# **ndarry基本概念**

**ndarray是numpy库中的一种数据类型，ndarray 对象是用于存放同类型元素的多维数组，就跟pytorch中的张量tensor类似。**

**多维ndarray中，每一维都叫一个轴axis。很多对于ndarray对象的运算都是基于axis进行，比如sum、mean等都会有一个axis参数（针对对这个轴axis进行某些运算操作）。**

**对于多维数组，因为每一个轴都有一个索引，所以这些索引由逗号进行分割，例如：**

>>> x = np.arange(0, 100, 5).reshape(4, 5)

>>> x

array([[ 0, 5, 10, 15, 20],

[25, 30, 35, 40, 45],

[50, 55, 60, 65, 70],

[75, 80, 85, 90, 95]])

>>> x[1,2] #第1行，第2列

35

>>> x[1:4, 3] #第1行到第3行中所有第3列的元素

array([40, 65, 90])

>>> x[:, 4] #所有行中的所有第4列的元素

array([20, 45, 70, 95])

>>> x[0:3, :] #第0行到第三行中所有列的元素

array([[ 0, 5, 10, 15, 20],

[25, 30, 35, 40, 45],

[50, 55, 60, 65, 70]])

**当提供的索引比轴数少时，缺失的索引表示整个切片（只能缺失后边的轴）；当提供的索引为:时，也表示整个切片；可以使用...代替几个连续的:索引。**

>>> x[1:3] #缺失第二个轴

array([[25, 30, 35, 40, 45],

[50, 55, 60, 65, 70]])

>>> x[:, 1:3] #第一个轴是 :

array([[ 5 10]

[30 35]

[55 60]

[80 85]])

>>> x[..., 0:4] #...代表了第一个轴的 : 索引

array([[ 0, 5, 10, 15],

[25, 30, 35, 40],

[50, 55, 60, 65],

[75, 80, 85, 90]])

**多维数组的迭代：可以使用ndarray的flat属性迭代数组中每一个元素，**

>>> for item in x.flat:

... print item,

...

0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95

# **产生ndarray类型的情况：**

**np.array()**

**np.arange(5)，[使用方法](onenote:语言.one" \l "Python%20的range()%20vs%20numpy的arange()&section-id={A0232E0C-9F26-4EBE-8041-AA3C4823A800}&page-id={F070CDFA-7560-4E78-9958-94C5B99A8220}&base-path=//G:/研究生/OneNote笔记/专业)**

np.random.randn(5,3)

numpy.empty

**numpy.zeros**

**numpy.ones**

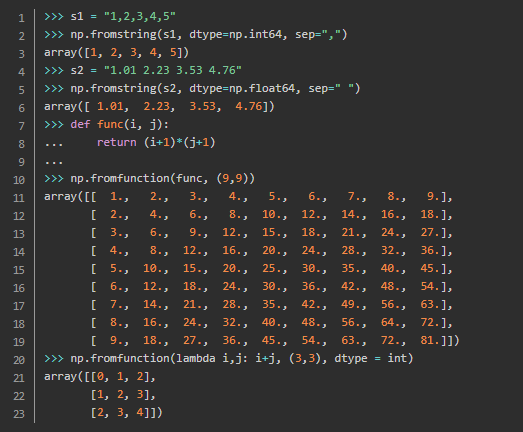
numpy.asarray

类似 numpy.array，但 numpy.asarray 参数只有三个，比 numpy.array 少两个。

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

np.fromstring，从字符串中读取数据并创建数组

np.fromfunction，由第一个参数作为计算每个数组元素的函数（函数对象或者lambda表达式均可），第二个参数为数组的形状



numpy.frombuffer

numpy.fromiter

**numpy.arange，numpy 包中的使用 arange 函数创建数值范围并返回 ndarray 对象**

**numpy.linspace，用于创建一个一维数组，数组是一个等差数列构成的，格式如下：**

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

**numpy.logspace，**与linspace类似，只不过它创建的是一个**等比数列**，也是一个一维数组

np.zeros\_like(a)，创建shape相同的多维数组：ones\_like、zeros\_like、empty\_like

diag函数返回一个矩阵的对角线元素、或者创建一个对角阵，对角线由参数k控制

diagflat函数以输入作为对角线元素，创建一个矩阵，对角线由参数k控制

tri函数生成一个矩阵，在某对角线以下元素全为1，其余全为0，对角线由参数k控制

tril函数输入一个矩阵，返回该矩阵的下三角矩阵，下三角的边界对角线由参数k控制

triu函数与tril类似，返回的是矩阵的上三角矩阵

vander函数输入一个一维数组，返回一个范德蒙德矩阵

# 花式索引，指的是利用整数数组进行索引。【原来这就是我之前一直找的那个方法！！！用下标array作为索引！】

花式索引根据索引数组的值作为目标数组的某个轴的下标来取值。对于使用一维整型数组作为索引，如果目标是一维数组，那么索引的结果就是对应位置的元素；如果目标是二维数组，那么就是对应下标的行。

