NumPy 字节交换 NumPy 矩阵库(Matrix)

NumPy 副本和视图

副本是一个数据的完整的拷贝，如果我们对副本进行修改，它不会影响到原始数据，物理内存不在同一位置。

视图是数据的一个别称或引用，通过该别称或引用亦便可访问、操作原有数据，但原有数据不会产生拷贝。如果我们对视图进行修改，它会影响到原始数据，物理内存在同一位置。

视图一般发生在：

- 1、numpy 的切片操作返回原数据的视图。
- 2、调用 ndarray 的 view() 函数产生一个视图。

副本一般发生在：

- Python 序列的切片操作，调用deepCopy()函数。
- 调用 ndarray 的 copy() 函数产生一个副本。

无复制

简单的赋值不会创建数组对象的副本。 相反，它使用原始数组的相同id()来访问它。 id()返回 Python 对象的通用标识符，类似于 C 中的指针。

此外，一个数组的任何变化都反映在另一个数组上。 例如，一个数组的形状改变也会改变另一个数组的形状。

实例

```
import numpy as np
a = np.arange(6)
print ('我们的数组是: ')
print (a)
print ('调用 id() 函数: ')
print (id(a))
print ('a 赋值给 b: ')
b = a
print (b)
print ('b 拥有相同 id(): ')
print (id(b))
print ('修改 b 的形状: ')
b.shape = 3,2
print (b)
print ('a 的形状也修改了: ')
print (a)
```

输出结果为：

```
我们的数组是:
[0 1 2 3 4 5]
调用 id() 函数:
4349302224
```

```
a 赋值给 b:
[0 1 2 3 4 5]
b 拥有相同 id():
4349302224
修改 b 的形状:
[[0 1]
 [2 3]
 [4 5]]
a 的形状也修改了:
[[0 1]
 [2 3]
 [4 5]]
```

视图或浅拷贝

ndarray.view() 方会创建一个新的数组对象，该方法创建的新数组的维数更改不会更改原始数据的维数。

实例

```
import numpy as np
# 最开始 a 是个 3X2 的数组
a = np.arange(6).reshape(3,2)
print ('数组 a: ')
print (a)
print ('创建 a 的视图: ')
b = a.view()
print (b)
print ('两个数组的 id() 不同: ')
print ('a 的 id(): ')
print (id(a))
print ('b 的 id(): ' )
print (id(b))
# 修改 b 的形状，并不会修改 a
b.shape = 2,3
print ('b 的形状: ')
print (b)
print ('a 的形状: ')
print (a)
```

输出结果为：

```
数组 a:
[[0 1]
 [2 3]
 [4 5]]
创建 a 的视图:
[[0 1]
 [2 3]
 [4 5]]
两个数组的 id() 不同:
a 的 id():
```

```
4314786992
b 的 id():
4315171296
b 的形状:
[[0 1 2]
 [3 4 5]]
a 的形状:
[[0 1]
 [2 3]
 [4 5]]
```

使用切片创建视图修改数据会影响到原始数组：

实例

```
import numpy as np
arr = np.arange(12)
print ('我们的数组: ')
print (arr)
print ('创建切片: ')
a=arr[3:]
b=arr[3:]
a[1]=123
b[2]=234
print(arr)
print(id(a),id(b),id(arr[3:]))
```

输出结果为：

```
我们的数组:
[ 0  1  2  3  4  5  6  7  8  9 10 11]
创建切片:
[  0   1   2   3 123 234   6   7   8   9  10  11]
4545878416 4545878496 4545878576
```

变量 a,b 都是 arr 的一部分视图，对视图的修改会直接反映到原数据中。但是我们观察 a,b 的 id，他们是不同的，也就是说，视图虽然指向原数据，但是他们和赋值引用还是有区别的。

副本或深拷贝

ndarray.copy() 函数创建一个副本。 对副本数据进行修改，不会影响到原始数据，它们物理内存不在同一位置。

实例

```
import numpy as np
a = np.array([[10,10], [2,3], [4,5]])
print ('数组 a: ')
print (a)
print ('创建 a 的深层副本: ')
b = a.copy()
print ('数组 b: ')
```

```
print (b)
# b 与 a 不共享任何内容
print ('我们能够写入 b 来写入 a 吗? ')
print (b is a)
print ('修改 b 的内容: ')
b[0,0] = 100
print ('修改后的数组 b: ')
print (b)
print ('a 保持不变: ')
print (a)
```

输出结果为：

```
数组 a:
[[10 10]
 [ 2  3]
 [ 4  5]]
创建 a 的深层副本:
数组 b:
[[10 10]
 [ 2  3]
 [ 4  5]]
我们能够写入 b 来写入 a 吗?
False
修改 b 的内容:
修改后的数组 b:
[[100  10]
 [  2   3]
 [  4   5]]
a 保持不变:
[[10 10]
 [ 2  3]
 [ 4  5]]
```

