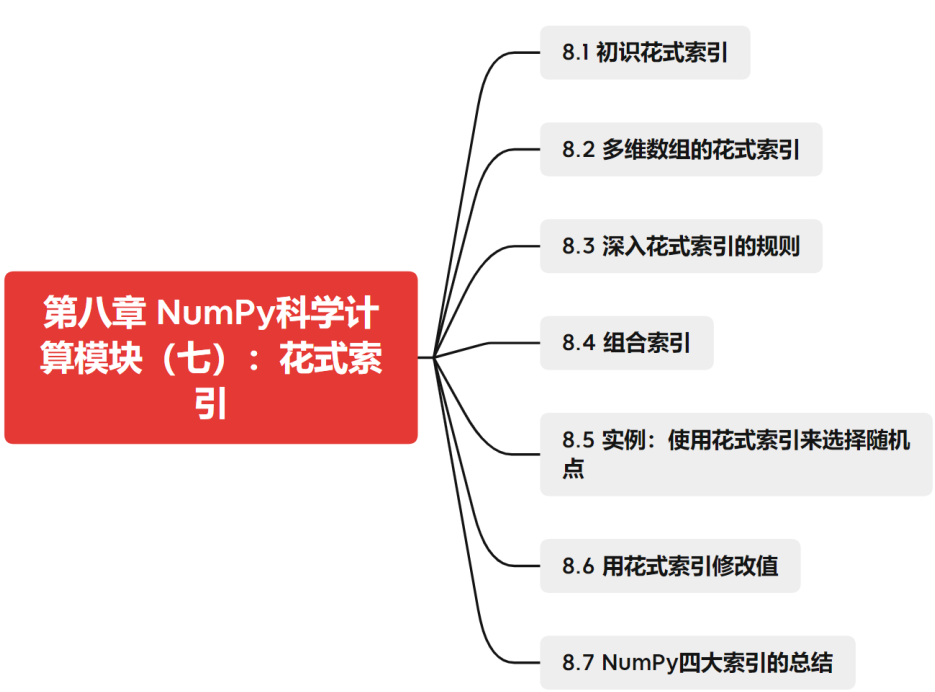
第八章 NumPy科学计算模块（七）：花式索引

思维导图



8.1 初识花式索引

任务：NumPy数组的四大索引

> 知识点：
>
> - 单索引：根据**单个索引值**，返回**单个元素**。例如，`x[0,2]`。
> - 切片索引：使用`[start:stop:step]`切片规则来索引元素，返回**一组元素**。例如，`x[:2,1:]`。
> - 掩码索引：使用逻辑运算得到布尔数组作为**掩码**，返回掩码为`True`的**一维数组元素**。例如，`x[x > 0]`。
> - 花式索引：fancy indexing，向数据数组传递**索引数组**以便批量获得**多个数组元素**。(本节学习重点)

> 多维数组的索引：**单索引、切片索引、花式索引的多维数组索引方式。**
>
> - 按**一维数组索引**的方式，**对每个维度分别使用索引**。
> - 例如，`x[0,1]`，每个维度分别使用索引，第1维索引为`0`；第2维索引为`1`；然后NumPy使用各维度上的索引来**定位元素**，获得`(0,1)`坐标位置上的元素；不同维度的索引，则通过`,`相隔。

> 多维数组的索引：**掩码索引的多维数组索引方式。**
>
> - 一维索引和多维索引一样，只需要输入整个掩码数组，并不需要为每个维度分别输入隐码数组。`x[x>1]`。

```
import numpy as np
rand = np.random.RandomState(42)
```

```
# 创建二维数组
x = rand.randint(10, size=(5,5))
print(x)

print('--单索引--')
print(x[0,1])

# 第1维0~1;第2维1~end
print('--切片索引--')
print(x[:2,1:])


#不需要为每个维度单独索引
print('--掩码索引--')
# 构造掩码
mask = x<3
print(mask)
print(x[mask])
```

```
# 不需要为每个维度单独索引
print('--一维掩码索引--')
x = rand.randint(10, size=(5,))
print(x)

# 构造掩码
mask = x<3
print(mask)
print(x[mask])

print('--多维掩码索引--')
x = rand.randint(10, size=(5,3))
print(x)

# 构造掩码
mask = x<3
print(mask)
print(x[mask])
```

任务：初识花式索引

> 知识点：花式索引直观认识
>
> - **切片的瓶颈**：虽然切片可以批量获得一组数组元素，但是切片规则获得的元素必须要有一定的规律`[start:stop:step]`。对于无规则的批量索引，切片则无法实现。例如，一维数组，按顺序获得`1`,`2`,`4`,`0`位置的元素。
> - 为解决上述问题，NumPy引入了花式索引机制。
> - 将**单索引**推广至**索引数组**，从而实现批量索引数组的元素。
> - 相对于单索引，花式索引使用一个**整型数组**作为**索引数组**，来获得多个数组元素。

下面通过一个例子，如何将将**单索引**推广至**花式索引**，来方便地实现数组元素的批量获取。

```
x = rand.randint(100, size=10)
print(x)
```

> **方法一：单索引。**依次使用单索引来获得元素。

```
res = [x[3], x[7], x[2]]
print(res)
```

> **方法二：花式索引。**构造索引数组，然后再将该数组作为**整体**传递给数组。

```
ind = [3, 7, 4]
res = x[ind]
print(res)
```

任务：掌握索引相关的术语

> 知识点：(重点) **索引相关的术语。**
>
> - 数据数组：用于被索引的数组。
> - 结果数组：索引之后得到的数组。
> - 索引数组：该数组的元素是索引值，必须是整形，用于索引数据数组。注意：索引值不能超过数据数组最大的索引，否则将会出错。
> - 数组的总维度：数组的`.ndim`属性。
> - 数组的某个维度：多维数组，需要指明索引是针对第几维或哪个`axis`。
> - 数组的形状：数组的`.shape`属性。

