索引

杨资璋（翻译）

ndarrays可以使用标准的 Python 语法x[obj]进行索引，其中x是数组obj是选择。共有三种可用的索引方式：域访问、基本切片和高级索引。具体的方式取决于obj。

Note 在 Python 中x[(exp1, exp2, ..., expN)]和x[exp1, exp2, ..., expN]是等价的；后者只是前者的句法优化。

1 基本切片和索引

基本的切片将 Python 的基本切片概念扩展到了 N 维。当obj是切片对象（通过括号中的start:stop:step构造）、整数或者切片和整数的元组时，基本切片将会被触发。省略号和newaxis对象也可以穿插其中。

使用 N 个整数进行索引是最简单的情况，它会放回表示对应元素的数组标量。和 Python 一样，所有的索引都是从 0 开始的：对于第 i 个索引 n_i，合法的范围是 $0 \le n_i < d_i$，其中 d_i 是数组形状的第 i 个元素。负索引通过从末尾开始计数被穿插其中（也就是，当 $n_i < 0$ 时，它表示 $n_i + d_i$）。

所有通过基本切片生成的数组都是原数组的视图。

Note NumPy 切片会创造一个视图而不是如 Python 自建序列例如 string, tuple 和 list 创建一个备份。对于一个大数组，当从中提取一小部分之后大数组就变得无用的情况，尤其要注意。因为这一小部分提取的数组包含对大数组的引用，而大数组的内存直到所有从它衍生出的数组都被垃圾收集后才会被释放。在这种情况下，显式使用copy()是推荐的。

序列切片的标准规则适用于每一个维度的基本切片（包括使用步长索引）。一些需要记住的有用概念包括：

- 基本的切片语法`i:j:k`其中 i 是起始索引，j 是停止索引，k 是步长（$k \neq 0$）。
- 负数 i 和 j 会被作为 $n+i$ 和 $n+j$ 进行穿插，其中 n 是对应维度的元素数量。负数 k 则会使得朝着更小的索引进行迈步。
- 假设 n 为被切片维度的元素数量。那么，如果 i 没有被给出则若 $k > 0$，i 为 0，若 $k < 0$ 则 $i = n - 1$。如果 j 没有给出则类似。注意，`::`和`:`是一样的，都表示选择对应维度的所有切片。
- 如果选择元组中对象的数量小于 N，这时则假设后续的所有维度都使用`:`。

```
>>> x = np.array([[[1],[2],[3]], [[4],[5],[6]]])
>>> x.shape
(2, 3, 1)
>>> x[1:2]
array([[[4],
        [5],
        [6]]])

```

- 省略号将扩展为选择元组为索引所有维度所需的`:`对象数量。在大多数情况下，这意味着扩展后的选择元组长度为`x.ndim`。仅能存在一个单独的省略号表示。

```
>>> x[...,0]
array([[1, 2, 3],
       [4, 5, 6]])

```

- 选择元组中的每个`newaxis`对象起到为结果选择元组扩展一个单位长度的维度的作用。加入维度是`newaxis`在选择元组中的位置。

```
>>> x[:,np.newaxis,:,:].shape
(2, 1, 3, 1)

```

- 一个整数 i，返回与`i:i+1`相同的值，除了返回对象的维度减少 1。特别的，一个第 p 个元素为整数的选择元组（其余皆为`:`）放回对应维度为 $N-1$ 的子数组。当 $N = 1$ 时，这时返回的元素是一个数组标量。

- 如果选择元组除了第 p 个元素为切片对象`i:j:k`,其余元素皆是`:`,这时返回的数组维度为 N,这个数组通过将使用整数 $i, i+k, \ldots, i+(m-1)k < j$ 索引返回的子数组拼接而成。
- 使用有着不止一个非`:`条目的切片元组进行切片时，基本切片将会类似重复应用使用单个非`:`条目进行切片，其中非`:`条目被以此使用（将所有其他的非`:`条目替换为`:`）。因此在基本切片中，`x[ind1,..., ind2, :]`等价于`x[ind1][..., ind2, :]`。

  Warning 这一条对于高级索引是不适用的。

- 你可以使用切片来设置数组中的值，但是（与列表不同）你不能增长数组。在`x[obj] = value`中，设置的 value 的形状一定要与`x[obj]`的形状相同（或广播后相同）。

