 `axis` 参数，对该数组进行求和聚合，输出结果数组的 `shape` 、 `ndim` 等信息。
    - 使用 `sum()` 、 `np.add.reduce()` 函数，输出结果数组，然后对比 `np.sum()` 的结果。

    代码：

```
x2 = np.random.randint(10, size=(4,7))

py_sum = sum(x)
print('py_sum', py_sum.shape, py_sum.ndim,"\n")

sum1 = np.sum(x, axis=0)
print('axis=0', sum1.shape, sum1.ndim,"\n")

sum2 = np.sum(x, axis=1)
print('axis=1', sum2.shape, sum2.ndim,"\n")

print('np.add.reduce()', np.add.reduce(x2))
print('np.sum(x2)', np.sum(x2))
```

    输出结果：

```
py_sum (8, 10) 2

axis=0 (8, 10) 2

axis=1 (4, 10) 2

np.add.reduce() [26 15 21 20 12 15 22]
np.sum(x2) 131
```

- 3.随机生成一个 `size=(4,7,8,10)` 的四维整形数组。使用 `np.sum()` 对该数组进行求和聚合，通过设置 `axis` 参数，总共有多少种聚合的结果，列举每种 `axis` 对应的结果数组的 `shape` 和 `ndim` 。并用程序验证结论。

    代码：

```
x4 = np.random.randint(10, size=(4,7,8,10))

sum1 = np.sum(x, axis=0)
print('axis=0', sum1.shape, sum1.ndim,"\n")

sum2 = np.sum(x, axis=1)
print('axis=1', sum2.shape, sum2.ndim,"\n")

sum3 = np.sum(x, axis=2)
print('axis=2', sum3.shape, sum3.ndim,"\n")
```

输出结果：

```
axis=0 (8, 10) 2

axis=1 (4, 10) 2

axis=2 (4, 8) 2
```

6.1.2 数组聚合之最小值和最大值

任务：`np.min()`和`np.max()`数组的最小值和最大值

知识点： `np.min()` 和 `np.max()` 分别获取numpy数组的最小值和最大值。由于用法与 `np.sum()` 类似，这里不具体展开。

- 与 `np.sum()` 类似， `np.min()` 和 `np.max()` 也可以通过参数 `axis` 指明对哪个 `axis` 轴元素进行聚合。
- 最小值方法语法： `numpy.min(a, axis=None, out=None)` 。
- 最大值方法语法： `numpy.max(a, axis=None, out=None)` 。
- python也提供了内置 `min()` 和 `max()` 方法，但是该功能**仅限于一维数组，不能用于高维数组**。
- 新numpy版本中， `np.min()` 的别名方法 `np.amin()` ； `np.max()` 的别名方法 `np.amax()` 。功能等价。

```
# 对大数组进行运算
big_array = np.random.rand(1000000)
%time min(big_array)
%time np.min(big_array)
```

```python
x = np.random.randint(10, size=(3,4))
print(x)

#python内置的min不能用于高维数组
#print('min(x)', min(x))

print('np.min(x)', np.min(x))
# 对二维数组，axis=0，沿列求最小值
print('np.min(x, axis=0)', np.min(x, axis=0))
# 对二维数组，axis=1，沿行求最小值
print('np.min(x, axis=1)', np.min(x, axis=1))
```

习题：

- 随机生成一个 `size=(4,7,2)` 的三维整形数组。
-
    - 使用 `np.min()` 和 `np.max()` ，设置不同的 `axis` 参数，对该数组进行聚合，输出结果数组的 `shape` 、 `ndim` 等信息。
    - 尝试着使用python内置函数，输出结果数组。
- 代码：
-
```python
x = np.random.randint(10, size=(4,7,2))

min1 = np.min(x, axis=0)
print('np.min1(x, axis=0)',min1.shape, min1.ndim, "\n",)
max1 = np.max(x, axis=0)
print('np.max1(x, axis=0)',max1.shape, max1.ndim, "\n")

min2 = np.min(x, axis=1)
print('np.min2(x, axis=1)',min2.shape, min2.ndim, "\n",)
max2 = np.max(x, axis=1)
print('np.max2(x, axis=1)',max2.shape, max2.ndim, "\n",)
```

- 输出结果：
-
```
np.min1(x, axis=0) (7, 2) 2

np.max1(x, axis=0) (7, 2) 2

np.min2(x, axis=1) (4, 2) 2

np.max2(x, axis=1) (4, 2) 2
```