更多相关文章

Python 直接赋值、浅拷贝和深度拷贝解析

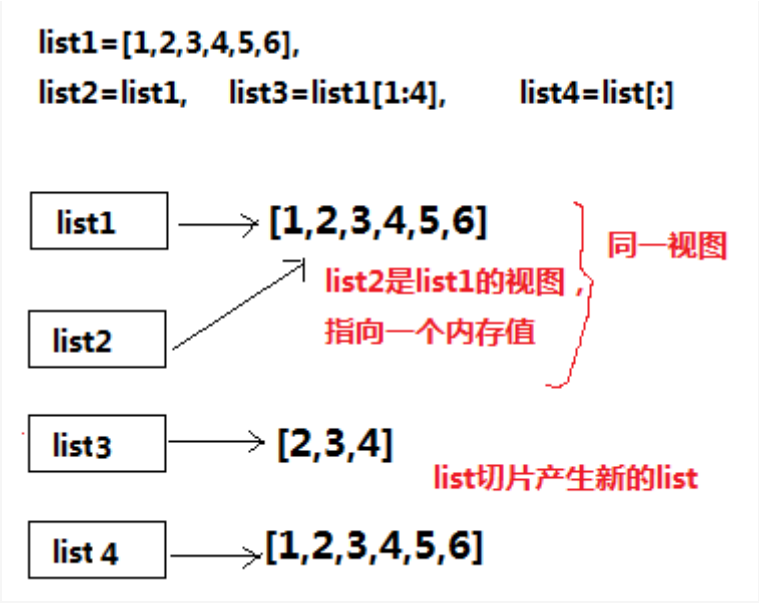
← NumPy 字节交换 NumPy 矩阵库(Matrix) →

1 篇笔记

写笔记

Python 中 list 的拷贝与 numpy 的 array 的拷贝

1.python中列表list的拷贝，会有什么需要注意的呢？



Python 变量名相当于标签名。

`list2=list1` 直接赋值，实质上指向的是同一个内存值。任意一个变量 list1（或list2）发生改变，都会影响另一个 list2(或list1)。

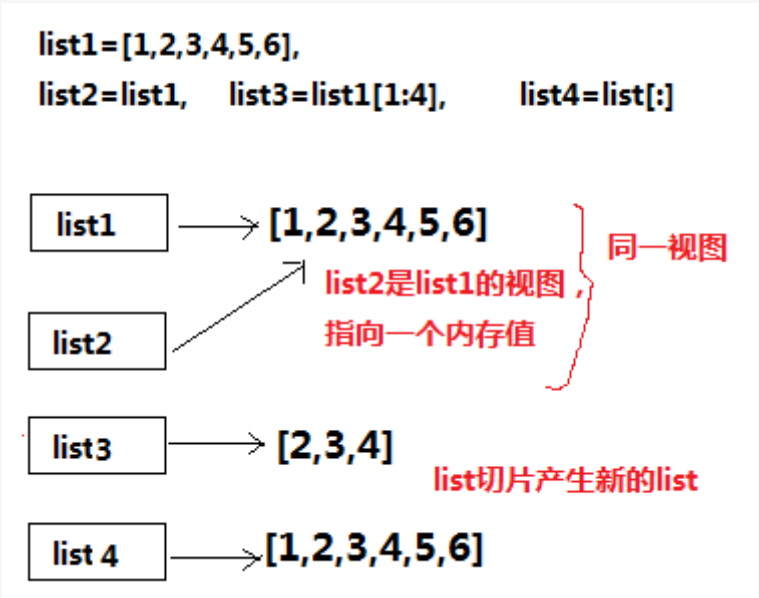
例如：

```
>>> list1=[1,2,3,4,5,6]
>>> list2=list1
>>> list1[2]=88
>>> list1
[1, 2, 88, 4, 5, 6]
>>> list2
[1, 2, 88, 4, 5, 6]
```

而 list3 和 list4 是通过切片对 list1 的复制操作，分别指向了新的值。任意改变 list3 或 list4 的值，不会影响其他。

2.要使用 ndarray 类型的数组，需要 `from numpy import *` 引用工具包 numpy。

而对 ndarray 类型的数据进行拷贝时，跟 list 类型有一点区别。



数组切片是原始数组的视图，这意味着数据不会被复制，视图上的任何修改都会被直接反映到源数组上。

array1, array2, array3, array4 实际指向同一个内存值，任意修改其中的一个变量，其他变量值都会被修改。

若想要得到的是 ndarray 切片的一份副本而非视图，就需要显式的进行复制操作函数 copy()。

例如：

```
array5=array1.copy()      # 对原始的 array1 的复制
array6=array1[1:4].copy() # 对切片 array1[1:4] 的复制
```

那么，修改 array5 或 array6，就不会影响 array1。

BDD 5个月前 (10-19)