2 高级索引

当选择对象`obj`是非元组序列对象、整数或 bool 的`ndarray`或者一个有着至少一个序列对象或整数或 bool 的`ndarray`的元组时，高级索引将会被触发。共有两种高级索引类型：整数和 Boolean。

高级索引总是返回数据的*备份*（与返回视图的基本索引相反）。

Warning 高级索引的定义意味着`x[(1, 2, 3), ]`与`x[(1,2,3)]`是完全不同的。后者与`x[1, 2, 3]`等价，将会触发基本索引，而后者则会触发高级索引。确保你明白为什么这会发生。同时注意`x[[1, 2, 3]]`也会触发高级索引，然而由于前面提到的丢弃的数值兼容性，`x[[1,2,slice(None)]]`将会触发基本索引。

2.1 整数数组索引

整数数组索引允许根据 N 维索引选择数组中的任意项。每个整数数组表示该维度的多个索引。

2.1.1 纯整数数组索引

当索引由与被索引数组的维度一样多的整数数组组成时，索引是直接的，但与切片不同。

高级索引总是被广播并作为一个被迭代：

```
    result[i_1, ..., i_M] == x[ind_1[i_1, ..., i_M], ind_2[i_1, ..., i_M],
                               ..., ind_N[i_1, ..., i_M]]
```

注意结果的形状与广播的索引数组形状（`ind_1, ..., ind_N`）是相同的。

为了实现和上述基本切片相似的行为，可以使用广播。函数`ix_`可以帮助这个广播。这最好通过例子理解。

Example 使用高级索引，角元素应该从一个 4×3 的数组中被选择出来。因此所有列在`[0, 2]`同时行在`[0, 3]`的元素应该被选出来。为了使用高级索引，需要显式选择每一个元素。使用之前解释的方法，可以使用如下代码：

```
>>> x = np.array([[ 0,  1,  2],
...               [ 3,  4,  5],
...               [ 6,  7,  8],
...               [ 9, 10, 11]])
>>> rows = np.array([[0, 0],
                     [3, 3],], dtype=np.intp)
>>> columns = np.array([[0, 2],
                        [0, 2], dtype=np.intp])
>>> x[rows, columns]
array([[0, 2],
       [9, 11]])
```

然而，由于上面的索引数组仅仅是在重复它们自己，所有可以使用广播 () 来简化：

```
>>> rows = np.array([0, 3], dtype=np.intp)
>>> columns = np.array([0, 2], dtype=np.intp)
>>> rows[:, np.newaxis]
array([[0],
       [3]])
>>> x[rows[:, np.newaxis], columns]
array([[ 0,  2],
       [ 9, 11]])
```

这个广播也可以通过`ix_`函数实现：

```
>>> x[np.ix_(rows, columns)]
array([[ 0,  2],
       [ 9, 11]])
```

注意在之前的例子中没有使用`np.ix_`，只有对角线元素被选择了。这个不同是关于使用多个高级索引进行索引最重要的事情。

2.1.2 将高级和基本索引结合

当索引中存在至少一个切片（:，省略号（...）或（newaxis）时，这时行为可能变得更加复杂。它就像为每一个高级索引元素拼接它们的索引结果。

在最简单的例子中，仅存在一个高级索引。一个单独的高级索引可以替换一个切片并得到相同的结果，然而，它是一个备份并可能有不同内内存布局。当可能时，切片是更好的。

Example

```
>>> x[1:2, 1:3]
array([[4, 5]])
>>> x[1:2, [1, 2]]
array([[4, 5]])
```

理解这种情况最简单的方式也许是考虑结果的形状。索引操作共有两部分，通过基本索引定义的子空间和通过高级索引定义的子空间。需要搞清楚索引结合的两种情况：

- 高级索引被切片、省略号或者newaxis分开。例如x[arr1, :, arr2]
- 所有的高级索引彼此紧贴。例如x[..., arr1, arr2, :]，但x[arr1, :, 1]则不属于这种情况，在这里1被视为高级索引

