集合支持数学上的集合操作，例如联合、交集、差集、对称差集。考虑以下示例集合：

```
In [135]: a = {1, 2, 3, 4, 5}
In [136]: b = {3, 4, 5, 6, 7, 8}
```

两个集合的联合就是两个集合中不同元素的并集。可以通过 union 方法或 | 二元操作符完成：

```
In [137]: a.union(b)
Out[137]: {1, 2, 3, 4, 5, 6, 7, 8}

In [138]: a | b
Out[138]: {1, 2, 3, 4, 5, 6, 7, 8}
```

交集包含了两个集合中同时包含的元素。可以使用 & 操作符或 intersection 方法获得交集：

```
In [139]: a.intersection(b)
Out[139]: {3, 4, 5}

In [140]: a & b
Out[140]: {3, 4, 5}
```

表 3-1 是常用的集合方法列表。

表 3-1：Python 集合操作

函数	替代方法	描述
a.add(x)	N/A	将元素 x 加入集合 a
a.clear()	N/A	将集合重置为空，清空所有元素
a.remove(x)	N/A	从集合 a 移除某个元素
a.pop()	N/A	移除任意元素，如果集合是空的抛出 keyError
a.union(b)	a|b	a 和 b 中的所有不同元素
a.update(b)	a|=b	将 a 的内容设置为 a 和 b 的并集
a.intersection(b)	a&b	a、b 中同时包含的元素
a.intersection_update(b)	a&= b	将 a 的内容设置为 a 和 b 的交集
a.difference(b)	a-b	在 a 不在 b 的元素
a.difference_update(b)	a-=b	将 a 的内容设为在 a 不在 b 的元素
a.symmetric_difference(b)	a^b	所有在 a 或 b 中，但不是同时在 a、b 中的元素
a.symmetric_difference_update(b)	a^=b	将 a 的内容设为所有在 a 或 b 中，但不是同时在 a、b 中的元素
a.issubset(b)	N/A	如果 a 包含于 b 返回 True

表 3-1：Python 集合操作（续）

函数	替代方法	描述
`a.issuperset(b)`	N/A	如果 a 包含 b 返回 True
`a.isdisjoint(b)`	N/A	a、b 没有交集返回 True

所有的逻辑集合运算都有对应操作，允许你用操作的结果替代操作左边的集合内容。对于大型集合，下面的代码效率更高：

```
In [141]: c = a.copy()

In [142]: c |= b

In [143]: c
Out[143]: {1, 2, 3, 4, 5, 6, 7, 8}

In [144]: d = a.copy()

In [145]: d &= b

In [146]: d
Out[146]: {3, 4, 5}
```

和字典类似，集合的元素必须是不可变的。如果想要包含列表型的元素，必须先转换为元组：

```
In [147]: my_data = [1, 2, 3, 4]

In [148]: my_set = {tuple(my_data)}

In [149]: my_set
Out[149]: {(1, 2, 3, 4)}
```

你还可以检查一个集合是否是另一个集合的子集（包含于）或超集（包含）：

```
In [150]: a_set = {1, 2, 3, 4, 5}

In [151]: {1, 2, 3}.issubset(a_set)
Out[151]: True

In [152]: a_set.issuperset({1, 2, 3})
Out[152]: True
```

当且仅当两个集合的内容一模一样时，两个集合才相等：

```
In [153]: {1, 2, 3} == {3, 2, 1}
Out[153]: True
```

3.1.6 列表、集合和字典的推导式

*列表推导式*是最受欢迎的 Python 语言特性之一。它允许你过滤一个容器的元素，用一种

```
In [27]: arr1.dtype
Out[27]: dtype('float64')

In [28]: arr2.dtype
Out[28]: dtype('int64')
```

除了 np.array，还有很多其他函数可以创建新数组。例如，给定长度及形状后，zeros 可以一次性创造全 0 数组，ones 可以一次性创造全 1 数组。empty 则可以创建一个没有初始化数值的数组。想要创建高维数组，则需要为 shape 传递一个元组：

```
In [29]: np.zeros(10)
Out[29]: array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])

In [30]: np.zeros((3, 6))
Out[30]:
array([[ 0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.]])

In [31]: np.empty((2, 3, 2))
Out[31]:
array([[[ 0.,  0.],
        [ 0.,  0.],
        [ 0.,  0.]],
       [[ 0.,  0.],
        [ 0.,  0.],
        [ 0.,  0.]]])
```