下面结合一个一维数组的花式索引例子，来了解术语。

```
print('--x称为数据数组--')
x = rand.randint(10, size=(5,))
print(x)

print('--数组维度和形状--')
print(x.ndim, x.shape)

print('--ind称为索引数组--')
ind = [1,2,4,0]
print(ind)

print('--x_new称为结果数组--')
print('--x[ind]为x指定第1维的索引为ind--')
x_new = x[ind]
print(x_new)
```

```
x = rand.randint(100, size=10)
print(x)

ind = np.array([[3, 7], [4, 5]])
res = x[ind]
print(res)
# 结果数组shape与索引数组一致
print('数据数组shape：', x.shape, x.ndim)
print('结果数组shape：', res.shape, res.ndim)
print('索引数组shape：', ind.shape, ind.ndim)
```

8.2 多维数组的花式索引

任务：多维数据数组的单索引和切片

要理解多维数组的花式索引，先回顾多维数组的单索引和切片。

> 知识点：(回顾) 多维数组的单索引和切片。以三维数组`x`为例。
>
> - 多维数组的元素访问：在`[]`中，以`,`相隔不同维度的索引值，来获得数组元素。
> - **单索引**：获得一个元素。`x[i,j,k]`，其中`i`，`j`和`k`为索引值分别对应`axis=0`、`axis=1`和`axis=2`的坐标值。例如，`x[2,1,3]`。
> - **切片**：获得一组元素。而单索引本质上是切片的一个特例。例如，`x[2:,:,:3]`。
> - **单索引和切片**的组合：有些`axis`使用单索引，有些`axis`使用切片。例如，`x[2,1:,:3]`。

```
# 创建一个三维数组
x = np.arange(36).reshape((3, 3, 4))
print('数据数组：', x.ndim, x.shape)
print(x)

# 单索引
print('---单索引---')
print(x[2,1,3])

# 切片
print('---切片---')
print(x[2:,1:,:3])

# 单索引和切片的组合
print('---单索引和切片的组合---')
# 中间的：不能被缺省
print(x[2,:,:3])
```

任务：多维数组的索引缺省值

> 知识点(规则)：在NumPy中，**多维数组的索引是否可以被缺省?**
>
> - 规则：**允许**数组的**高维**`axis`**索引**被缺省(被缺省的索引之后，**没有显式索引值**)。
>   - 被缺省的`axis`索引值默认使用切片`:`，即返回该`axis`的所有元素。
>   - 例如，三维数组`x[2,2]`等价于`x[2,2,:]`。四维数组`x[2,2]`等价于`x[2,2,:,:]`。
> - 规则：**不允许**数组的**低维**`axis`**索引**被缺省(被缺省的索引之后，**还有显式索引值**)。

    - 例如，第1维和第3维有显式的索引，第2维的索引就不能被缺省，即两个`,`之间不能为空。`x[2,:,2]`不能缺省为`x[2,,2]`。

```
# 缺省索引
print('---缺省索引---')
# 高维索引：第3维的索引可以被缺省
print(x[2,2])

# 低维索引不能被缺省
# 出错的缺省索引，两个,之间不能为空
print('---出错的缺省索引---')
#print(x[2,,2])
```

任务：掌握结果数组总维度的演算规则

> 知识点(规则)：结果数组的总维度`.ndim`的换算公式。
>
> - 结果数组`.ndim` = 数据数组`.ndim` - 使用单索引的`axis`个数。
> - **切片对结果数组的维度没影响**，哪怕切片返回的元素只有一个或者为空。

```
# 创建一个三维数组
x = np.arange(36).reshape((3, 3, 4))
print('数据数组：', x.ndim, x.shape)

print('---结果数组.ndim = 数据数组.ndim - 使用单索引的axis个数---')
x_new = x[2,1,3]
# x_new的.ndim = 数据数组.ndim - 3个axis使用单索引 = 3 - 3 =0
print(x_new)
print('x[2,1,3]：', x_new.ndim, x_new.shape)
```

```
x_new = x[2,:,3]
# x_new的.ndim = 数据数组.ndim - 2个axis使用单索引 = 3 - 2 =1
print(x_new)
print('x[2,:,3]：', x_new.ndim, x_new.shape)
```

```
x_new = x[2,:,:]
# 自行分析
print('x[2,:,3]：', x_new.ndim, x_new.shape)
```

> 知识点：如果某个维度**切片返回的元素是空**，对结果数组`.ndim`并未有影响。

```
print('--切片对结果数组.ndim没影响--')
# 第3维切片为1个元素
x_new = x[2:,:,3:]
print(x_new)
# 自行分析
print('x[2,:,3]: ', x_new.ndim, x_new.shape)

#坑：第3维切片为0个元素，结果数组为空数组[]
# 根据NumPy规则，这数组还是有维度。
x_new = x[2:,:,4:]
print(x_new)
# 自行分析
print('x[2,:,3]: ', x_new.ndim, x_new.shape)
```

任务：多维数组的花式索引

之前回顾了多维数组的单索引和切片，接下来讲下多维数组的花式索引的使用。

> 知识点：
>
> - 本质上，花式索引就是将**多个单索引组成为一个索引数组(序列)**。通过该索引数组，批量获得元素。
> - 多维数组的单索引：必须为每个`axis`指定索引值。例如：`x[1,2,1]`。
> - 多维数组的花式索引：类似，也必须要对每个`axis`指定索引数组。例如：`x[ind1,ind2,ind3]`。
> - 规则：如果数据数组总维度为`.ndim`，那就需要`.ndim`个之对应的索引数组。

```
# 创建一个二维数组
x = np.random.randint(10, size=(3, 4))
print('数据数组: ', x.ndim, x.shape)
print(x)

# 使用单索引
print(x[1, 2])

# 创建x.ndim=2个一维索引数组
axis0 = np.array([0, 1, 2, 1])
axis1 = np.array([2, 1, 1, 1])

# 将一维数组分别作为数据数组x的索引，并使用`,`相隔。
new_x = x[axis0, axis1]

# 结果数组为一维，它的shape为索引数组的shape
print('数据数组: ', new_x.ndim, new_x.shape)
print(new_x)
```

