第 3 章

NumPy 数据处理基石

Excel 中的数组非常有用，不但能灵活地处理数据，更重要的是能提高程序的运算速度，我们也可以把 NumPy 理解为 Excel 中的数组。本章主要讲解 NumPy 数组的创建与转换、数组的预处理，以及数组维度的转换。最后讲解与数组有关的 Series 数据与 DataFrame 表格的创建。

3.1 NumPy 的定义

在第 2 章中获取 DataFrame 表格数据时，使用 values 属性返回数组类型的数据。DataFrame 表格可以由 Series 数据构成，Series 数据的本质是带标签的一维数组。这里说的数组是指 NumPy 库中的数组，因为 Python 本身并没有数组这种数据类型。

NumPy 是一个运行速度非常快的数学库。支持大量维度数组与矩阵运算，而 Pandas 是基于 NumPy 的一种数据分析工具，因此要学好 Pandas，就必须对 NumPy 有所了解。为了让 NumPy 表达起来更方便，将 NumPy 简写为 np。

3.2 NumPy 数组的创建与转换

本节讲解普通数组、序列数组、随机小数数组和随机整数数组的创建，以及如何将整个 DataFrame 表格数据转换为数组，将 DataFrame 表格每列数据转换为数组。

3.2.1 普通数组

创建普通数组可以使用 np.array()函数，此函数的参数可以是任何序列类型的对象。例如将列表创建为普通数组的代码如下：

```
#chapter3\3-2\3-2-1.ipynb
import numpy as np                                  #导入 NumPy 库，并命名为 np
arr = np.array(['张三','李四','王二'])              #创建普通数组，并赋值给 arr 变量
print(arr)                                          #打印 arr 数组
print(type(arr))                                    #打印 arr 数据类型
```

运行结果如下：

```
['张三' '李四' '王二']
<class 'numpy.ndarray'>
```

print(arr)打印出的数组显示为['张三' '李四' '王二']，呈现方式与列表比较相似，列表的元素之间用逗号分隔，数组的元素之间用空格分隔。

print(type(arr))打印出的类型显示为<class 'numpy.ndarray'>，ndarray 表明是普通数组类型，ndarray 是具有相同类型和大小项目的（通常是固定大小的）多维容器。

既然普通数组是多维容器，接下来再创建一个多维数组，示例代码如下：

```
#chapter3\3-2\3-2-2.ipynb
import numpy as np                                          #导入 NumPy 库，并命名为 np
arr = np.array((['张三','李四','王二'],[66,88,99]))        #创建多维数组，并赋值给 arr 变量
print(arr)                                                  #打印 arr 数组
print(type(arr))                                            #打印 arr 数据类型
```

运行结果如下：

```
[['张三' '李四' '王二']
['66' '88' '99']]
<class 'numpy.ndarray'>
```

以上代码创建的是二维数组，数组中的元素数据类型必须保持一致，在上面二维数组中，列表[66,88,99]中的元素是数字类型，但生成为数组后，数组中所有元素的数据类型转换为了字符串类型。

3.2.2 序列数组

创建一个指定数字范围内的等差序列数组，可以使用 np.arange()函数。它创建数组的方式非常灵活，参数可以是 1 个、2 个或者 3 个，示例代码如下：

```
#chapter3\3-2\3-2-3.ipynb
import numpy as np                                          #导入 NumPy 库，并命名为 np
arr1 = np.arange(4)                                         #np.arange()为 1 个参数
arr2 = np.arange(7,12)                                      #np.arange()为 2 个参数
arr3 = np.arange(100,110,2)                                 #np.arange()为 3 个参数
print(arr1)                                                 #打印 arr1 数组
print(arr2)                                                 #打印 arr2 数组
print(arr3)                                                 #打印 arr3 数组
```

