Numpy基本使用

一、numpy概述

1. Numerical Python，数值的Python，补充了Python语言所欠缺的数值计算能力。
2. Numpy是其它数据分析及机器学习库的底层库。
3. Numpy完全标准C语言实现，运行效率充分优化。
4. Numpy开源免费。

1. numpy发展历史

1. 1995年，Numeric，Python语言数值计算扩充。
2. 2001年，Scipy->Numarray，多维数组运算。
3. 2005年，Numeric+Numarray->Numpy。
4. 2006年，Numpy脱离Scipy成为独立的项目。

2. numpy的核心：多维数组

1. 代码简洁：减少Python代码中的循环。
2. 底层实现：厚内核(C)+薄接口(Python)，保证性能。

二、numpy基础

1. ndarray数组

用np.ndarray类的对象表示n维数组

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

1）内存中的ndarray对象

元数据（metadata）

存储对目标数组的描述信息，如：dim count、dimensions、dtype、data等。

实际数据

完整的数组数据。将实际数据与元数据分开存放，一方面提高了内存空间的使用效率，另一方面减少对实际数据的访问频率，提高性能。

2）ndarray数组对象的特点

1. Numpy数组是同质数组，即所有元素的数据类型必须相同
2. Numpy数组的下标从0开始，最后一个元素的下标为数组长度减1

3）ndarray数组对象的创建

np.array(任何可被解释为Numpy数组的逻辑结构)

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

np.arange(起始值(0),终止值,步长(1))

```
import numpy as np

# 创建0~5(不包含5)，步长为1的数组
a = np.arange(0, 5, 1)
print(a)

# 创建0~10(不包含10)，步长为2的数组
b = np.arange(0, 10, 2)
print(b)
```

np.zeros(数组元素个数, dtype='类型')

```
import numpy as np

# 创建值全为0、包含10个元素的数组
a = np.zeros(10)
print(a)
```

np.ones(数组元素个数, dtype='类型')

```
import numpy as np

# 创建值全为1、包含10个元素的数组
a = np.ones(10)
print(a)
```

4）ndarray对象属性的基本操作

数组的维度：np.ndarray.shape

元素的类型：np.ndarray.dtype

数组元素的个数：np.ndarray.size

```
import numpy as np

a = np.array([[[1, 2, 3],
               [3, 4, 5]],
              [[5, 6, 7],
               [7, 8, 9]]])
print(a.shape)  # 打印a对象的形状

print(a[0])  # 取第0页
print(a[0][0])  # 取第0页第0行
print(a[1][0][1])  # 取第0页第0行第1列
print(a[1, 0, 1])  # 取第0页第0行第1列另一种写法

# 遍历三维数组
for i in range(a.shape[0]):  # 遍历每页
    for j in range(a.shape[1]):  # 遍历每行
        for k in range(a.shape[2]):  # 遍历每列
            print(a[i, j, k], end=" ")  # 每列后加个空格分隔
        print("")  # 每行结束后换行
```

5）ndarray对象属性操作详解

① 基本数据类型

类型名	类型表示符
布尔型	bool_
有符号整数型	int8(范围-128~127)/int16/int32/int64
无符号整数型	uint8(范围0~255)/uint16/uint32/uint64
浮点型	float16/float32/float64
复数型	complex64/complex128
字串型	str_，每个字符用32位Unicode编码表示

类型字符码

类型	字符码
np.bool_	?
np.int8/16/32/64	i1/i2/i4/i8
np.uint8/16/32/64	u1/u2/u4/u8
np.float/16/32/64	f2/f4/f8
np.complex64/128	c8/c16
np.str_	U<字符数>
np.datetime64	M8[Y] M8[M] M8[D] M8[h] M8[m] M8[s]

示例：自定义符合类型数组

```
# 自定义复合类型
import numpy as np

data = [
    ('zs', [90, 80, 85], 15),
    ('ls', [92, 81, 83], 16),
    ('ww', [95, 85, 95], 15)
]
# 第一种设置dtype的方式
a = np.array(data, dtype='U3, 3int32, int32')  # 3个Unicode字符，3个int32，1个int32
print(a)
print(a[0]['f0'])  # 打印第0个元素第1个域
print(a[1]['f1'])  # 打印第1个元素第1个域

print("====================================")

# 第二种设置dtype的方式
b = np.array(data, dtype=[('name', 'str_', 2),  # 第一个字段：名称、类型、长度
                          ('scores', 'int32', 3),  # 第二个字段：名称、类型、长度
                          ('ages', 'int32', 1)])  # 第三个字段：名称、类型、长度
print(b[0]['name'], ":", b[0]['scores'])  # 打印0号元素name域，scores域

print("====================================")

# 第三种设置dtype的方式
c = np.array(data, dtype={'names': ['name', 'scores', 'ages'],  # 所有域名称列表
                          'formats': ['U3', '3int32', 'int32']})  # 所有域类型列表
print(c[0]['name'], ":", c[0]['scores'], ":", c.itemsize)  # 打印0号元素name域，scores域，对象长度

print("====================================")

# 第四种设置dtype的方式
d = np.array(data, dtype={'names': ('U3', 0),  # 第一个域名称、类型、偏移
                          'scores': ('3int32', 16),  # 第二个域名称、类型、偏移
```

```
                            'ages': ('int32', 28)})  # 第三个域名称、类型、偏移
print(d[0]['names'], d[0]['scores'], d.itemsize)
```