代码解析：

- 二维数组`x`的`.ndim`为2。如果使用单索引来获得数组的元素，就需要给定数组的二维索引值(坐标)，这样才能唯一的定位到指定的元素。
- 类似于单索引，花式索引也需要指定两个`.shape`一致的索引数组，作为花式索引的二维索引。

- 索引数组`axis0 = np.array([0, 1, 2, 0])`为第1维提供索引值，`axis1 = np.array([2, 1, 1, 1])`为第2维提供索引值。
- `new_x = x[axis0, axis1]`，将索引数组`axis0`和`axis1`作为两个维度的索引值，来获得结果数组`new_x`的元素。例如，`new_x`的第1个元素为二维索引`(axis0[0],axis1[0])`对应的元素：`x[0,2]`。
- 根据规则，结果数组的`.shape`等于索引数组的`.shape`。这里的索引数组`axis0`和`axis1`为一维数组，那么结果数组也是一维数组。

8.3 深入花式索引的规则

任务：索引的越界异常

> 知识点：
>
> - 无论是单索引还是花式索引，它们的索引值都不能超过数据数组`.shape`的上界。
> - 否则就会数组越界异常`IndexError`。

```
# 花式索引越界异常
# 创建一个二维数组
x = np.random.randint(10, size=(3, 4))
print(x.shape)
# x.shape为(3, 4)，第1维的元素索引上界为3
# 第1维的第4个索引值10，超过数据数组的上界，引起越界异常。
axis0 = np.array([0, 1, 2, 10])
axis1 = np.array([2, 1, 1, 1])

# 将一维数组分别作为数据数组x的索引，并使用`,`相隔。
new_x = x[axis0, axis1]

# 结果数组为一维，它的shape为索引数组的shape
print('数据数组：', new_x.ndim, new_x.shape)
print(new_x)
```

下面再看个三维数组的一维花式索引的例子。根据规则可知，三维数组`.ndim`为3，那么就需要有3个`.shape`一样的一维索引数组。

```
# 创建一个三维数组
x = np.arange(36).reshape((3, 3, 4))
print('数据数组：', x.ndim, x.shape)
print(x)

# 创建x.ndim=3个一维索引数组
axis0 = np.array([0, 1, 2])
axis1 = np.array([2, 1, 1])
axis2 = np.array([2, 1, 3])

# 将一维数组分别作为数据数组x的索引，并使用,相隔。
new_x = x[axis0, axis1, axis2]

# 结果数组为一维，它的shape为索引数组的shape
print('数据数组：', new_x.ndim, new_x.shape)
print(new_x)
```

任务：深入花式索引.shape规则

> 知识点：
>
> - 当花式索引的索引数组为**多维**时，**结果数组**是**索引数组对原数据数组的元素索引**。
> - 例如：假设数据数组为一维数组，索引数组为`[[3, 7],[4, 5]]`，那么结果数组是二维数组，元素为`[[x[3], x[7]], [x[4], x[5]]]`。

> 知识点(规则)：花式索引`.shape`规则
>
> - 规则：结果数组与索引数组的`.shape`**一致**。
> - 规则：结果数组与原数据数组的`.shape`**无关**。
> - 例如：一维数据数组`.shape: (10,)`；二维索引数组的`.shape: (100,20)`；结果数组为二维，它的`.shape: (100,20)`。

```
x = rand.randint(100, size=10)
print(x)

ind = np.array([[3, 7], [4, 5]])
res = x[ind]
print(res)
# 结果数组shape与索引数组一致
print('数据数组shape：', x.shape, x.ndim)
print('结果数组shape：', res.shape, res.ndim)
print('索引数组shape：', ind.shape, ind.ndim)
```

任务：掌握维度规则

> 知识点(规则)：**数据数组、花式索引数组、结果数组的维度关系。**
>
> - **一维数组的一维花式索引**：数据数组为一维数组，花式索引数组为一维数组。**结果数组为一维数组。**
> - **一维数组的多维花式索引**：数据数组为一维数组，花式索引数组为多维数组。**结果数组为多维数组。**
> - **多维数组的一维花式索引**：数据数组为多维数组，花式索引数组为一维数组。**结果数组为一维数组。**
> - **多维数组的多维花式索引**：数据数组为多维数组，花式索引数组为多维数组。**结果数组为多维数组。**

```
# 一维数组的一维花式索引，结果数组为一维数组
x = rand.randint(100, size=10)
print(x)

ind = np.array([3, 7, 4, 5])
res = x[ind]
print(res)
# 结果数组shape与索引数组一致
print('数据数组shape：', x.shape, x.ndim)
print('结果数组shape：', res.shape, res.ndim)
print('索引数组shape：', ind.shape, ind.ndim)
```

```
# 多维数组的一维花式索引，结果数组为一维数组
x = rand.randint(100, size=(6,6))
print(x)

ind = np.array([3, 1, 4, 5])
# 二维数组的单索引x[i,j]，需要分别对两个维度给出索引值i和j。
# 类似的，二维数组索引，需要用两个索引数组，来指定索引值。
# 详细请看后续章节的解析。
res = x[ind,ind]
print(res)
# 结果数组shape与索引数组一致
print('数据数组shape：', x.shape, x.ndim)
print('结果数组shape：', res.shape, res.ndim)
print('索引数组shape：', ind.shape, ind.ndim)
```

任务：花式索引的广播规则

> 知识点：多维数组的索引，在每个维度上的索引必须要`.shape`匹配，这样才能唯一地确定被索引元素。那么，当索引数组`.shape`不匹配时会怎么办？
>
> - 类似于通用函数，NumPy会尝试使用**广播**技术来解决这一问题。使得每个`axis`上的索引数组`.shape`都一致。
> - 只有**满足广播两大规则**的索引数组，才能被广播。

