NumPy数组基本用法

1. `Numpy`是`Python`科学计算库，用于快速处理任意维度的数组。
2. `NumPy`中最核心的部分是**N维数组类型ndarray**，它描述了**相同类型**的“items”的集合。
3. `numpy.ndarray`支持向量化运算。
4. `NumPy`使用c语言写的，底部解除了`GIL`，其对数组的操作速度不受`Python`解释器限制，性能远远优于原生`Python`，基本是一到两个数量级的差距，而且数据量越大，`Numpy`的优势更明显。

Numpy中的数组：

`Numpy`中的数组的使用跟`Python`中的列表非常类似。他们之间的区别如下：

1. 一个列表中可以存储多种数据类型。比如`a = [1,'a']`是允许的，而数组只能存储同种数据类型。
2. 数组可以是多维的，当多维数组中所有的数据都是数值类型的时候，相当于线性代数中的矩阵，是可以进行相互间的运算的。

创建数组（np.ndarray对象）：

`Numpy`经常和数组打交道，因此首先第一步是要学会创建数组。在`Numpy`中的数组的数据类型叫做`ndarray`。以下是两种创建的方式：

1. 根据`Python`中的列表生成：

```
import numpy as np
a1 = np.array([1,2,3,4])
print(a1)
print(type(a1))
```

2. 从头开始创建数组，使用`np.arange`和`np.linspace`生成

   `np.arange`的用法类似于`Python`中的`range`：

```
import numpy as np
a1 = np.arange(2,21,2)
print(a1)
```

   `np.linspace`将创建具有指定数量元素的数组，并在指定的开始值和结束值之间平均间隔：

```
import numpy as np
a2 = np.linspace(0,1,9)
print(a2)
```

3. 使用`np.random`生成随机数的数组：

```
a1 = np.random.random((2,2)) # 生成2行2列的随机数的数组
a2 = np.random.randint(0,10,size=(3,3)) # 元素是从0-10之间随机的3行3列的数组
```

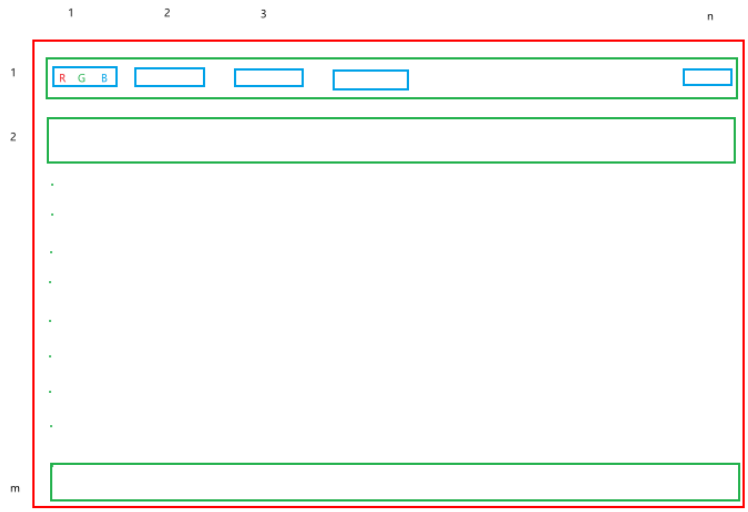
4. 使用函数生成特殊的数组：

```
import numpy as np
a1 = np.zeros((2,2)) #生成一个所有元素都是0的2行2列的数组
a2 = np.ones((3,2)) #生成一个所有元素都是1的3行2列的数组
a3 = np.full((2,2),8) #生成一个所有元素都是8的2行2列的数组
a4 = np.eye(3) #生成一个在斜方形上元素为1，其他元素都为0的3x3的矩阵
a5 = np.empty((3,3)) #生成一个3行3列的数组，数组元素内容初始化为随机值，取决于内存的状态
```

5. 从文件中获取对应的数组

   比如可以利用matplotlib的imread获取图像，生成一个三维数组(M,N,RGB)，该三维数组分别对应图片的高、宽、RGB值

```
import matplotlib.pyplot as plt  #需要导入matplotlib库
img_path = "Numpy/res/guido.jpg"  #图片所在路径
a5_img = plt.imread(img_path)   #通过imread方法获取图片，生成一个图片三维数组
```



数据类型

`NumPy`支持比`Python`更多种类的数字类型。支持的原始类型与 C 中的原始类型紧密相关：

NUMPY 的类型	C 的类型	描述
np.bool	bool	存储为字节的布尔值（True或False）
np.byte	signed char	平台定义
np.ubyte	unsigned char	平台定义

NUMPY 的类型	C 的类型	描述
np.short	short	平台定义
np.ushort	unsigned short	平台定义
np.intc	int	平台定义
np.uintc	unsigned int	平台定义
np.int_	long	平台定义
np.uint	unsigned long	平台定义
np.longlong	long long	平台定义
np.ulonglong	unsigned long long	平台定义
np.half / np.float16		半精度浮点数：符号位，5位指数，10位尾数
np.single	float	平台定义的单精度浮点数：通常为符号位，8位指数，23位尾数
np.double	double	平台定义的双精度浮点数：通常为符号位，11位指数，52位尾数。
np.longdouble	long double	平台定义的扩展精度浮点数
np.csingle	float complex	复数，由两个单精度浮点数（实部和虚部）表示
np.cdouble	double complex	复数，由两个双精度浮点数（实部和虚部）表示。
np.clongdouble	long double complex	复数，由两个扩展精度浮点数（实部和虚部）表示。