6.1.3 通过面向对象方式聚合数组

任务：通过对象方法的方式对数组进行聚合

> 知识点：
>
> - `np.min()` 、 `np.max()` 、 `np.sum()` 在数组对象 `x` 中，可以通过更简洁的**面向对象方法**被调用。
> - 调用方式： `x.min()` 、 `x.max()` 、 `x.sum()` 。其中 `x` 为数组对象。
> - `np.min()` 第一个参数为数组。然而，在 `x.min()` 中，通过对象 `x` 调用 `min()` 方法， `x` 被隐式传递给了 `min()` 方法。因此，不再需要额外传递数组参数。
> - 通过对象来调用聚合方法，可实现数组的求和、最小值和最大值等功能。
> - `axis` 参数功能类似，可以实现多维数组的聚合功能。

> 知识点：可以通过 `axis` 参数，指明聚合函数是针对数组的哪个 `axis` 轴进行操作的。

> 知识点：如果缺省 `axis` 参数，那么聚合函数将会返回对整个数组的聚合结果。

```
x = np.random.randint(10, size=(3,4))
print(x)

# 缺省`axis`参数，返回整个数组元素的求和
print('x.sum()', x.sum())

# 对二维数组，axis=0，沿列求和
print('x.sum(axis=0)', x.sum(axis=0))

# 对二维数组，axis=1，沿行求和
print('x.sum(axis=1)', x.sum(axis=1))
```

习题：

- 随机生成一个 `size=(4,7,4)` 的三维整形数组，使用 `x.min()` 、 `x.max()` 对原数组进行聚合，分析不同 `axis` 参数对结果的影响，并输出结果数组的 `shape` 和 `ndim` 。

  代码：

  ```
  x = np.random.randint(10, size=(4,7,4))
  ```

```
2
3  min1 = x.min(axis=0)
4  print('x.min1(axis=0)',min1.shape, min1.ndim, "\n",)
5  max1 = x.max(axis=0)
6  print('x.max1(axis=0)',max1.shape, max1.ndim, "\n")
7
8  min2 = x.min(axis=1)
9  print('x.min2(axis=1)',min2.shape, min2.ndim, "\n",)
10 max2 = x.max(axis=1)
11 print('x.max2(axis=1)',max2.shape, max2.ndim, "\n",)
12
13 min3 = x.min(axis=2)
14 print('x.min3(axis=2)',min3.shape, min3.ndim, "\n",)
15 max3 = x.max(axis=2)
16 print('x.max3(axis=2)',max3.shape, max3.ndim, "\n",)
```

输出结果：

```
1  x.min1(axis=0) (7, 4) 2
2
3  x.max1(axis=0) (7, 4) 2
4
5  x.min2(axis=1) (4, 4) 2
6
7  x.max2(axis=1) (4, 4) 2
8
9  x.min3(axis=2) (4, 7) 2
10
11 x.max3(axis=2) (4, 7) 2
```

任务：numpy数组的聚合函数清单

numpy提供了很多聚合函数，下表提供了聚合函数的清单。NaN安全版本表示，数组含有缺失值时的聚合操作，后面小节会涉及到缺失值数组。

表格最右边栏不能完全显示，可以拖动滚动条显示结果。

函数名称	NaN安全版本	描述
`np.sum`	`np.nansum`	计算元素的和

函数名称	NaN安全版本	描述
`np.prod`	`np.nanprod`	计算元素的积
`np.mean`	`np.nanmean`	计算元素的平均值
`np.std`	`np.nanstd`	计算元素的标准差
`np.var`	`np.nanvar`	计算元素的方差
`np.min`	`np.nanmin`	找出最小值
`np.max`	`np.nanmax`	找出最大值
`np.argmin`	`np.nanargmin`	找出最小值的索引
`np.argmax`	`np.nanargmax`	找出最大值的索引
`np.median`	`np.nanmedian`	计算元素的中位数
`np.percentile`	`np.nanpercentile`	计算基于元素排序的统计值
`np.any`	`N/A`	验证任何一个元素是否为真
`np.all`	`N/A`	验证所有元素是否为真

习题:

- 随机生成一个 `size=(4,7)` 的二维整形数组。任选2种聚合方法(之前没学过的)，设置不同 `axis` 参数对数组进行聚合，并分析结果。

  代码:

```
x = np.random.randint(10, size=(4,7))

mean1 = x.mean(axis=0)
print('x.mean1(axis=0)',mean1)
var1 = x.var(axis=0)
print('x.var1(axis=0)',var1)

mean2 = x.mean(axis=1)
print('x.mean2(axis=1)',mean2)
var2 = x.var(axis=1)
print('x.var2(axis=1)',var2)

```

输出结果：

```
x.mean1(axis=0) [4.25 3.5  4.   5.   5.25 4.5  2.75]
x.var1(axis=0) [ 6.6875  8.75   12.5    10.      7.1875  6.75
 1.1875]
x.mean2(axis=1) [4.85714286 3.28571429 5.42857143 3.14285714]
x.var2(axis=1) [11.26530612  6.7755102   6.81632653  4.12244898]
```

6.1.4 实例：分析美国总统身高特征

任务：分析美国总统身高特征

用numpy的聚合功能来概括一组数据的信息是非常有用的。下面将通过一个简单的例子了解这项功能在实际问题中的应用——计算所有美国总统的身高。这个数据在president_heights.csv文件中. 需要用Pandas模块来读文件并抽取身高信息。(请注意，身高的计量单位是厘米)。后续章节会详细解释Pandas。

> 知识点：如果发现模块不存在，安装 `pandas` 、 `matplotlib` 和 `seaborn` 模块。在控制台中，运行 `conda install pandas matplotlib seaborn` 命令进行安装(并选中 `Y`)。

任务：通过pandas模块导入数据

```
import pandas as pd
data = pd.read_csv('president_heights.csv')
heights = np.array(data['height(cm)'])
print(heights)
```

任务：通过聚合方法概括数组信息

针对身高数组 `heights` ，使用聚合函数获得各种概括的统计信息。

```
print("Mean height:       ", heights.mean())
print("Standard deviation:", heights.std())
print("Minimum height:    ", heights.min())
print("Maximum height:    ", heights.max())
```

在这个例子中，聚合函数将整个数组缩减到一个值(思考为何是一个值而不是数组)，通过这些统计信息值，可以掌握 `heights` 数组的分布信息。下面获得分位数统计信息。

```
print("25th percentile:   ", np.percentile(heights, 25))
print("Median:            ", np.median(heights))
print("75th percentile:   ", np.percentile(heights, 75))
```

任务：可视化结果

可以看到，美国总统的身高中位数是 182cm，或者说不到 6 英尺。当然，有些时候将数据可视化更有用。这时可以先进行一个快速的可视化，通过Matplotlib模块(第4章将详细讨论该工具)建立可视化结果。

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn; seaborn.set()  # 设置绘图风格

plt.hist(heights)
plt.title('Height Distribution of US Presidents')
plt.xlabel('height (cm)')
plt.ylabel('number');
```

6.2 (重要)数组向量化技术之广播

6.2.1 认识NumPy的广播机制

任务：认识NumPy的广播机制

> 知识点：之前学习了NumPy是如何通过**向量化**技术(通用函数)来避免缓慢的Python循环。二元通用函数的向量化运算，要求两个参与运算的数组 `.shape` **必须相同**。
>
> - 因为需要对两个运算数组的相应元素进行逐个计算。
> - 如果 `.shape` 不匹配，必定有某些数组元素无法参与运算，致使程序出错。
> - 为解决数组 `.shape` 不匹配问题，NumPy专门设计了一套广播机制(broadcasting)。
> - **在满足一定条件下**，NumPy在底层通过广播机制，修正运算的数组 `.shape` 匹配。进而允许 `.shape` 不匹配的数组，可以参与向量化运算。

- 广播技术堪称NumPy数组向量化的黑科技，它为NumPy通用函数带来了高效和便捷。