在第一种情况，高级索引操作得到的维度首先出现在结果数组中，然后是子空间维度。在第二种情况，来自高级索引操作的维度被插入到与它们在初始数组中相同的结果数组中的位置（后一种逻辑使简单高级索引的行为就像切片一样）。

Example 假设x.shape为 (10, 20, 30)，ind是形状为 (2, 3, 4) 的 intp 数组，那么由于 (20,) 的子空间被替换为了 (2, 3, 4) 的广播后的索引子空间，所以result = x[..., ind, :]的形状为 (10, 2, 3, 4, 30)。如果我们让i，j，k遍历形状为 (2, 3, 4) 的子空间，这时result[..., i, j, k, :] = x[..., ind[i, j, k], :]。这个例子产生与x.take(ind, axis=-2)相同的结果。

Example 假设x.shape为 (10, 20, 30, 40, 50)，ind_1和ind_2可以广播为 (2, 3, 4)。这时由于形状为 (20, 30) 的子空间被替换为来自索引的 (2, 3, 4) 子空间，所以x[:, ind_1, ind_2]的形状为 (10, 2, 3, 4, 40, 50)。然而，由于在索引

子空间中没有明确的地方可以删除，因此它被添加到了开头，所以x[:, ind_1, :, ind_2]的形状为 (2, 3, 4, 10, 30, 50)。使用.transpose()来将子空间移动到任意需要的位置一直是可行的。注意这个例子无法通过take复现。

2.2 boolean 数组索引

当obj是例如可能从比较运算符返回的布尔类型的数组对象时，就会发生这种高级索引。单个布尔索引数组实际上与x[obj.nonzero()]相同，其中，如上所述，obj.nonzero()返回整数索引数组的元组（长度为obj.ndim），显示obj的True元素。但是，当obj.shape == x.shape时，它的速度更快。

如果obj.ndim == x.ndim，则x[obj]返回一个一维数组，其中填充了与obj的True值对应的x元素。搜索顺序将是行优先，C 样式。如果obj在x边界之外的条目处具有True值，则将引发索引错误。如果obj小于x，则使用False进行填充。

Example 一个常见的用例是过滤所需的元素值。例如，您可能希望从数组中选择所有不是 NaN 的条目：

```
>>> x = np.array([[1., 2.], [np.nan, 3.], [np.nan, np.nan]])
>>> x[~np.isnan(x)]
array([1., 2., 3.])
```

或者希望为所有负元素添加一个常量：

```
>>> x = np.array([1., -1., -2., 3])
>>> x[x < 0] += 20
>>> x
array([  1.,  19.,  18.,   3.])
```

通常，如果索引包含布尔数组，结果将与将obj.nonzero()插入相同位置并使用上述整数数组索引机制相同。x[ind_1, boolean_array, ind_2]等价于x[(ind_1,) + boolean_array.nonzero() + (ind_2,)]。

如果只有一个布尔数组而没有整数索引数组，这很简单。只需要注意确保布尔索引具有与它应该使用的维度一样多的维度。

Example 从数组中，选择总和小于或等于 2 的所有行：

```
>>> x = np.array([[0, 1], [1, 1], [2, 2]])
>>> rowsum = x.sum(-1)
>>> x[rowsum <= 2, :]
array([[0, 1],
       [1, 1]])
```

使用obj.nonzero()类比可以最好地理解组合多个布尔索引数组或布尔与整数索引数组。函数ix_也支持布尔数组并且可以正常工作。

Example 使用布尔索引选择加起来为偶数的所有行。同时，应使用高级整数索引选择列0和2。使用ix_函数可以通过以下方式完成：

```
>>> x = np.array([[ 0,  1,  2],
[ 3,  4,  5],
[ 6,  7,  8],
[ 9, 10, 11]])
>>> rows = (x.sum(-1) % 2) == 0
>>> rows
array([False,  True, False,  True])
>>> columns = [0, 2]
>>> x[np.ix_(rows, columns)]
array([[ 3,  5],
[ 9, 11]])
```

如果没有np.ix_调用，只会选择对角线元素。或者没有np.ix_（比较整数数组示例）：

```
>>> rows = rows.nonzero()[0]
>>> x[rows[:, np.newaxis], columns]
array([[ 3,  5],
       [ 9, 11]])
```