由于其中许多都具有依赖于平台的定义，因此提供了一组固定大小的别名：

数据类型	描述	唯一标识符
bool	用一个字节存储的布尔类型（True或False）	'b'
int8	一个字节大小，-128 至 127	'i1'
int16	整数，16 位整数(-32768 ~ 32767)	'i2'
int32	整数，32 位整数(-2147483648 ~ 2147483647)	'i4'
int64	整数，64 位整数(-9223372036854775808 ~ 9223372036854775807)	'i8'
uint8	无符号整数，0 至 255	'u1'
uint16	无符号整数，0 至 65535	'u2'
uint32	无符号整数，0 至 2 ** 32 - 1	'u4'
uint64	无符号整数，0 至 2 ** 64 - 1	'u8'
float16	半精度浮点数：16位，正负号1位，指数5位，精度10位	'f2'
float32	单精度浮点数：32位，正负号1位，指数8位，精度23位	'f4' or 'f'
float64	双精度浮点数：64位，正负号1位，指数11位，精度52位	'f8' or 'd'
complex64	复数，分别用两个32位浮点数表示实部和虚部	'c8'
complex128	复数，分别用两个64位浮点数表示实部和虚部	'c16'
object_	python对象	'O'
string_	字符串	'S'

数据类型	描述	唯一标识符
unicode_	unicode类型	'U'

我们可以看到，`Numpy`中关于数值的类型比`Python`内置的多得多，这是因为`Numpy`为了能高效处理处理海量数据而设计的。举个例子，比如现在想要存储上百亿的数字，并且这些数字都不超过254（一个字节内），我们就可以将数据类型设置为`int8`，这样就比使用`int64`更能节省内存空间了。

`ndarray.dtype`：

因为数组中只能存储同一种数据类型，因此可以通过`dtype`获取数组中的元素的数据类型。以下是`ndarray.dtype`的相关操作如下：

1. 默认的数据类型：

```
import numpy as np
a1 = np.array([1,2,3])
print(a1.dtype)
# 如果是windows系统，默认是int32
# 如果是mac或者linux系统，则根据系统来
```

2. 指定`dtype`：

```
import numpy as np
a1 = np.array([1,2,3],dtype=np.int64)
# 或者 a1 = np.array([1,2,3],dtype="i8")
print(a1.dtype)
```

3. 修改`dtype`：

```
import numpy as np
a1 = np.array([1,2,3])
print(a1.dtype) # window系统下默认是int32
# 以下修改dtype
a2 = a1.astype(np.int64) # astype不会修改数组本身，而是会将修改后的结果返回
print(a2.dtype)
```

数组的属性

`ndarray.ndim`

数组的维数。比如：

```
a1 = np.array([1,2,3])
print(a1.ndim) # 维度为1
a2 = np.array([[1,2,3],[4,5,6]])
print(a2.ndim) # 维度为2
```

```
a3 = np.array([
    [
        [1,2,3],
        [4,5,6]
    ],
    [
        [7,8,9],
        [10,11,12]
    ]
])
print(a3.ndim) # 维度为3
```

ndarray.shape

数组的形状。比如以下代码：

```
a1 = np.array([1,2,3])
print(a1.shape) # 输出(3,)，意思是一维数组，有3个数据

a2 = np.array([[1,2,3],[4,5,6]])
print(a2.shape) # 输出(2,3)，意思是二维数组，2行3列

a3 = np.array([
    [
        [1,2,3],
        [4,5,6]
    ],
    [
        [7,8,9],
        [10,11,12]
    ]
])
print(a3.shape) # 输出(2,2,3)，意思是三维数组，总共有2个元素，每个元素是2行3列的

a4 = np.array([[1,2,3],[4,5]])
print(a4.shape) # 输出(2,)，意思是a4是一个一维数组，总共有2列
print(a4) # 输出[list([1, 2, 3]) list([4, 5])]，其中最外面层是数组，里面是Python列表
```

另外，我们还可以通过`ndarray.reshape`来重新修改数组的维数。示例代码如下：

```
a1 = np.arange(12) #生成一个有12个数据的一维数组
print(a1)

a2 = a1.reshape((3,4)) #变成一个2维数组，是3行4列的
print(a2)

a3 = a1.reshape((2,3,2)) #变成一个3维数组，总共有2块，每一块是3行2列的
print(a3)
```

```
a4 = a2.reshape((12,)) # 将a2的二维数组重新变成一个12列的1维数组
print(a4)

a5 = a2.flatten() # 不管a2是几维数组，都将他变成一个一维数组
print(a5)
```

> *注意，`reshape`并不会修改原来数组本身，而是会将修改后的结果返回。如果想要直接修改数组本身，那么可以使用`resize`来替代`reshape`。*

ndarray.size

获取数组中总的元素的个数。比如有个二维数组：

```
import numpy as np
a1 = np.array([[1,2,3],[4,5,6]])
print(a1.size) #打印的是6，因为总共有6个元素
```

ndarray.itemsize

数组中每个元素占的大小，单位是字节。比如以下代码：

```
a1 = np.array([1,2,3],dtype=np.int32)
print(a1.itemsize) # 打印4，因为每个字节是8位，32位/8=4个字节
```

ndarray.nbytes

数组元素消耗的总字节数，单位是字节。比如以下代码：

```
a1 = np.array([1,2,3],dtype=np.int32)
print(a1.nbytes) # 打印12，因为总共有3个元素。每个元素占4个字节
```