> 知识点(规则)：花式索引广播后的结果数组`.shape`。
>
> - 规则：结果数组的`.shape`是**广播后的索引数组**`.shape`，而不是原索引数组`.shape`。

```
import numpy as np
rand = np.random.RandomState(42)

print('---数据数组---')
x = rand.randint(100, size=(6, 6))
print(x)

print('---索引数组---')
ind1 = np.array([1, 3, 2])
ind2 = np.array([3, 2, 5])[:, np.newaxis]
print('ind1:', ind1.ndim, ind1.shape)
print('ind2:', ind2.ndim, ind2.shape)
```

```
print('---结果数组---')
x_new = x[ind1, ind2]
print('x_new:', x_new.ndim, x_new.shape)
print(x_new)
```

广播分析：

- `ind1`的`.shape`为`(3,)`，`ind2`的`.shape`为`(3,1)`。
- 根据广播规则1，先给小维度的`ind1`左边补1，此时`ind1`的`.shape`为`(1,3)`。
- 两个数组的维度依然不匹配，满足广播规则2，两个数组不匹配的维度形状都为1，分别将`ind1`和`ind2`广播为`(3,3)`。
- 两个数组`.shape`匹配，完成一一对应位置的索引。

8.4 组合索引

> 知识点：
>
> - 四大索引技术允许组合使用，不同维度可以使用不同的索引技术，实现更强大的索引操作。
> - 当`.shape`不匹配时，使用了**广播技术**。

任务：单索引和花式索引的组合

> 知识点：单索引和花式索引的组合。
>
> - 标量(单索引)和数组(花式索引)的广播。
> - 使用广播规则1，对单索引广播为花式索引一样的`.shape`。

```
print('---数据数组---')
x = np.arange(12).reshape((3, 4))
print(x)

print('---索引数组---')
ind = np.array([2, 0, 1])
print('ind:', ind.ndim, ind.shape)

print('---结果数组---')
# 单索引+花式索引
x_new = x[2,ind]
print('x_new:', x_new.ndim, x_new.shape)
print(x_new)
```

> 单索引和花式索引的广播解析：
>
> - 第1维单索引：`2`；第2维花式索引：`[2, 0, 1]`。
> - 使用广播规则1，将第1维广播为`[2, 2, 2]`。
> - 组合索引，元素的索引为：`(2,2)`、`(2,0)`、`(2,1)`。

任务：切片索引和花式索引组合

> 知识点：切片索引和花式索引的组合。
>
> - 一维数组(切片索引)和数组(花式索引)的广播。
> - 使用条件：
>   - 不允许不同`.shape`的花式索引同时存在；
>   - 但允许不同`.shape`的切片索引同时存在。

> 知识点：索引组合的`.shape`不匹配问题。
>
> - 然而，大部分的索引组合，在各维度上的`.shape`都不能满足**通用函数的广播规则**。
> - **通用函数的广播规则的缺陷**：例如，第1维为切片索引：`[1,2]`；第2维为花式索引：`[1,0,1,2]`。如果直接使用通用函数的广播规则，将会出错。因为，**不能满足广播规则2**：第1维`.shape:(2,)`和第1维`.shape:(4,)`，如下面程序。

```
# 通用函数广播规则
ind1 = np.array([1,2])
ind2 = np.array([2, 0, 1, 2])
ind1 + ind2
```

为解决上述问题，NumPy针对索引组合，对广播机制进行了修正。

> 知识点(规则)：
>
> - 切片的索引数组一定是一维；花式索引的索引数组通常是一维或多维。
> - 说明：多维数组，`axis=n`表示当前第n个维度，称比`axis=n`小的维度为**低维**，比它大的为**高维**。
> - 组合索引结果数组的形状：使用广播，将**低维到高维的各索引数组进行依次的直和**。比如三维数组，3个索引数组的`.shape`从低维到高维依次为`(3,)`、`(2,3)`、`(4,)`，那么最后组合索引的结果数组为`(3,2,3,4)`。

```
print('---数据数组---')
x = np.random.randint(10,size=(3, 4, 5))
print('x:', x.ndim, x.shape)

print('---索引数组---')
ind1 = np.array([2, 0, 1, 1, 0, 2]).reshape(2, 3)
print('ind2:', ind2.ndim, ind2.shape)

print('---结果数组---')
x_new = x[2:, ind2, :4]
print('x_new:', x_new.ndim, x_new.shape)
```

任务：深入组合索引的广播机制

NumPy底层通过广播机制实现了**低维到高维的各索引数组进行依次的直和**。下面机制看是复杂，其实很简单，对着例子理解规则。

> 知识点：从广播机制角度深入理解
>
> - 广播规则1的修正 (对着例子理解规则)：

1. 计算所有索引数组的总维度`D`。
2. 从低维到高维，对于当前索引数组，计算**比它低维的索引数组的总维度**`D1`，对它**左补**`D1`**个1**。
3. 然后再通过**右补1**的方式，使得当前索引数组的维度等于总维度`D`。

- 广播规则2保持不变。
- 该规则适合**切片索引**和**组合索引**，不适合单纯的花式索引。

例如，三维数组的组合索引，第1维为**切片**索引为`[1,2]`；第2维为**切片**索引为`[1,2]`；第3维为**花式**索引为`[1,0,1,2]`，总维度为1+1+1=3。