想要使用 np.empty 来生成一个全 0 数组，并不安全，有些时候它可能会返回未初始化的垃圾数值。

arange 是 Python 内建函数 range 的数组版：

```
In [32]: np.arange(15)
Out[32]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

表 4-1 展示的是标准数组的生成函数。由于 NumPy 专注于数值计算，如果没有特别指明的话，默认的数据类型是 float64（浮点型）。

表 4-1：数组生成函数

函数名	描述
array	将输入数据（可以是列表、元组、数组以及其他序列）转换为 ndarray，如不显式指明数据类型，将自动推断；默认复制所有的输入数据
asarray	将输入转换为 ndarray，但如果输入已经是 ndarray 则不再复制
arange	Python 内建函数 range 的数组版，返回一个数组

表 4-1：数组生成函数（续）

函数名	描述
ones	根据给定形状和数据类型生成全 1 数组
ones_like	根据所给的数组生成一个形状一样的全 1 数组
zeros	根据给定形状和数据类型生成全 0 数组
zeros_like	根据所给的数组生成一个形状一样的全 0 数组
empty	根据给定形状生成一个没有初始化数值的空数组
empty_like	根据所给数组生成一个形状一样但没有初始化数值的空数组
full	根据给定的形状和数据类型生成指定数值的数组
full_like	根据所给的数组生成一个形状一样但内容是指定数值的数组
eye, identity	生成一个 N×N 特征矩阵（对角线位置都是 1，其余位置是 0）

4.1.2 ndarray 的数据类型

数据类型，即 dytpe，是一个特殊的对象，它包含了 ndarray 需要为某一种类型数据所申明的内存块信息（也称为*元数据*，即表示数据的数据）：

```
In [33]: arr1 = np.array([1, 2, 3], dtype=np.float64)

In [34]: arr2 = np.array([1, 2, 3], dtype=np.int32)

In [35]: arr1.dtype
Out[35]: dtype('float64')

In [36]: arr2.dtype
Out[36]: dtype('int32')
```

dtype 是 NumPy 能够与其他系统数据灵活交互的原因。通常，其他系统提供一个硬盘或内存与数据的对应关系，使得利用 C 或 Fortran 等底层语言读写数据变得十分方便。数据的 dtype 通常都是按照一个方式命名：类型名，比如 `float` 和 `int`，后面再接上表明每个元素位数的数字。一个标准的双精度浮点值（Python 中数据类型为 `float`），将使用 8 字节或 64 位。因此，这个类型在 NumPy 中称为 `float64`。表 4-2 将展现所有的 NumPy 所支持的数据类型。

不要担心如何记住 NumPy 数据类型，尤其当你还是新手的时候。通常你只需要关心数据的大类，比如是否是浮点型、整数、布尔值、字符串或某个 Python 对象。当你需要在内存或硬盘上做更深入的存取操作时，尤其是大数据集时，你才真正需要了解存储的数据类型。

表 4-2：NumPy 数据类型

类型	类型代码	描述
int8, uint8	i1, u1	有符号和无符号的 8 数位整数
int16, uint16	i2, u2	有符号和无符号的 16 数位整数
int32, uint32	i4, u4	有符号和无符号的 32 数位整数
int64, uint64	i8, u8	有符号和无符号的 64 数位整数
float16	f2	半精度浮点数
float32	f4 或 f	标准单精度浮点数；兼容 C 语言 float
float64	f8 或 d	标准双精度浮点数；兼容 C 语言 double 和 Python float
float128	f16 或 g	拓展精度浮点数
complex64, complex128, complex256	c8,c16,c32	分别基于 32 位、64 位、128 位浮点数的复数
bool	?	布尔值，存储 True 或 False
object	O	Python object 类型
string_	S	修正的 ASC II 字符串类型；例如生成一个长度为 10 的字符串类型，使用 'S10'
unicode_	U	修正的 Unicode 类型，生成一个长度为 10 的 Unicode 类型，使用 'U10'

你可以使用 astype 方法显式地转换数组的数据类型：

```
In [37]: arr = np.array([1, 2, 3, 4, 5])

In [38]: arr.dtype
Out[38]: dtype('int64')

In [39]: float_arr = arr.astype(np.float64)

In [40]: float_arr.dtype
Out[40]: dtype('float64')
```

在上面例子中，整数被转换成了浮点数。如果我把浮点数转换成整数，则小数点后的部分将被消除：

```
In [41]: arr = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])

In [42]: arr
Out[42]: array([  3.7,  -1.2,  -2.6,   0.5,  12.9,  10.1])