```
import numpy as np
a = np.array([0, 1, 2])
print('a:', a)
b = np.array([5, 5, 5])
print('b:', b)
print('a+b:', a + b)
```

知识点：

- 在满足一定规则下，广播(broadcasting)允许二元通用函数输入**不同大小的数组**。
- 标量可认为是**零维数组**，它的 `.ndim` 为0； `.shape` 为 `()` 。

6.2.2 广播机制的使用

任务：广播之标量和数组运算

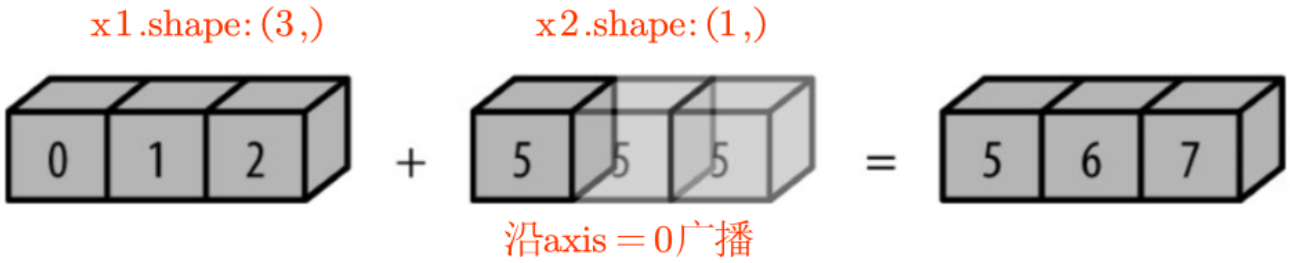
下面给出标量和一维数组的通用函数运算(相加)。

```
# 输出一维数组的维度和shape
x1 = np.arange(3)
print(x1, x1.shape, x1.ndim)

# 输出标量的维度和shape
x2 = np.array(5)
print(x2, x2.shape, x2.ndim)

# 调用二元通用函数(相加)进行运算，底层调用广播机制
print(x1 + x2)
```

知识点：广播机制的理解。

- 注意： `x2` 是标量 `.shape` 为 `()` ，**可等价视为一维数组**，即 `.shape` 为 `(1,)` 。下面对 `x2` 的分析都采用一维数组方式。
- `x1` 的 `.shape` 为 `(3,)` 和 `x2` 的 `.shape` 为 `(1,)` 。

- 参与二元通用函数运算的 `x1` 和 `x2` 不匹配。
- 广播机制直观理解(结合下图):
-
    - 首先NumPy对标量 `x2` **采用元素复制的方式**，将它扩展为 `[5, 5, 5]` 一维向量。
    - 此时新的 `x2` 的 `.shape` 为 `(3,)` ，和 `x1` 匹配，满足通用函数运算条件。
    - 然后，执行二元通用函数(相加)运算。
- 上面仅是广播机制的直观解释，实质上NumPy并非直接通过复制元素的方式，来匹配运算数组的 `.shape` ，而是采用了一种更高效的方式(这里就不展开讨论)。



浅色的盒子表示广播的值。同样需要注意的是，这个额外的内存并没有在实际操作中进行分配，但是这样的想象方式更方便我们从概念上理解。

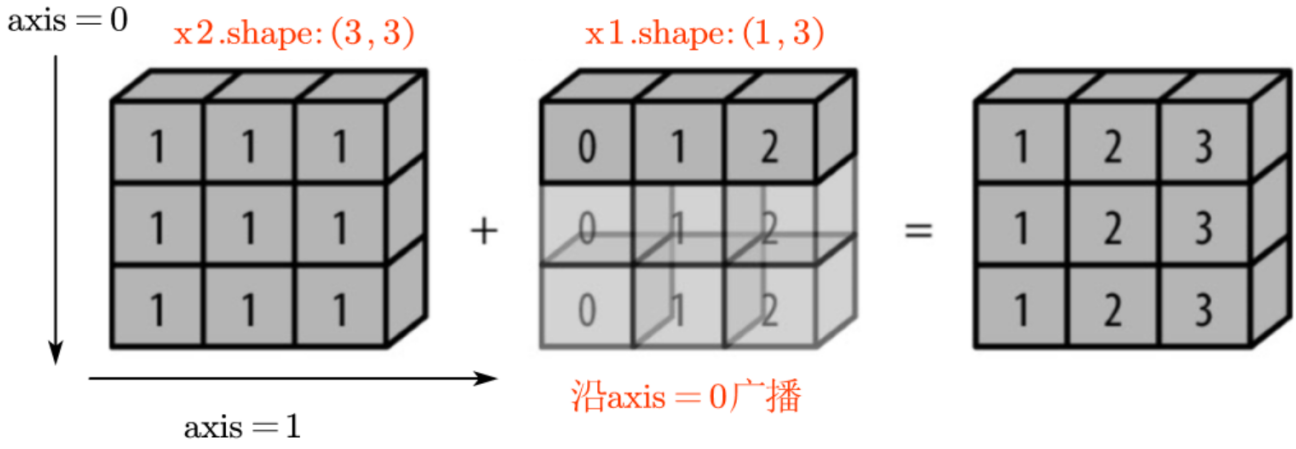
任务：广播之一维和二维数组运算

广播机制同样适用于更高维数组之间的运算。下面给出一维数组和二维数组的通用函数运算(相加)。