- 第1维广播后`(2,1,1)`：第1维索引数组的`.shape`为`(2,)`，将它右补2个1，广播为`(2,1,1)`。
- 第1维广播后`(1,2,1)`：第2维索引数组的`.shape`为`(2,)`，低维的索引数组总维度为1 (比它低维的有1个一维数组)，将它左补1个1，右补1个1，广播为`(1,2,1)`。
- 第1维广播后`(1,1,4)`：第3维索引数组的`.shape`为`(4,)`，低维的索引数组总维度为2，将它左补2个1，右补0个1，广播为`(1,1,4)`。
- 观察，本质上就是对其他维度进行**左右补1**。
- 然后，使用原规则2，对上述索引数组进行广播，最终的`.shape`为`(2,2,4)`，其实就是低维到高维的各索引数组的直和。

```
print('---数据数组---')
x = np.arange(12).reshape((3, 4))
print(x.ndim, x.shape)

print('---索引数组---')
ind = np.array([2, 0, 1, 2])
print('ind:', ind.ndim, ind.shape)

print('---结果数组---')
# 切片+花式索引
# 第1维切片：[1,2]
# 第2维花式索引：[2, 0, 1, 2]
x_new = x[1:, ind]
print('x_new:', x_new.ndim, x_new.shape)
print(x_new)
```

程序解析(切片和花式索引)：

- 第1维切片索引：`[1, 2]`；第2维花式索引：`[2, 0, 1, 2]`。
- 两个数组的`.shape`不匹配，也不满足广播规则。对于正常的通用函数，就会出错。具体运行下面代码进行测试。
- 使用广播规则1，将第1维广播为`[2, 2, 2, 2]`。
- 组合索引，元素的索引为：`(2,2)`、`(2,0)`、`(1,2)`、**(2,2)**。

任务：切片和掩码索引组合

知识点：切片和掩码索引组合。以`.shape:(4,5)`的二维数据数组`x`为例。

- **规则**：掩码数组`.shape`必需要和**匹配**对应维度的数据数组`.shape`，否则会出错。例如，如果第1维使用掩码索引，那么对应的布尔数组的`.shape`必需是`(4,)`；如果是第2维，那么必需是`(5,)`。

- 掩码索引首先需要将**布尔值译码为数组索引**，译码规则为`True`元素所对应的位置作为索引，`False`元素跳过。比如`mask = [True,False,True,False]`译码为`[1,3]`。
- 然后，类似于**切片和花式索引的组合**，该组合也需要使用广播机制。
- 结果数组`.shape`：类似于花式索引，掩码索引**译码后索引数组**和切片的**直和**。例如，`x[mask,]`
- 一般，掩码索引不和花式索引进行组合。

```
print('---数据数组---')
x = np.arange(20).reshape((4, 5))
print(x.ndim, x.shape)

print('---掩码数组---')
mask = np.array([1, 0, 1, 0], dtype=bool)
print('mask为第1维索引：', mask.shape)
print(mask)

print('---结果数组---')
# 切片+掩码索引
# 第1维掩码索引，译码为：[1,3]
# 第2维切片：[1,2,3]
x_new = x[mask, :4]
print('x_new:', x_new.ndim, x_new.shape)
print(x_new)
```

8.5 实例：使用花式索引来选择随机点

任务：随机选择数组元素

本节将使用花式索引来随机选择数组(矩阵)元素，返回数据子集。这里，使用×N×D矩阵表示数据数组，其中第1维表示样本点(N个样本)，第2维表示样本的特征(共有D维特征)。

步骤一：使用二维正态分布生成数据数组。

```
import numpy as np

# 二维正态分布的均值和方差
mean = [0, 0]
cov = [[1, 2],[2, 5]]
# 调用高维正态分布，生成100个样本点
x = np.random.multivariate_normal(mean, cov, 100)
print(x.ndim, x.shape)
```

步骤二：使用`matplotlib`可视化数据。

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn; seaborn.set()

# 绘制二维散点图
plt.scatter(x[:, 0], x[:, 1])
```

步骤三：将利用花式索引随机选取20个不重复的点。调用`np.random.choice()`函数，生成不重复的索引数组。

> 知识点：`np.random.choice(a, size, replace=False, p=None)`
>
> - 描述：参数从区间`a`中随机生成`size`个服从p分布的随机数，p的默认值为均匀分布。
> - 参数：整形`a`为表示`[0~a]`区间。
> - 参数：`size`为生成的索引数组的形状。
> - 参数：`replace=False`，代表抽样不放回(**不重复**)，`True`代表有放回抽样(**重复**)。
> - 返回：`[0~a]`的随机数组。

调用`np.random.choice`，从`[0~x.shape[0]]`区间中，生成20个不重复的索引数组。其中`x.shape[0]`为样本数。

```
indices = np.random.choice(x.shape[0], 20, replace=False)
print(indices)

# 根据随机索引数组，使用花式索引，选择20个样本子集
# axis=0 代表样本维度；axis=1代表特征维度。
# 因此，花式索引放在axis=0。
selection = x[indices, :]
print(selection.shape)
```

步骤四：可视化选中的点，并在图上用大圆圈标示出。

```
plt.scatter(x[:, 0], x[:, 1], alpha=0.3)
plt.scatter(selection[:, 0], selection[:, 1], facecolor='none', edgecolor='b',
s=200);
```

8.6 用花式索引修改值

任务：使用花式索引修改数组元素

> 知识点：
>
> - 类似于单索引，可通过花式索引来批量修改数组元素值。
> - 可以赋值为单值或相同`.shape`的数组。

```
x = np.arange(10)
print(x)

# 索引数组
ind = np.array([2, 1, 8, 4])
# 将相应的元素批量修改为99
x[ind] = 99
print(x)
```

```
x = np.arange(10)
print(x)

# 索引数组
ind = np.array([2, 1, 8, 4])
# 也可以依次对应元素批量修改
x[ind] = [1,2,3,4]
print(x)
```

> 知识点：花式索引可以使用任何数值运算，来修改数组元素。

```
x[ind] -= 10
print(x)
```

> 知识点：**重复的索引会引起一些出乎意料的结果(逻辑错误)。**

```
x = np.zeros(10)

# 索引数组为两个0(重复)索引。
ind = [0, 0]

# 赋值：4,6
x[ind] = [4, 6]
print(x)
```

观察结果，发现赋值的`4`不见了。这是因为赋值首先执行`x[0] = 4`，然后`x[0] = 6`，因此`x[0]`最终的值为6。但是，下面的例子，**将会出现逻辑异常。**