In [43]: arr.astype(np.int32)
Out[43]: array([ 3, -1, -2,  0, 12, 10], dtype=int32)
```

如果你有一个数组，里面的元素都是表达数字含义的字符串，也可以通过 astype 将字

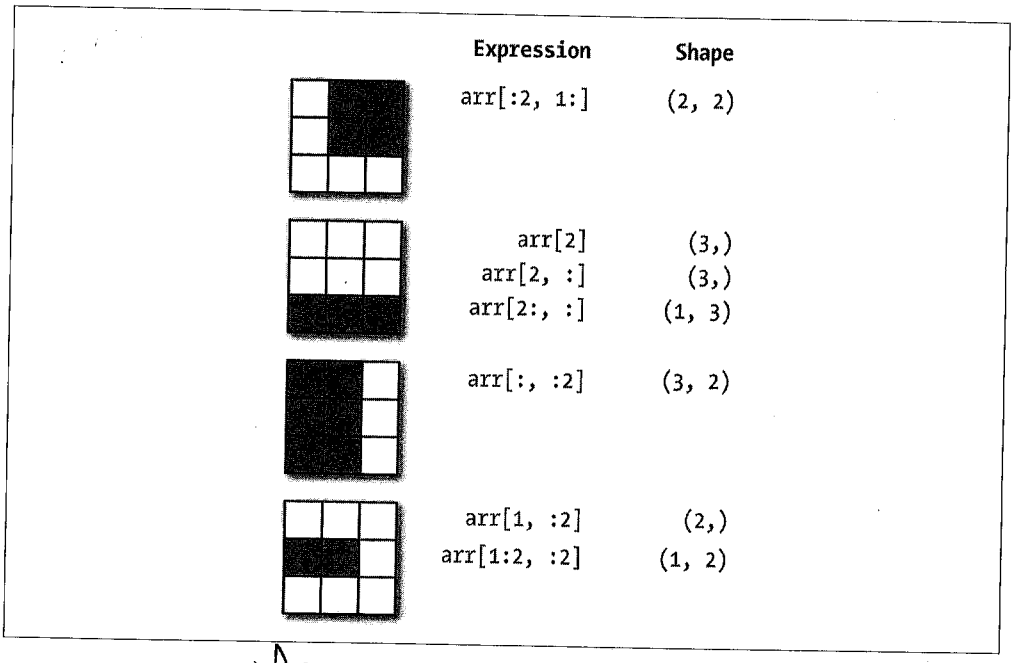

图 4-2：二维数组的切片

4.1.5 布尔索引

让我们考虑以下例子，假设我们的数据都在数组中，并且数组中的数据是一些存在重复的人名。我会使用 numpy.random 中的 randn 函数来生成一些随机正态分布的数据：

```
In [98]: names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])

In [99]: data = np.random.randn(7, 4)

In [100]: names
Out[100]:
array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'],
      dtype='<U4')

In [101]: data
Out[101]:
array([[ 0.0929,  0.2817,  0.769 ,  1.2464],
       [ 1.0072, -1.2962,  0.275 ,  0.2289],
       [ 1.3529,  0.8864, -2.0016, -0.3718],
       [ 1.669 , -0.4386, -0.5397,  0.477 ],
       [ 3.2489, -1.0212, -0.5771,  0.1241],
       [ 0.3026,  0.5238,  0.0009,  1.3438],
       [-0.7135, -0.8312, -2.3702, -1.8608]])
```

假设每个人名都和 data 数组中的一行相对应，并且我们想要选中所有 'Bob' 对应

```
       -0.6605])

In [144]: y
Out[144]:
array([ 0.8626, -0.01  ,  0.05  ,  0.6702,  0.853 , -0.9559, -0.0235,
       -2.3042])

In [145]: np.maximum(x, y)
Out[145]:
array([ 0.8626,  1.0048,  1.3272,  0.6702,  0.853 ,  0.0222,  0.7584,
       -0.6605])
```

这里，numpy.maximum 逐个元素地将 x 和 y 中元素的最大值计算出来。

也有一些通用函数返回多个数组。比如 modf，是 Python 内建函数 divmod 的向量化版本。它返回了一个浮点值数组的小数部分和整数部分：

```
In [146]: arr = np.random.randn(7) * 5

In [147]: arr
Out[147]: array([-3.2623, -6.0915, -6.663 ,  5.3731,  3.6182,  3.45  ,  5.0077])

In [148]: remainder, whole_part = np.modf(arr)

In [149]: remainder
Out[149]: array([-0.2623, -0.0915, -0.663 ,  0.3731,  0.6182,  0.45  ,  0.0077])

In [150]: whole_part
Out[150]: array([-3., -6., -6.,  5.,  3.,  3.,  5.])
```

通用函数接收一个可选参数 out，允许对数组按位置操作：

```
In [151]: arr
Out[151]: array([-3.2623, -6.0915, -6.663 , 5.3731, 3.6182, 3.45 , 5.0077])

In [152]: np.sqrt(arr)
Out[152]: array([ nan,  nan,  nan,  2.318,  1.9022,  1.8574,  2.2378])

In [153]: np.sqrt(arr, arr)
Out[153]: array([ nan,  nan,  nan,  2.318,  1.9022,  1.8574,  2.2378])