花式索引跟切片不一样，它总是将数据复制到新数组中。

1、传入顺序索引数组

实例

import numpy as np

x=np.arange(32).reshape((8,4))

print (x[[4,2,1,7]])

2、传入倒序索引数组

实例

import numpy as np

x=np.arange(32).reshape((8,4))

print (x[[-4,-2,-1,-7]])

输出结果为：

[[16 17 18 19]

[24 25 26 27]

[28 29 30 31]

[ 4 5 6 7]]

3、传入多个索引数组（要使用np.ix\_）

实例

import numpy as np

x=np.arange(32).reshape((8,4))

print (x[np.ix\_([1,5,7,2],[0,3,1,2])])

输出结果为：

[[ 4 7 5 6]

[20 23 21 22]

[28 31 29 30]

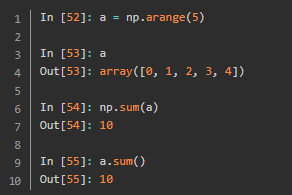
[ 8 11 9 10]]

# **ndarry常用属性：**

|  |  |
| --- | --- |
| size | 数组中元素个数 |
| itemsize | 数组中单个元素的字节长度 |
| dtype | 数组元素的数据类型对象 |
| ndim | 数组的维度（改变维度用a.reshape()） |
| shape | 数组的形状 |

# **对ndarray操作的函数：**

在numpy中，一般可直接用于ndarray类型数据上的方法也有与之对应的numpy函数可执行相同操作：



源文档 <<https://www.jianshu.com/p/0c6c14d9e963>>

# **ndarry与list互转**

list(arr)，转为list

np.array(list)，转为ndarry

# **怎么拼接两个ndarry：**

**方法3最适合大规模数据拼接。**

**方法1.先转为list，再用a\_list.extend(b\_list)拼接。**

该方法只适用于简单的一维数组拼接，转换很耗时，大量数据拼接不建议。

首先将数组转成列表，然后利用列表的拼接函数append()、extend()等进行拼接处理，最后将列表转成数组。

示例1：

>>> import numpy as np

>>> a=np.array([1,2,5])

>>> b=np.array([10,12,15])

>>> a\_list=list(a)

>>> b\_list=list(b)

>>> a\_list.extend(b\_list)

>>> a\_list

[1, 2, 5, 10, 12, 15]

>>> a=np.array(a\_list)

>>> a

array([ 1, 2, 5, 10, 12, 15])

**方法2.用numpy提供的numpy.append(arr, values, axis=None)函数。**

对于参数规定，要么一个数组和一个数值；要么两个数组，不能三个及以上数组直接append拼接。append函数返回的始终是一个一维数组。

示例2：

>>> a=np.arange(5)

>>> a

array([0, 1, 2, 3, 4])

>>> np.append(a,10)

array([ 0, 1, 2, 3, 4, 10])

>>> a

array([0, 1, 2, 3, 4])

**方法3.用numpy提供的numpy.concatenate((a1,a2,...), axis=0)函数。**

能够一次完成多个数组的拼接。其中a1,a2,...是数组类型的参数

示例3：

>>> a=np.array([1,2,3])

>>> b=np.array([11,22,33])

>>> c=np.array([44,55,66])

**>>> np.concatenate((a,b,c),axis=0) # 默认情况下，axis=0可以不写**

array([ 1, 2, 3, 11, 22, 33, 44, 55, 66]) #对于一维数组拼接，axis的值不影响最后的结果

>>> a=np.array([[1,2,3],[4,5,6]])

>>> b=np.array([[11,21,31],[7,8,9]])

>>> np.concatenate((a,b),axis=0)

array([[ 1, 2, 3],

[ 4, 5, 6],

[11, 21, 31],

[ 7, 8, 9]])

**>>> np.concatenate((a,b),axis=1) #axis=1表示对应行的数组进行拼接**

array([[ 1, 2, 3, 11, 21, 31],

[ 4, 5, 6, 7, 8, 9]])

**方法2和3时间性能比较**

对numpy.append()和numpy.concatenate()两个函数的运行时间进行比较

示例4：

>>> from time import clock as now

>>> a=np.arange(9999)

>>> b=np.arange(9999)

>>> time1=now()

>>> c=np.append(a,b)

>>> time2=now()

>>> print time2-time1

28.2316728446

>>> a=np.arange(9999)

>>> b=np.arange(9999)

>>> time1=now()

>>> c=np.concatenate((a,b),axis=0)

>>> time2=now()

>>> print time2-time1

20.3934997107

可知，concatenate()效率更高，适合大规模的数据拼接

**参考：<https://blog.csdn.net/zyl1042635242/article/details/43162031>**