```
x = np.zeros(10)
ind = [0, 0]

x[ind] += 1
# 重复索引0位置元素两次+1，结果发现x[0]不是2
print(x)
# 索引数组
ind = [2, 3, 3, 4, 4, 4]
# 对应元素+1运算
x[ind] += 1
print(x)
```

按重复索引进行重复`+1`运算，因此`x[3]`的值应该为2，`x[4]`的值为3。实际结果并未发生多次累加，而是仅仅加1。这又是为何？下面来分析下原因：

- 从概念的角度理解，这是因为`x[ind] += 1`是`x[ind] = x[ind] + 1`的简写。使用花式索引进行运算是**一次性批量的，而不是按索引值依次逐个进行的。**
- 因此，`x[ind] + 1`中的`x[ind]`一直保持原数组元素值，并未受到改变。那么结果当然就不能发生累加了。

> 知识点：

> - 为了实现**重复索引的累加操作**，可以借助通用函数中的`at()`来实现。
> - `np.add.at(array, index_array, b)`：对`array`数组的指定位置`index_array`索引数组，执行累积加`b`运算。

```
x = np.zeros(10)
ind = [2, 3, 3, 4, 4, 4]
np.add.at(x, ind, 2)
print(x)
```

8.7 NumPy四大索引的总结

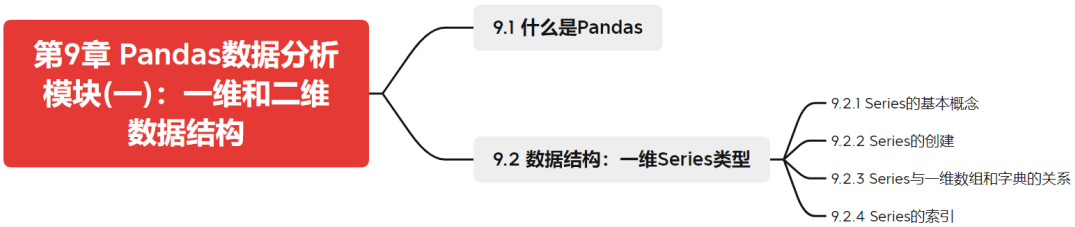
NumPy的所有索引方式都已经学习完毕，下面它们作个简单的总结。

> 知识点：
>
> - 单索引`arr[0]`
> - 切片索引`arr[:5]`
> - 掩码索引`arr[arr>0]`
> - 花式索引`arr[[0,2,4]]`

> 知识点：四种索引的比较
>
> - 切片、掩码和花式索引：都可以**批量地获得数组的多个元素。**
> - 切片：主要用来提取批量的、**有一定规律**的元素。
> - 掩码索引：主要用来筛选符合特定条件的元素。结果数组是满足`True`的**一维数组**，和原数据数组`.shape`无关
> - 花式索引：通过自定义指定索引数组(类似切片)，可以随意选取元素。结果数组和索引数组`.shape`一样，和原数据数组`.shape`无关。
> - 这四种索引方式可以两两组合，产生更多灵活的索引结果。

第9章 Pandas数据分析模块(一)：一维和二维数据结构

思维导图



9.1 什么是Pandas

任务：为何需要Pandas模块

> 知识点：
>
> - NumPy模块的核心数据结构是`ndarray`，它为Python提供了**多维数组的存储和处理方法。**
> - 多维是通过轴(`axes`)对数据展开操作。每一个维度相应地被称为一个轴（`axes`），`ndim`是轴的总量，`shape`描述了每个轴上的数据规模。
> - NumPy模块为多维数组提供高效的数值计算，例如广播、通用函数、聚合操作、四大索引等技术。

> 需求分析：
>
> - NumPy模块仅适合处理**结构化数据**，即数值类型的，不含缺失值的数据。
> - 在现实应用中，存在大量的**非结构化数据**，比如时间序列、带标签、含有缺失值等数据；此外，需要对**表格类型**数据进行分组、数据透视等加工处理。对于这类问题，NumPy模块并不适合处理。

为了解决上述需求，Pandas模块在NumPy模块的基础上，专门针对**非结构化类型数据**提供高效的数据处理。Pandas是一个强大的分析结构化数据的工具集；它继承了NumPy的优势(提供高性能的矩阵运算)；用于数据挖掘和数据分析，同时也提供数据清洗功能。

任务：Pandas的主要特点

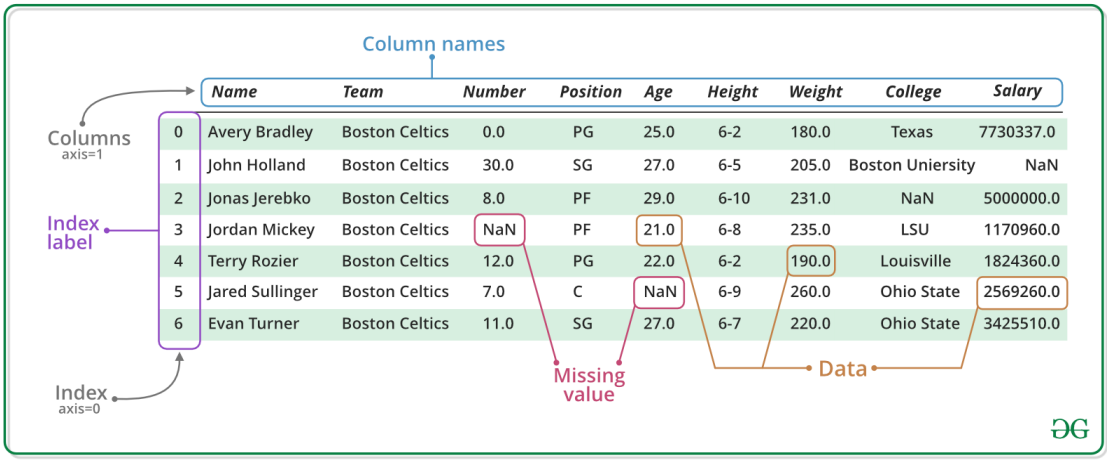
Pandas是一个开放源码的Python库，它使用强大的数据结构提供高性能的数据操作和分析工具。Pandas用于广泛的领域，包括数据分析，人工智能，金融，经济，统计，分析等学术和商业领域。

> 知识点：
>
> - 带标签的数据结构：高效的一维`Series`和二维`DataFrame`数据结构，具有默认和自定义的索引。
> - 多样化数据格式：将数据从**不同文件格式**加载到内存中的数据对象的工具，例如`txt`、`csv`、`SQL`数据库等。
> - 缺失值：轻松处理**缺失值**数据(以`NaN`表示)以及非浮点数据。
> - 数据重构：按数据**分组**进行聚合和转换，**数据透视**功能。
> - 个性化索引：提供高效的标签切片、花式索引、简单索引和多级索引。
> - 支持**时间序列**数据，数据重采样功能。

任务：Pandas的三大数据结构

> 知识点：
>
> - 系列(`Series`)类型：一维数据结构，它是由一组数据以及与之相关的数据标签(即索引)组成。
> - 数据帧(`DataFrame`)类型：二维数据结构，它是Pandas中的一个**表格型**的数据结构，包含有一组有序的列，每列可以是不同的值类型(数值、字符串、布尔型等)。`DataFrame`即有行索引也有列索引，可以被看做是由`Series`组成的字典。
> - 面板(`Panel`)类型：三维数据结构，用于高维的数据描述。
> - 理解：`Series`是基本数据的容器、`DataFrame`是`Series`的容器，`Panel`是`DataFrame`的容器。

> 知识点：
>
> - `Series`和`DataFrame`是Pandas用的最多的数据结构。
> - `Series`是一维数据结构。例如，下面的`Series`是整数集合。
>
> | 10 | 23 | 56 | 17 | 52 | 61 | 73 | 90 | 26 | 72 |
> |---|---|---|---|---|---|---|---|---|---|
>
> - `DataFrame`是二维数据结构。例如，表格数据，带标签，异构混合类型。



	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston Uniersity	NaN
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0	6-8	235.0	LSU	1170960.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN	6-9	260.0	Ohio State	2569260.0
6	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0

任务：Pandas的导包

Anaconda自带安装Numpy、Pandas、Matplot模块。测试工作环境是否有安装好了Pandas，导入相关包如下：