```
# 输出一维数组的维度和shape
x1 = np.arange(3)
print(x1, x1.shape, x1.ndim)

# 输出二维数组的维度和shape
x2 = np.ones((3, 3))
print(x2, x2.shape, x2.ndim)

# 调用二元通用函数(相加)进行运算，底层调用广播机制
print(x1 + x2)
```

知识点：广播机制的理解。

- 注意： `x1` 是一维向量 `.shape` 为 `(3,)` ，**可等价视为二维数组(行向量)**，即 `.shape` 为 `(1,3)` 。下面对 `x1` 的分析都采用行向量。
- `x1` 的 `.shape` 为 `(1,3)` 和 `x2` 的 `.shape` 为 `(3,3)` 。
- 参与二元通用函数运算的 `x1` 和 `x2` 不匹配。
- 广播机制直观理解(结合下图理解)：
-
  - 观察可知， `x1` 在第1维度(`axis=0`)的 `.shape` 和 `x2` 不匹配。
  - 那么NumPy对 `x1` 将沿着第1维度扩展至 `.shape` 为 `(3,3)` 的二维数组(类似复制元素的方式)。
  - 此时新的 `x1` 的 `.shape` 为 `(3,3)` ，和 `x2` 匹配，满足通用函数运算条件。
  - 然后，执行二元通用函数(相加)运算。



浅色的盒子表示广播的值。

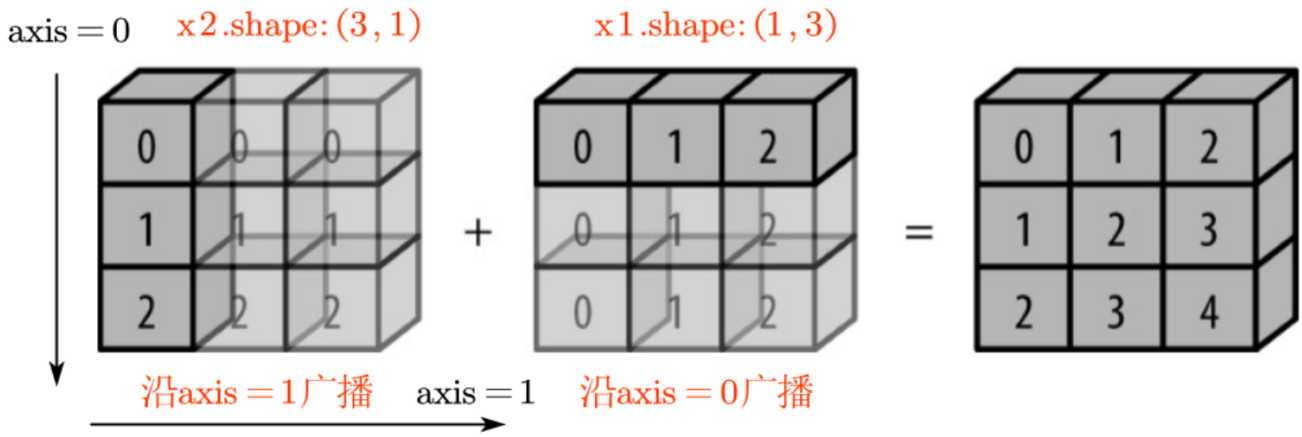
任务：两个数组同时广播

上面两个例子都是对其中的一个数组进行广播。下面，将学习更复杂的两个数组同时广播。

```
# 输出一维数组的维度和shape
x1 = np.arange(3)
print(x1, x1.shape, x1.ndim)

# 将x1通过np.newaxis升维为二维数组(列向量)
# np.newaxis如何升维请自行回顾(之前知识点)
x2 = x1[:, np.newaxis]
# 输出二维数组(列向量)的维度和shape
print(x2, x2.shape, x2.ndim)