示例：日期类型使用

```
# 测试日期类型数组
import numpy as np

f = np.array(['2020', '2020-01-01', '2021-01-01 01:01:01', '2020-02-01'])
f = f.astype('M8[D]') # 将f数组转换为日期
print(f)

f = f.astype('int32') # 转换为整数，为离计算机元年的天数
print(f)
print(f[3] - f[0]) # 第3个元素减第0个元素的值，为相差天数
```

示例：虚数类型使用

```
# 虚数类型示例
import numpy as np

a = np.array([[1 + 1j, 2 + 4j, 3 + 7j],
              [4 + 2j, 5 + 5j, 6 + 8j],
              [7 + 3j, 8 + 6j, 9 + 9j]])
print(a.T) # 转置

for x in a.flat:
    print(x.real, ", ", x.imag)
```

② 维度操作

视图变维（数据共享）： reshape() 与 ravel()

```
import numpy as np

a = np.arange(1, 9)
print(a)  # [1 2 3 4 5 6 7 8]

b = a.reshape(2, 4)  # 视图变维，变为2行4列的二维数组
print(b)

c = b.reshape(2, 2, 2)  # 视图变维，变为2页2行2列的三维数组
print(c)

d = c.ravel()  # 视图变维，变为1维数组
print(d)
```

就地变维：直接改变原数组对象的维度，不返回新数组

```
d = c.ravel()  # 视图变维，变为1维数组，使用原数据
print(d)
```

复制变维（数据独立）： flatten()

```
# 拉升成1维，返回一份新数据
e = c.flatten()
print(e)
a += 10
print(a, e, sep='\n')
```

③ 切片操作

```
#数组对象切片的参数设置与列表切面参数类似
#   步长+：默认切从首到尾
#   步长-：默认切从尾到首
数组对象[起始位置:终止位置:步长, ...]
#默认位置步长：1
```

```
import numpy as np

a = np.arange(1, 10)
print(a)  # 1 2 3 4 5 6 7 8 9

print(a[:3])  # 1 2 3
print(a[3:6])   # 4 5 6
print(a[6:])  # 7 8 9
print(a[::-1])  # 9 8 7 6 5 4 3 2 1
print(a[:-4:-1])  # 9 8 7
print(a[-4:-7:-1])  # 6 5 4
print(a[-7::-1])  # 3 2 1
print(a[::])  # 1 2 3 4 5 6 7 8 9
print(a[:])  # 1 2 3 4 5 6 7 8 9
print(a[::3])  # 1 4 7
print(a[1::3])  # 2 5 8
print(a[2::3])  # 3 6 9
```

ndarray数组的掩码操作

```
import numpy as np

a = np.arange(1, 6, 1)  # 在1~6(不包含6)之间每隔1产生一个数字
print(a)

mask = [True, False, True, False, False]
print(a[mask]) # 掩码操作，对应True的位置上的元素保留，False位置的元素丢弃
```

多维数组的切片操作

```
import numpy as np

a = np.arange(1, 28) # 产生1~28(不包含28)范围内的数字
a.resize(3, 3, 3)
print(a)

print("====================")
# 切出1页
print(a[1, :, :])

print("====================")
# 切出所有页的1行
print(a[:, 1, :])

print("====================")
# 切出0页所有行的第1列
print(a[0, :, 1])
```

④ 组合与拆分

垂直方向操作:

```
import numpy as np

a = np.arange(1, 7).reshape(2, 3)
b = np.arange(7, 13).reshape(2, 3)
print(a)
print(b)
print("===============")

# 垂直方向完成组合操作，生成新数组
c = np.vstack((a, b))
print(c)
print("===============")

# 垂直方向完成拆分操作，生成两个数组
d, e = np.vsplit(c, 2) # 拆分时垂直方向上的元素要能整除个数
print(d)
print(e)
```