运行结果如下：

```
[0 1 2 3]
[ 7 8 9 10 11]
[100 102 104 106 108]
```

通过观察上面代码运行的结果，可以得出 np.arange()函数的 3 种情况。

(1) 1 个参数时，起始值默认为 0，参数值为终止值，步长值默认为 1。

(2) 2 个参数时，第 1 个参数为起始值，第 2 个参数为终止值，步长值默认为 1。

(3) 3 个参数时，第 1 个参数为起始值，第 2 个参数为终止值，第 3 个参数为步长值。

3.2.3　随机数组

在做数据测试时，经常需要生成一些随机数据。在 NumPy 中可使用 np.random.rand()函数生成随机小数。下面分别创建单个随机小数，一维、二维和三维随机小数数组，示例代码如下：

```
#chapter3\3-2\3-2-4.ipynb
import numpy as np                                  #导入 NumPy 库，并命名为 np
rnd = np.random.rand()                              #创建单个随机小数
arr1 = np.random.rand(4)                            #创建一维随机小数数组
arr2 = np.random.rand(2,4)                          #创建二维随机小数数组
arr3 = np.random.rand(2,4,3)                        #创建三维随机小数数组
print(rnd)                                          #打印单个随机小数
print(arr1)                                         #打印 arr1 随机小数数组
print(arr2)                                         #打印 arr2 随机小数数组
print(arr3)                                         #打印 arr3 随机小数数组
```

运行结果如下：

```
#打印的 rnd 随机小数如下所示
0.8988843798934052

#打印的 arr1 随机小数数组如下所示
[0.6587055  0.96748318  0.13119374  0.49618439]

#打印的 arr2 随机小数数组如下所示
[[0.52212765  0.74314483  0.87741946  0.19396058]
 [0.59023371  0.07754737  0.5386383  0.91278296]]

#打印的 arr3 随机小数数组如下所示
[[[0.14415337  0.52691613  0.84457373]
  [0.35925086  0.66358187  0.60983805]
  [0.4784553  0.5777714  0.74465181]
  [0.20927365  0.43836476  0.90326771]]
 [[0.62324085  0.65458412  0.41754146]
  [0.20506602  0.75364  0.5395965]
  [0.34599508  0.22240642  0.99718073]
  [0.67285606  0.5008097  0.1211656 ]]]
```

上面的示例只演示到了三维随机小数数组，如果要创建更多维度，在 np.random.rand()函数的参数中添加即可。

创建随机整数数组可使用 np.random.randint()函数，使用函数的第 1 和 2 个参数分别指定随机整数的起始值和终止值，使用 size 参数指定随机整数的维度。下面分别创建单个

随机整数，一维、二维和三维随机整数数组，示例代码如下：

```
#chapter3\3-2\3-2-5.ipynb
import numpy as np                                    #导入 NumPy 库,并命名为 np
rndint = np.random.randint(10,99)                     #创建单个随机整数
arr1 = np.random.randint(10,99,size = (3))            #创建一维随机整数数组
arr2 = np.random.randint(10,99,size = (3,2))          #创建二维随机整数数组
arr3 = np.random.randint(10,99,size = (3,2,4))        #创建三维随机整数数组
print(rndint)                                         #打印单个随机整数
print(arr1)                                           #打印 arr1 随机整数数组
print(arr2)                                           #打印 arr2 随机整数数组
print(arr3)                                           #打印 arr3 随机整数数组
```

运行结果如下：

```
#打印的 rndint 随机整数如下所示
26

#打印的 arr1 随机整数数组如下所示
[52  48  80]

#打印的 arr2 随机整数数组如下所示
[[29  89]
 [76  98]
 [56  59]]

#打印的 arr3 随机整数数组如下所示
[[[56  71  97  13]
  [72  67  24  26]]
 [[88  86  88  98]
  [18  29  90  23]]
 [[55  62  85  18]
  [91  35  78  34]]]
```

3.2.4 转换数组

通过前文我们已了解了 NumPy 数组的结构及创建方式，但在实际工作中，不但需要创建数组，很多时候更需要将 DataFrame 表格和 Series 数据转换为数组。

1. DataFrame 表格转换为数组

首先读取 Excel 文件中的数据，如图 3-1 所示，将信息表数据读取给 df 变量，df 是 DataFrame 类型表格，再分别使用 np.array()函数、df.to_numpy()函数和 df.values 属性将 df 表转换为数组，示例代码如下：

	A	B	C
1	姓名	入职日期	基础工资
2	张三	2020/11/5	12000
3	李四	2018/6/3	14000
4	王二	2019/10/19	13500