```
# 领域内惯用的三个模块导包统一简写命名方式
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
print(pd.__version__)
```

> 知识点：
>
> - 建议在Jupyter的第一个`cell`添加三个模块的导包语句。之后的cell，不再需要重复导包。
> - **本章所有的例子，都默认已经导包。**

9.2 数据结构：一维Series类型

9.2.1 Series的基本概念

任务：Series的基本概念

> 知识点：
>
> - `Series`是带索引的一维数据结构，数据类型可以是整数，字符串，浮点数，Python对象等。
> - 类似于字典，`Series`的索引可以是任何的**不可变类型**。
> - 类似于NumPy数组，`Series`数据必须是**相同**类型。

任务：Series的构造函数

`Series`的创建函数描述如下：

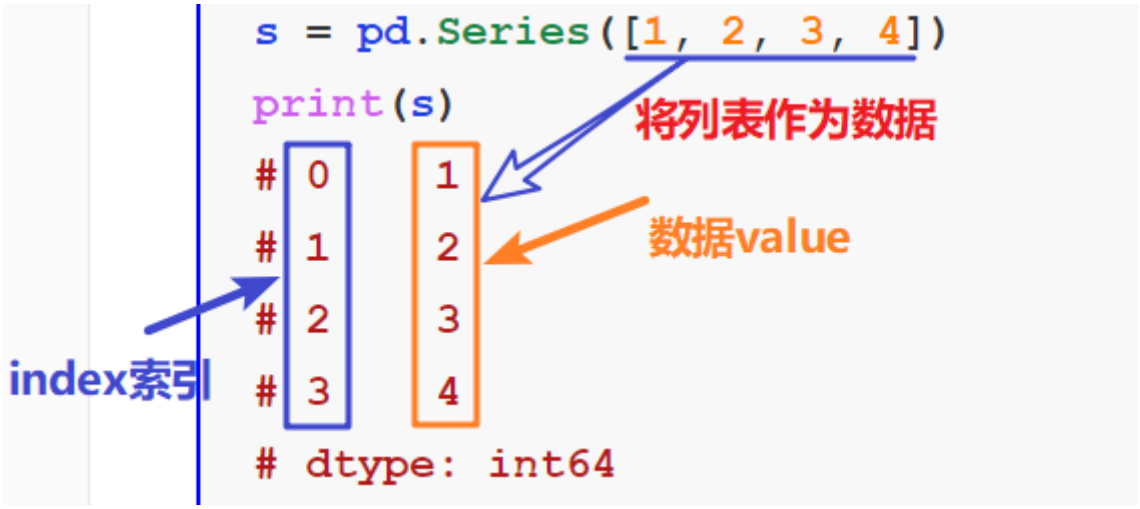
```
pandas.Series(data, index, dtype, copy)
```

构造函数的参数如下：

- `data`：数据。数组`ndarray`，列表`list`、字典、标量或常量。
- `index`：索引值必须是**唯一**的(不可变类型)，与**数据的长度相同。** 默认`np.arange(n)`如果没有索引被传递。
- `dtype`：`dtype`用于数据类型。如果没有，Pandas会根据数据类型进行推断。
- `copy`：复制数据，默认为`false`。

例子：创建一个`Series`对象。