# 调用二元通用函数(相加)进行运算，底层调用广播机制
print(x1 + x2)
```

知识点：广播机制的理解。

- 注意： `x1` 是一维向量 `.shape` 为 `(3,)` ，**在NumPy中可等价视为二维数组(行向量)**，即 `.shape` 为 `(1,3)` 。下面对 `x1` 的分析都采用行向量。
- `x1` 的 `.shape` 为 `(1,3)` 和 `x2` 的 `.shape` 为 `(3,1)` 。
- 参与二元通用函数运算的 `x1` 和 `x2` 不匹配。
- 广播机制直观理解(结合下图理解)：
-
    - 观察可知， `x1` 和 `x2` 在两个维度上都不匹配。
    - 首先，NumPy对 `x1` 将沿着第1维度扩展至 `.shape` 为 `(3,3)` 的二维数组(类似复制元素的方式)。
    - 然后，NumPy对 `x2` 将沿着第2维度扩展至 `.shape` 为 `(3,3)` 的二维数组(类似复制元素的方式)。
    - 此时新的 `x1` 的 `.shape` 为 `(3,3)` ，和 `x2` 的 `.shape` 为 `(3,3)` ，它们相互匹配，满足通用函数运算条件。
    - 然后，执行二元通用函数(相加)运算。



浅色的盒子表示广播的值。

6.2.3 (重要)广播的两大规则

任务：掌握广播两大规则

知识点：并非所有的数组都能使用NumPy的广播功能。NumPy设定了一组严格规则，只有满足了才能进行数组广播。

- 规则1(`.ndim` 不匹配)：如果两个数组 `.ndim` 维度数不相同，那么较小 `.ndim` 的数组 `.shape` 将会在**最左边补1**。

- 规则2(.shape 不匹配)：如果两个数组 .shape 存在不匹配。检查不匹配的那维 .shape **小的值**是否等于1，如果是，那么就按大的 .shape 值进行广播；如果为否，就会出现异常。
- 口诀：先看 .ndim ；后看 .shape ，检查不匹配维度是否有1；都不满足，那么就会异常。

任务：一个数组的广播

二维数组与一维数组相加。

```
x1 = np.ones((2, 3))
print(x1.ndim, x1.shape,'\n', x1)

x2 = np.arange(3)
print(x2.ndim, x2.shape,'\n', x2)

print(x1 + x2)
```

请仔细看下面的广播规则分析，掌握2个规则的适用场合。

广播规则分析：

- x1 的 .ndim: 2 ， .shape: (2,3) 。
- x2 的 .ndim: 1 ， .shape: (3,) 。
- x2 应用规则1：由于 x2 的 .ndim 小于 x1 ，将 x2 的 .shape 左边补1为 (1,3) 。
- x2 应用规则2：由于新 x2 的 .shape: (1,3) ，和 x1 在第1维上不匹配，且 x2 在该维的 .shape 为1，满足规则2。因此，将 x2 广播为 .shape: (2,3) 。
- 最终两个数组 .shape: (2,3) ，满足通用函数运算要求。

任务：两个数组的广播

行向量和列向量相加。

```
x1 = np.arange(3).reshape((3, 1))
print(x1.ndim, x1.shape,'\n', x1)

x2 = np.arange(3)
print(x2.ndim, x2.shape,'\n', x2)

print(x1 + x2)
```

广播规则分析：

- x1 的 .ndim: 2 ， .shape: (3,1) 。
- x2 的 .ndim: 1 ， .shape: (3,) 。
- x2 应用规则1：由于 x2 的 .ndim 小于 x1 ，将 x2 的 .shape 左边补1为 (1,3) 。
- x2 应用规则2：由于新 x2 的 .shape: (1,3) ，和 x1 在第1维上不匹配，且 x2 在该维的 .shape 为1，满足规则2。因此，将 x2 广播为 .shape: (3,3) 。
- x1 应用规则2：由于新 x2 的 .shape: (3,3) ，和 x1 在第2维上不匹配，且 x1 在该维的 .shape 为1，满足规则2。因此，将 x1 广播为 .shape: (3,3) 。
- 最终两个数组 .shape: (3,3) ，满足通用函数运算要求。

任务：广播异常处理

两个数组广播如果都不满足上述两条规则，那么就会发生异常。