In [154]: arr
Out[154]: array([ nan,  nan,  nan,  2.318,  1.9022,  1.8574,  2.2378])
```

表 4-3 和表 4-4 列举的是可用的通用函数

表 4-3：一元通用函数

函数名	描述
abs、fabs	逐元素地计算整数、浮点数或复数的绝对值
sqrt	计算每个元素的平方根（与 arr ** 0.5 相等）
square	计算每个元素的平方（与 arr**2 相等）

表 4-3：一元通用函数（续）

函数名	描述
exp	计算每个元素的自然指数值 e^x
log、log10、log2、log1p	分别对应：自然对数（e 为底）、对数 10 为底、对数 2 为底、log（1+x）
sign	计算每个元素的符号值：1（正数）、0（0）、-1（负数）
ceil 小整数）	计算每个元素的最高整数值（即大于等于给定数值的最
floor 大整数）	计算每个元素的最小整数值（即小于等于给定元素的最
rint	将元素保留到整数位，并保持 dtype
modf	分别将数组的小数部分和整数部分按数组形式返回
isnan	返回数组中的元素是否是一个 NaN（不是一个数值），形式为布尔值数组
isfinite、isinf	分别返回数组中的元素是否有限（非 inf、非 NaN）、是否无限的，形式为布尔值数组
cos、cosh、sin、 sinh、tan、tanh	常规的双曲三角函数
arccos、arccosh、arcsin、 arcsinh、arctan、arctanh	反三角函数
logical_not	对数组的元素按位取反（与 ~arr 效果一致）

表 4-4：二元通用函数

函数名	描述
add	将数组的对应元素相加
subtract	在第二个数组中，将第一个数组中包含的元素去除
multiply	将数组的对应元素相乘
divide, floor_divide	除或整除（放弃余数）
power	将第二个数组的元素作为第一个数组对应元素的幂次方
maximum, fmax	逐个元素计算最大值，fmax 忽略 NaN
minimum, fmin	逐个元素计算最小值，fmin 忽略 NaN
mod	按元素的求模计算（即求除法的余数）
copysign	将第一个数组的符号值改为第二个数组的符号值
greater,greater_equal,less, less_equal,equal,not_equal	进行逐个元素的比较，返回布尔值数组（与数学操作符 >、>=、<、<=、==、!= 效果一致）
logical_and,logical_or, logical_xor	进行逐个元素的逻辑操作（与逻辑操作符 &、|、^ 效果一致）

表 4-5：基础数组统计方法

方法	描述
sum	沿着轴向计算所有元素的累和，0 长度的数组，累和为 0
mean	数学平均，0 长度的数组平均值为 NaN
std, var	标准差和方差，可以选择自由度调整（默认分母是 n）
min, max	最小值和最大值
argmin, argmax	最小值和最大值的位置
cumsum	从 0 开始元素累积和
cumprod	从 1 开始元素累积积

表 4-6：数组的集合操作

方法	描述
unique(x)	计算 x 的唯一值，并排序
intersect1d(x, y)	计算 x 和 y 的交集，并排序
union1d(x, y)	计算 x 和 y 的并集，并排序
in1d(x, y)	计算 x 中的元素是否包含在 y 中，返回一个布尔值数组
setdiff1d(x, y)	差集，在 x 中但不在 y 中的 x 的元素
setxor1d(x, y)	异或集，在 x 或 y 中，但不属于 x、y 交集的元素

表 4-7：常用 numpy.linalg 函数

函数	描述
diag	将一个方阵的对角（或非对角）元素作为一维数组返回，或者将一维数组转换成一个方阵，并且在非对角线上有零点
dot	矩阵点乘
trace	计算对角元素和
det	计算矩阵的行列式
eig	计算方阵的特征值和特征向量
inv	计算方阵的逆矩阵
pinv	计算矩阵的 Moore-Penrose 伪逆
qr	计算 QR 分解
svd	计算奇异值分解（SVD）
solve	求解 x 的线性系统 Ax = b，其中 A 是方阵
lstsq	计算 Ax = b 的最小二乘解

在启动 Jupyter 的时候，你可以添加 *--config* 参数：

```
jupyter notebook --config=~/.jupyter/my_custom_config.py
```

B.6 附录小结

在你实验完本书中的代码示例后，你的技能获得增长，并成为了一名 Python 编程者，我

表 4-8：numpy.random 中的部分函数列表

函数	描述
seed	向随机数生成器传递随机状态种子
permutation	返回一个序列的随机排列，或者返回一个乱序的整数范围序列
shuffle	随机排列一个序列
rand	从均匀分布中抽取样本
randint	根据给定的由低到高的范围抽取随机整数
randn	从均值 0 方差 1 的正态分布中抽取样本（MATLAB 型接口）
binomial	从二项分布中抽取样本
normal	从正态（高斯）分布中抽取样本
beta	从 beta 分布中抽取样本
chisquare	从卡方分布中抽取样本
gamma	从伽马分布中抽取样本
uniform	从均匀 [0,1）分布中抽取样本