信息表

图 3-1 被读取的 Excel 文件

```
#chapter3\3-2\3-2-6.ipynb
import numpy as np,pandas as pd                    #导入 NumPy 库和 Pandas 库,并分别命名为 np 和 pd
df = pd.read_excel('3-2-6.xlsx','信息表') #读取 Excel 中的信息表数据
arr1 = np.array(df)                                #使用 np.array()函数转换为数组
arr2 = df.to_numpy()                               #使用 df.to_numpy()函数转换为数组
arr3 = df.values                                   #使用 df.values 属性转换为数组
print(arr1)                                        #打印 arr1 数组
```

运行结果如下：

```
[['张三' Timestamp('2020-11-05 00:00:00') 12000]
 ['李四' Timestamp('2018-06-03 00:00:00') 14000]
 ['王二' Timestamp('2019-10-19 00:00:00') 13500]]
```

在上面的代码中，变量 arr1、arr2 和 arr3 返回的数组结果均相同，所以只打印了 arr1 数组。

2. Series 数据转换为数组

Series 数据也可以转换为数组，同样用 np.array()函数、s.to_numpy()函数和 s.values 属性 3 种不同的方法将 Series 数据转换为数组，同样使用图 3-1 所示的 Excel 文件，读取并赋值给 df 变量，然后循环读取 df 中的每列数据，使用不同的方法转换为数组，示例代码如下：

```
#chapter3\3-2\3-2-7.ipynb
import numpy as np,pandas as pd                    #导入 NumPy 库和 Pandas 库,并分别命名为 np 和 pd
df = pd.read_excel('3-2-7.xlsx','信息表') #读取 Excel 中的信息表数据
for t,s in df.items():
    print(t)                                       #打印 df 列索引名称
    arr1 = np.array(s)                             #使用 np.array()函数转换列数据为数组
    arr2 = s.to_numpy()                            #使用 s.to_numpy()函数转换列数据为数组
    arr3 = s.values                                #使用 s.values 属性转换列数据为数组
    print(arr3)                                    #打印 arr3 数组
    print('--------------------')
```

运行结果如下：

```
姓名
['张三' '李四' '王二']
--------------------
入职日期
['2020-11-05T00:00:00.000000000' '2018-06-03T00:00:00.000000000'
'2019-10-19T00:00:00.000000000']
--------------------
基础工资
[12000 14000 13500]
--------------------
```

上面的代码是将 df 表中每列数据循环出来赋值给 s 变量，s 变量中存储的就是 Series

数据。再通过 3 种不同方法分别转换赋值给变量 arr1、arr2 和 arr3，它们存储的数据相同，并且均是一维数组。

3.3 NumPy 数组的预处理

在对数组做运算之前，可能会对数据做一些预处理。例如数据类型的转换，缺失值的处理，重复值的处理等。如果没做好这些预处理，后续数组的运算将无法进行下去。

3.3.1 类型转换

虽然要求 NumPy 数组的数据类型相同，但仍可能会遇到数据类型不一致的数组，或者用户需要将数组转换成指定的数据类型。数组的数据类型转换可以使用 astype()函数，示例代码如下：

```
#chapter3\3-3\3-3-1.ipynb
import numpy as np                                  #导入 NumPy 库,并命名为 np
arr = np.array([100,'123',99])                      #arr 数组
print(arr.astype('int'))                            #转换为整数类型
print(arr.astype('float'))                          #转换为小数类型
print(arr.astype('str'))                            #转换为字符串类型
```

运行结果如下：

```
[100  123  99]
[100.  123.  99.]
['100' '123' '99']
```

上面代码中只演示了'int'(整数)、'float'(小数)和'str'(字符串)3 种常见数据类型的转换，更详细的数据转换类型见表 3-1。

表 3-1 数据类型

类型名称	简写	注　　释
bool	?,b1	布尔型数据类型(True 或者 False)
int8	b,i1	字节(－128～127)
int16	h,i2	整数(－32768～32767)
int32	i,i4	整数(－2147483648～2147483647)，可表示为 int
int64	q,i8	整数(－9223372036854775808～9223372036854775807)
uint8	B,u1	无符号整数(0～255)
uint16	H,u2	无符号整数(0～65535)
uint32	I,u4	无符号整数(0～4294967295)
uint64	Q,u8	无符号整数(0～18446744073709551615)
float16	f2, e	半精度浮点数，包括 1 个符号位，5 个指数位，10 个尾数位
float32	f ,f4	单精度浮点数，包括 1 个符号位，8 个指数位，23 个尾数位
float64	d ,f8	双精度浮点数，包括 1 个符号位，11 个指数位，52 个尾数位，可表示为 float

