NumPy 广播(Broadcast) Numpy 数组操作

NumPy 迭代数组

NumPy 迭代器对象 numpy.nditer 提供了一种灵活访问一个或者多个数组元素的方式。

迭代器最基本的任务的可以完成对数组元素的访问。

接下来我们使用 arange() 函数创建一个 2X3 数组，并使用 nditer 对它进行迭代。

实例

```
import numpy as np
a = np.arange(6).reshape(2,3)
print ('原始数组是：')
print (a)
print ('\n')
print ('迭代输出元素：')
for x in np.nditer(a):
print (x, end=", " )
print ('\n')
```

输出结果为：

```
原始数组是：
[[0 1 2]
 [3 4 5]]


迭代输出元素：
0, 1, 2, 3, 4, 5,
```

以上实例不是使用标准 C 或者 Fortran 顺序，选择的顺序是和数组内存布局一致的，这样做是为了提升访问的效率，默认是行序优先（row-major order，或者说是 C-order）。

这反映了默认情况下只需访问每个元素，而无需考虑其特定顺序。我们可以通过迭代上述数组的转置来看到这一点，并与以 C 顺序访问数组转置的 copy 方式做对比，如下实例：

实例

```
import numpy as np
a = np.arange(6).reshape(2,3)
for x in np.nditer(a.T):
print (x, end=", " )
print ('\n')
for x in np.nditer(a.T.copy(order='C')):
print (x, end=", " )
print ('\n')
```

输出结果为：

```
0, 1, 2, 3, 4, 5,

0, 3, 1, 4, 2, 5,
```

从上述例子可以看出，a 和 a.T 的遍历顺序是一样的，也就是他们在内存中的存储顺序也是一样的，但是 `a.T.copy(order = 'C')` 的遍历结果是不同的，那是因为它和前两种的存储方式是不一样的，默认是按行访问。

控制遍历顺序

- `for x in np.nditer(a, order='F'):`Fortran order，即是列序优先；
- `for x in np.nditer(a.T, order='C'):`C order，即是行序优先；

实例

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print ('原始数组是: ')
print (a)
print ('\n')
print ('原始数组的转置是: ')
b = a.T
print (b)
print ('\n')
print ('以 C 风格顺序排序: ')
c = b.copy(order='C')
print (c)
for x in np.nditer(c):
print (x, end=", " )
print ('\n')
print ('以 F 风格顺序排序: ')
c = b.copy(order='F')
print (c)
for x in np.nditer(c):
print (x, end=", " )
```

输出结果为：

```
原始数组是:
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]


原始数组的转置是:
[[ 0 20 40]
 [ 5 25 45]
 [10 30 50]
 [15 35 55]]
```

```
以 C 风格顺序排序:
[[ 0 20 40]
 [ 5 25 45]
 [10 30 50]
 [15 35 55]]
0, 20, 40, 5, 25, 45, 10, 30, 50, 15, 35, 55,

以 F 风格顺序排序:
[[ 0 20 40]
 [ 5 25 45]
 [10 30 50]
 [15 35 55]]
0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55,
```

可以通过显式设置，来强制 nditer 对象使用某种顺序：

实例

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print ('原始数组是: ')
print (a)
print ('\n')
print ('以 C 风格顺序排序: ')
for x in np.nditer(a, order = 'C'):
print (x, end=", " )
print ('\n')
print ('以 F 风格顺序排序: ')
for x in np.nditer(a, order = 'F'):
print (x, end=", " )
```

输出结果为：

```
原始数组是:
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]


以 C 风格顺序排序:
0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55,

以 F 风格顺序排序:
0, 20, 40, 5, 25, 45, 10, 30, 50, 15, 35, 55,
```

修改数组中元素的值

nditer 对象有另一个可选参数 op_flags。 默认情况下，nditer 将视待迭代遍历的数组为只读对象（read-only），为了在遍历数组的同时，实现对数组元素值得修改，必须指定 read-write 或者 write-only 的模式。

实例

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print ('原始数组是: ')
print (a)
print ('\n')
for x in np.nditer(a, op_flags=['readwrite']):
x[...]=2*x
print ('修改后的数组是: ')
print (a)
```

输出结果为：

```
原始数组是:
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]


修改后的数组是:
[[  0  10  20  30]
 [ 40  50  60  70]
 [ 80  90 100 110]]
```

使用外部循环

nditer类的构造器拥有flags参数，它可以接受下列值：

参数	描述
c_index	可以跟踪 C 顺序的索引
f_index	可以跟踪 Fortran 顺序的索引
multi-index	每次迭代可以跟踪一种索引类型
external_loop	给出的值是具有多个值的一维数组，而不是零维数组

在下面的实例中，迭代器遍历对应于每列，并组合为一维数组。

实例

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print ('原始数组是: ')
print (a)
```

```
print ('\n')
print ('修改后的数组是: ')
for x in np.nditer(a, flags = ['external_loop'], order = 'F'):
print (x, end=", " )
```

输出结果为：

```
原始数组是:
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]


修改后的数组是:
[ 0 20 40], [ 5 25 45], [10 30 50], [15 35 55],
```

广播迭代

如果两个数组是可广播的，nditer 组合对象能够同时迭代它们。 假设数组 a 的维度为 3X4，数组 b 的维度为 1X4 ，则使用以下迭代器（数组 b 被广播到 a 的大小）。

实例

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print ('第一个数组为: ')
print (a)
print ('\n')
print ('第二个数组为: ')
b = np.array([1, 2, 3, 4], dtype = int)
print (b)
print ('\n')
print ('修改后的数组为: ')
for x,y in np.nditer([a,b]):
print ("%d:%d" % (x,y), end=", " )
```

输出结果为：

```
第一个数组为:
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]


第二个数组为:
[1 2 3 4]
```

修改后的数组为：

```
0:1, 5:2, 10:3, 15:4, 20:1, 25:2, 30:3, 35:4, 40:1, 45:2, 50:3, 55:4,
```

NumPy 广播(Broadcast) Numpy 数组操作

点我分享笔记