水平方向操作:

```
import numpy as np

a = np.arange(1, 7).reshape(2, 3)
b = np.arange(7, 13).reshape(2, 3)
print(a)
print(b)
print("===============")

# 水平方向完成组合操作，生成新数组
```

```
c = np.hstack((a, b))
print(c)
print("===============")

# 水平方向完成拆分操作，生成两个数组
d, e = np.hsplit(c, 2)
print(d)
print(e)
```

长度不等的数组组合，可以通过填充，填充成相等长度进行合并。例如：

```
import numpy as np

a = np.array([1, 2, 3, 4, 5])
b = np.array([1, 2, 3, 4])
print(a)
print(b)
print("========================")

# 填充b数组使其长度与a相同
b = np.pad(b, pad_width=(0, 1),  # 对b进行填充，前面填充0个，后面填充1个
           mode='constant',  # 填充常数值
           constant_values=-1)  # 填充常数值为-1
print(b)
print("========================")

# 垂直方向完成组合操作，生成新数组
c = np.vstack((a, b))
print(c)

```

多维数组组合与拆分的相关函数：

```
# 通过axis作为关键字参数指定组合的方向，取值如下：
# 若待组合的数组都是二维数组：
#   0：垂直方向组合
#   1：水平方向组合
# 若待组合的数组都是三维数组：
#   0：垂直方向组合
#   1：水平方向组合
#   2：深度方向组合
np.concatenate((a, b), axis=0)
# 通过给出的数组与要拆分的份数，按照某个方向进行拆分，axis的取值同上
np.split(c, 2, axis=0)
```

简单的一维数组组合方案

```
import numpy as np

a = np.arange(1, 5)  # [1, 2, 3, 4]
b = np.arange(6, 10)  # [6, 7, 8, 9]
print(a)
print(b)
print("========================")
```

```
8
9  # 把两个数组摞在一起成两行
10 c = np.row_stack((a, b))
11 print(c)
12 print("======================")
13
14 # 把两个数组组合在一起成两列
15 d = np.column_stack((a, b))
16 print(d)
17
```

6）ndarray类的其他属性

- shape - 维度
- dtype - 元素类型
- size - 元素数量
- ndim - 维数，len(shape)
- itemsize - 元素字节数
- nbytes - 总字节数 = size x itemsize
- real - 复数数组的实部数组
- imag - 复数数组的虚部数组
- T - 数组对象的转置视图
- flat - 扁平迭代器

```
1  import numpy as np
2
3  a = np.array([[1 + 1j, 2 + 4j, 3 + 7j],
4                [4 + 2j, 5 + 5j, 6 + 8j],
5                [7 + 3j, 8 + 6j, 9 + 9j]])
6  print(a.ndim)  # 维数
7  print(a.itemsize)  # 元素大小
8  print(a.nbytes)  # 总大小
9  print(a.real, a.imag, sep='\n')  # 分别打印出实部、虚部
10 print(a.T)  # 转置
11
12 print([elem for elem in a.flat])  # 扁平迭代
13
14 b = a.tolist()  # 转换成python列表
15 print(b)
```

2. ndarray与list的互转

- 方式一：利用列表创建一个数组

```
1  import numpy as np
2
3  # 创建一个一维 numpy 数组
4  arr1 = np.array([1, 2, 3, 4, 5])
5  print(type(arr1))
6
7  # 使用 tolist() 方法转换为列表
8  L = arr1.tolist()
9  print(type(L))
10
11 print(L)  # 输出: [1, 2, 3, 4, 5]
```

- 方式二：asarray函数

```
# asarray将列表转行为数组
arr2 = np.asarray(L)
print(type(arr2))
print(arr2)
```

- 方式三：列表推导式

```
import numpy as np

# 创建一个一维 numpy 数组
array_1d = np.array([1, 2, 3, 4, 5])

# 使用列表推导式转换为列表
list_1d = [x for x in array_1d]
print(type(list_1d))
print(list_1d)  # 输出: [1, 2, 3, 4, 5]
```

3. ndarray与list的比较

NumPy数组（ndarray）和Python列表（list）都是数据结构，但它们有不同的用途和性能特点。

- 归属：list是python内置类型，ndarray是Numpy库的类型；
- 性能：在数值计算中，NumPy数组的性能通常优于Python列表。这是因为NumPy数组是在C语言中实现的，而Python列表是在Python虚拟机中实现的；
- 功能：NumPy数组提供了许多高级的数学函数和操作，这些在列表中需要自己手动实现；
- 存储类型：NumPy数组中的所有元素必须是相同的类型，而Python列表可以包含不同类型的元素；
- 索引和切片：NumPy数组支持高效的索引和切片操作，而Python列表的这些操作通常较慢。