```
s = pd.Series([1, 2, 3, 4])
print(s)
# 0    1
# 1    2
# 2    3
# 3    4
# dtype: int64
```



> 知识点：
>
> - `Series`的左侧是`index`(索引)。
> - 调用`Series`构造函数时，若不指明`index`，那么Pandas会默认使用从0开始依次递增的数值作为`index`数组。
> - `Series`的右侧是`index`对应的`values`，即封装的数据。

任务：获取标签和数据

> 知识点：
>
> - `index`属性：获取**标签**索引。标签索引类型是`Index`，或它的子类型。
> - `values`属性：获取数据。类型是`ndarray`。

```
s = pd.Series([1, 2, 3, 4], index=['a','b','c','d'])
# 获得标签索引
print(s.index)
#Index(['a', 'b', 'c', 'd'], dtype='object')

# 获得数据，一维数组
print(type(s.values), s.values)
# <class 'numpy.ndarray'> [1 2 3 4]
```

9.2.2 Series的创建

任务：创建一个空`Series`

```
s = pd.Series()
print(s)
# Series([], dtype: float64)
```

`Series`有3中创建方式，从**标量**、从**数组**(可以是列表、元组和NumPy数组)和从**字典**。

任务：从标量创建`Series`

> 知识点：
>
> - 如果数据是标量值，则**必须**提供`index`索引参数。
> - 将重复该标量值，以匹配索引的长度。

```
print(pd.Series(5, index=[100, 200, 300]))
# 100    5
# 200    5
# 300    5
# dtype: int64
```

任务：从`ndarray`数组创建`Series`

> 知识点：
>
> - 如果数据是`ndarray`，则传递的索引必须具有相同的长度。
> - 如果没有传递索引值，那么默认索引将是范围(`n`)，其中`n`是数组长度，即`[0,1,…,len(data)-1]`。

```
data = np.array(['a','b','c','d'])
s = pd.Series(data)
print(s)
# 0   a
# 1   b
# 2   c
# 3   d
# dtype: object
```

这里没有传递任何索引，因此默认情况下，它分配了从`0`到`len(data)-1`的索引，即：`0`到`3`。

任务：从字典创建创建`Series`

> 知识点：
>
> - 字典(`dict`)可以作为输入传递。
> - 如果没有指定`index`，将字典的`key`用于构建`Series`的索引。
> - `Series`元素的顺序按传递的字典`key`顺序排列。

```
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data)
print(s)
# a 0.0
# b 1.0
# c 2.0
# dtype: float64
```

> 知识点：如果传递`index`参数。根据`index`，
>
> - 将字典中与相匹配的`key`对应的元素取出。
> - 没匹配的`key`对应的元素使用`NaN`(不是数字)填充。
> - `Series`元素的顺序按`index`顺序排列。

```
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data,index=['b','c','d','a'])
print(s)
# b 1.0
# c 2.0
# d NaN
# a 0.0
# dtype: float64
```

> 知识点：通过`index`可以对字典数据进行筛选。

```
# 保留index指定的索引的字典元素
print(pd.Series({2:'a', 1:'b', 3:'c'}, index=[3, 2]))
# 3    c
# 2    a
# dtype: object
```

9.2.3 Series与一维数组和字典的关系

任务：Serise是通用的NumPy数组

> 知识点：`Series`和NumPy的一维数组的本质区别在于索引：
>
> - 类似列表，NumPy数组通过**隐式**整数索引获取数值。
> - 类似字典，`Series`使用显示索引`index`与数值关联。
>
> - 显式索引让`Series`拥有了更强的数据处理能力。
> - 索引不仅仅是整数，可以是任意不可变类型(类似于字典的`key`)。

```
data = pd.Series([0.25, 0.5, 0.75, 1.0], index=['a', 'b', 'c', 'd'])
print(data)

# 类似字典，可以通过索引来访问元素。
data['b']# 0.5
# 可以使用不连续的索引。
data = pd.Series([0.25, 0.5, 0.75, 1.0], index=[2, 5, 3, 7])
print(data)
print(data[5])
```

任务：Series是特殊的字典

> 知识点：
>
> - 可以把`Series`看成是一种特殊的Python字典。
>   - 字典是将`key`映射到`value`的数据结构。
>   - `Series`是一组索引映射到一组**类型值**的数据结构。
> - 相对于，NumPy数组的底层实现是通过优化编译的，它的效率比普通的Python列表更高效。
> - `Series`是建立在NumPy基础上.因此，它的操作会比Python字典更高效。

例子：可以直接将字典转换为`Series`。

```
population_dict = {'California': 38332521, 'Texas': 26448193,
                   'New York': 19651127,'Florida': 19552860,
                   'Illinois': 12882135}
population = pd.Series(population_dict)
print(population)
```

> 知识点：
>
> - 用字典创建`Series`时，其索引默认按照顺序排列。
> - 与字典不同，`Series`是**有序**的，因此，它支持切片操作。

```
# 类似字典，通过标签来访问元素
print(population['California'])
# 支持切片操作。字典不能使用切片因为是无序的
print(population['California':'Illinois'])
```

9.2.4 Series的索引

`Series`共有两种访问元素方式：**隐式位置索引**和**显式标签索引**。

任务：隐式位置索引

> 知识点：类似于`ndarray`数组和列表，`Series`可以根据位置来**隐式索引**元素。

例子：检索第一个元素。由于数组是从零开始计数的，那么第一个元素存储在零位置。

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s[0])
# 1
```

例子：使用切片索引`Series`的前三个元素。

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s[:3])
# a  1
# b  2
# c  3
# dtype: int64
```

例子：使用切片索引`Series`的最后三个元素。

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s[-3:])
# c  3
# d  4
# e  5
# dtype: int64
```

任务：显式标签索引

> 知识点：类似于字典，`Series`可以通过**标签**来**显式索引**元素。

例子：使用标签索引来访问单个元素。

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s['a'])
# 1
```

例子：使用标签索引来访问多个元素。

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s[['a','c','d']])
# a  1
# c  3
# d  4
# dtype: int64
```

例子：如果不包含指定的标签，则会出现异常。

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s['f'])
# KeyError: 'f'
```