续表

类型名称	简写	注　　释
str	a，S	字符串，只能包含 ASCII 码字符，S 或 a 后带数字表示字符串长度，超出部分将被截断，例如 S20、a10
unicode	U	Unicode 字符串，U 后带数字表示字符串长度，超出部分将被截断，例如 U20
datetime64	M8	年('Y')、月('M')、周('W')、天('D')、小时('h')、分钟('m')、秒('s')、毫秒('ms')、微秒('μs')等
timedelta64		表示时间差，年('Y')、月('M')、周('W')、天('D')、小时('h')、分钟('m')、秒('s')、毫秒('ms')、微秒('μs')

在表 3-1 中的 datetime64(日期类型)的转换也比较常见，将文本型日期转换为标准日期没问题，但如果将数字转换为对应的日期，处理方式有些不同，示例代码如下：

```
#chapter3\3-3\3-3-2.ipynb
import numpy as np                                  #导入 NumPy 库，并命名为 np
arr = np.array([0,12525,145])                       #arr 数组
print(arr.astype('datetime64[D]'))                  #转换为日期类型
```

运行结果如下：

```
['1970-01-01' '2004-04-17' '1970-05-26']
```

通过观察运行的结果，发现数字转换为日期时，日期的第 1 天是从 1970/1/1 开始的，而不同开发环境的起始日期可能不相同，好在 Pandas 提供了 pd.to_datetime()函数，可以让用户自定义起始日期，例如转换 arr＝np.array([0,12525,145])，示例代码如下：

```
#chapter3\3-3\3-3-3.ipynb
import numpy as np,pandas as pd                     #导入 NumPy 库和 Pandas 库，并分别命名为 np 和 pd
arr = np.array([0,12525,145])                       #arr 数组
print(pd.to_datetime(arr,unit = 'D',origin = '1899-12-30'))      #转换为日期类型
```

运行结果如下：

```
DatetimeIndex(['1899-12-30', '1934-04-16', '1900-05-24'], dtype = 'datetime64[ns]',
freq = None)
```

上面代码的核心函数是 pd.to_datetime()，第 1 参数指定要转换 arr 数组，unit＝'D'表示把 arr 数组中的数字视为单位天，origin＝'1899-12-30'用于设置起始日期。

下面测试一下将 Excel 文件中的数字转换成对应的日期，如图 3-2 和图 3-3 所示，示例代码如下：

```
#chapter3\3-3\3-3-4.ipynb
import numpy as np,pandas as pd                     #导入 NumPy 库和 Pandas 库，并分别命名为 np 和 pd
df = pd.read_excel('3-3-4.xlsx','出生日期表')       #读取 Excel 中的出生日期表数据
df['出生日期'] = pd.to_datetime(df['出生日期'],unit = 'D',origin = '1899-12-30')
                                                    #将出生日期列数字转换为日期
df                                                  #输出转换后的效果
```

运行结果如图 3-3 所示。

	A	B	C
1	姓名	出生日期	
2	张三	29932	
3	李四	33730	
4	王二	32055	
5	麻子	33080	
6			

出生日期表

图 3-2　出生日期转换前

	姓名	出生日期
0	张三	1981-12-12
1	李四	1992-05-06
2	王二	1987-10-05
3	麻子	1990-07-26

图 3-3　出生日期转换后

注意：Pandas 和 NumPy 中的很多函数都有 dtype 参数，表明可以在参数中设置数据类型，关于更多的数据类型可以参考表 3-1。

3.3.2　缺失值处理

缺失值是指没有任何值的空元素，例如导入 Excel 文件后，如果某个单元格没有任何值，则会显示为 NaN 或者 NaT(缺失时间)。当然，在 NumPy 中也可以通过 np.nan 来生成缺失值。