```
x1 = np.ones((3, 2))
print(x1.ndim, x1.shape,'\n', x1)

x2 = np.arange(3)
print(x2.ndim, x2.shape,'\n', x2)

print(x1 + x2)
```

广播规则分析：

- x1 的 .ndim: 2 ， .shape: (3,2) 。
- x2 的 .ndim: 1 ， .shape: (3,) 。
- x2 应用规则1：由于 x2 的 .ndim 小于 x1 ，将 x2 的 .shape 左边补1为 (1,3) 。
- x2 应用规则2：由于新 x2 的 .shape: (1,3) ，和 x1 在第1维上不匹配，且 x2 在该维的 .shape 为1，满足规则2。因此，将 x2 广播为 .shape: (3,3) 。
- x1 应用规则2：由于新 x2 的 .shape: (3,3) ，和 x1 在第2维上不匹配，且 x1 在该维的 .shape 为2，**不满足规则2**(小的 .shape 必需要为1)。
- 程序将会报错(Traceback指出广播异常)。不满足通用函数运算要求。

任务：手动升维来满足广播条件

> 注意：
>
> - 如果能在 `x1` 数组的右边补1，而不是在左边补1，那么 `x1` 和 `x2` 后续的 `.shape` 就会变得兼容。
> - **但是这不被广播的规则所允许，因为不满足规则1(必需左边补1)。**
> - 如果逻辑上允许右边补1，那么可以数组变形函数 `reshape()` 或 `np.newaxis` 的方式手动升维。

```
x1 = np.ones((3, 2))
print(x1.ndim, x1.shape,'\n', x1)

# 使用np.newaxis在第2维上升维
# 可合并为x2 = np.arange(3)[:, np.newaxis]
x2 = np.arange(3)
x2 = x2[:, np.newaxis]
print(x2.ndim, x2.shape,'\n', x2)

print(x1 + x2)
```

6.2.4 实例：广播的应用

应用广播：数组归一化处理

> 知识点：
>
> - NumPy向量化技术包括：通用函数和广播技术。
> - 通用函数让NumPy用户免于低效的Python循环。
> - 广播技术扩展了NumPy向量化适用范围。

> 知识点：在数据预处理中，通常都需要对**数据进行归一化**。那么为何需要数据归一化?
>
> - 数据由样本组成，而样本是由特征所组成。
> - 通常，数组的行代表样本，列代表特征。(联想Excel表格)。
> - 例如，现在有一个身高体重的数据集，那么样本代表一条记录，每个样本都是有2个特征(身高和体重)组成。
> - 由于每个特征的量纲(单位)不一样，如果直接使用原始数据处理，将会对某些数值较小的特征造成偏见。为此，需要使用**数据归一化技术，将每个特征都进**

> **行无量纲化。**

假设数据集有10个样本，每个样本包含3个特征(属性或数值)，那么可以使用10×3的数组存放该数据。

```
# 随机生成10×3的数据数组
# 数组每行为1个样本，每个列为数据的1个特征。
X = np.random.random((10, 3))
print(X)

# 调用通用函数mean()沿第0轴(样本)计算均值(聚合)
Xmean = X.mean(axis=0)
print('mean:', Xmean)
```

```
# 中心化处理：对数组每个行加去特征矩阵
# 思考这里是如何进行广播的？
X_centered = X - Xmean

# 输出中心化后，每个特征的均值
# 查看归一化的数组的均值是否接近0。
# 计算机是
X_centered.mean(0)
计算机是对浮点数表示存在误差。在机器精度范围内，上述均值为 0。
```

应用广播：生成二维函数

下面将通过广播技术，生成一个二维函数，并绘制该函数图像。

定义函数, 和的定义域为 `[0, 50)` 。这里将 `x` 作为第1维坐标， `y` 作为第2维坐标。使用 `np.linspace()` 生成数据。

```
# x和y表示0~5区间50个步长的序列
# 二维数组的列向量(axis=0)
x = np.linspace(0, 5, 50)[:, np.newaxis]

# 一维数组，广播规则1，等价于
# 二维数组的行向量(axis=1)
# np.linspace(0, 5, 50)[np.newaxis,:]
y = np.linspace(0, 5, 50)
```

```
# 根据z的二元函数，生成函数值
# 思考：z的`.shape`是多少？
# 思考广播技术是如何进行的？
z = np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)
```

```
# 使用matplotlib模块绘制函数图像
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn; seaborn.set()  # 设置绘图风格

# 绘制函数等高线
fig, ax = plt.subplots()
CS = plt.contour(z)
# 设置等高线上的文本
ax.clabel(CS, inline=1, fontsize=10)
```