要判断数组中是否有缺失值，可使用 np.isnan()函数，将返回由布尔值组成的数组。还可以给缺失值填充指定的值，例如将缺失值填充数字 100，示例代码如下：

```
#chapter3\3-3\3-3-5.ipynb
import numpy as np                              #导入 NumPy 库,并命名为 np
arr = np.array([2,3,np.nan,36,np.nan,99])       #arr 数组
print(arr)                                      #打印 arr 数组
print(np.isnan(arr))                            #打印 arr 数组的每个元素是否是缺失值
arr[np.isnan(arr)] = 100                        #将 arr 数组中的所有缺失值填充为 100
print(arr)                                      #再次打印 arr 数组
```

运行结果如下：

```
#arr 数组返回结果如下所示
[ 2.  3.  nan 36.  nan 99.]

#判断 arr 数组是否是缺失值,返回结果如下所示
[False False True False True False]

#将 arr 数组缺失值填充为 100,返回结果如下所示
[ 2.  3. 100. 36. 100. 99.]
```

3.3.3　重复值处理

在做数据预处理时，去重复处理是比较常见的处理方式。在 NumPy 中，可以使用 np.unique()函数，示例代码如下：

```
#chapter3\3-3\3-3-6.ipynb
import numpy as np                              #导入 NumPy 库,并命名为 np
arr = np.array([9,1,2,2,1,5,9])                 #arr 数组
print(np.unique(arr))                           #打印去重复之后的 arr 数组
```

运行结果如下：

```
[1 2 5 9]
```

如果对多维数组做去重复处理，最后返回的是具有唯一值的一维数组，示例代码如下：

```
#chapter3\3-3\3-3-7.ipynb
import numpy as np                                    #导入 NumPy 库，并命名为 np
arr = np.array([[2,1,1],[2,5,1],[3,1,7]])             #arr 多维数组
print(np.unique(arr))                                 #打印去重复之后的 arr 数组
```

运行结果如下：

```
[1 2 3 5 7]
```

3.4　NumPy 数组维度转换

在做数据处理时，数据的呈现方式可能并不是用户希望的，这就需要用户对数据进行转换，本节讲解不同维度数组的相互转换，以及不同维度数组的合并。学会这些操作，用户在转换数据结构时将更加得心应手。

3.4.1　数组维度转换

本节讲解一维数组与多维数组的相互转换，本质就是数据的重新组合。用户如果需要对一组数据拆解或者合并，维度转换是不错的选择。

1. 一维数组转换为多维数组

数组的维度转换使用 reshape()函数，下面演示一下将一维数组转换为二维数组和三维数组，示例代码如下：

```
#chapter3\3-4\3-4-1.ipynb
import numpy as np                                    #导入 NumPy 库，并命名为 np
arr = np.arange(1,13)                                 #生成 1～16 连续值数组
print(arr.reshape(3,4))                               #打印生成的二维数组
print(arr.reshape(3,2,2))                             #打印生成的三维数组
```

运行结果如下：

```
#生成的二维数组结果如下
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]

#生成的三维数组结果如下
[[[ 1  2]
```

```
  [ 3  4]]
 [[ 5  6]
  [ 7  8]]
 [[ 9  10]
  [11  12]]]
```

2. 多维数组转换为多维数组

使用 reshape()函数也可以将多维数组转换为多维数组，例如可以将二维数组转换为另一种尺寸的二维数组，或者转换为其他多维数组，示例代码如下：

```
#chapter3\3-4\3-4-2.ipynb
import numpy as np                                          #导入 NumPy 库,并命名为 np
arr = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])       #arr 二维数组
print(arr.reshape(2,6))                                     #打印生成另一种二维数组
print(arr.reshape(2,2,3))                                   #打印生成的三维数组
```

运行结果如下：

```
#生成的另一种二维数组结果如下
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]

#生成的三维数组结果如下
[[[ 1  2  3]
  [ 4  5  6]]
 [[ 7  8  9]
  [10 11 12]]]
```

3. 多维数组转换为一维数组

将多维数组转换为一维数组，除了使用 reshape()函数之外，也可以使用 flatten()函数，并且这种转换方式更为直接，示例代码如下：

```
#chapter3\3-4\3-4-3.ipynb
import numpy as np                                  #导入 NumPy 库,并命名为 np
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])          #arr 二维数组
print(arr.reshape(arr.size))                        #计算多维数组元素个数,转换为一维数组
print(arr.flatten())                                #直接使用 NumPy 内置函数转换为一维数组
```

运行结果如下：

```
[1  2  3  4  5  6  7  8  9]
[1  2  3  4  5  6  7  8  9]
```

注意：数组之间的转换要遵循的原则是转换后的元素个数必须与转换前的元素个数相同，否则转换不成功。

3.4.2　数组合并

数组合并就是对两个及以上的数组做拼接，本节将讲解多个一维数组的合并，以及多个多维数组的合并。这些操作对数据的合并处理将非常有用。

1. 一维数组合并

如果要将多个一维数组合并成一个一维数组，首先要将多个一维数组组织在列表中，然后使用 np.concatenate()函数对列表中的一维数组合并，示例代码如下：

```
#chapter3\3-4\3-4-4.ipynb
import numpy as np                          #导入 NumPy 库，并命名为 np
arr1 = np.array([1,2,3])                    #arr1 一维数组
arr2 = np.array([4,5,6])                    #arr2 一维数组
lst = [arr1,arr2]                           #将 arr1 和 arr2 组合到列表中
arr3 = np.concatenate(lst)                  #合并处理
print(arr3)                                 #打印合并后的 arr3 数组
```

运行结果如下：

```
[1  2  3  4  5  6]
```

2. 多维数组合并

合并多个多维数组与合并多个一维数组方法基本相同，但由于多维数组合并需要用户确认是横向合并，还是纵向合并，所以要在 np.concatenate()函数中对 axis 参数指明合并方向，axis＝1 表示横向合并，axis＝0 表示纵向合并，示例代码如下：

```
#chapter3\3-4\3-4-5.ipynb
import numpy as np                          #导入 NumPy 库，并命名为 np
arr1 = np.array([[1,2,3],[4,5,6]])          #arr1 多维数组
arr2 = np.array([[7,8,9],[10,12,13]])       #arr2 多维数组
lst = [arr1,arr2]                           #将 arr1 和 arr2 组合到列表中
arr3 = np.concatenate(lst,axis = 1)         #横向合并
print(arr3)                                 #打印横向合并后的 arr3 数组
arr4 = np.concatenate(lst,axis = 0)         #纵向合并
print(arr4)                                 #打印纵向合并后的 arr4 数组
```

运行结果如下：

```
#横向合并多维数组返回结果如下所示
[[ 1  2  3  7  8  9]
 [ 4  5  6 10 12 13]]

#纵向合并多维数组返回结果如下所示
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 12 13]]
```

3.5 Series 数据的创建

Series 可以视为 DataFrame 表格的列，之前的 Series 数据都是通过拆解 DataFrame 表格获取的，如果需要用户创建，则可以使用 pd.Series()函数，该函数的参数说明如下所示。

pd.Series(data=None,index=None,dtype=None,name=None,copy=False)

data：提供创建 Series 的数据，可以是列表、数组和字典等可迭代对象。

index：提供 Series 的索引数据，允许有重复值，默认为 RangeIndex(0,1,2,…,n)。

dtype：设置 Series 数据的类型，如未指定则自动推断，关于数据类型可参考表 3-1。

name：设置 Series 数据的名称。

copy：是否复制输入数据。

接下来演示一下使用 pd.Series()函数在创建 Series 数据时的各种设置。

1. 创建 Series 数据

在创建 Series 数据时，可以使用列表、数组等可迭代对象作为 pd.Series()函数的 data 参数，以列表为例，示例代码如下：

```
#chapter3\3-5\3-5-1.ipynb
import pandas as pd                                          #导入 Pandas 库,并命名为 pd
s = pd.Series(['张三','李四','王麻子'],name = '姓名')          #用列表创建 Series,并设置名称
s                                                            #返回 Series 数据
```

运行结果如下：

```
0 张三
1 李四
2 王麻子
Name:姓名, dtype: object
```

2. 用字典创建 Series 数据

如果使用字典作为 pd.Series()函数的 data 参数，则字典的键对应设置为 Series 数据的索引，示例代码如下：

```
#chapter3\3-5\3-5-2.ipynb
import pandas as pd                                          #导入 Pandas 库,并命名为 pd
s = pd.Series({'A':'张三','B':'李四','C':'王麻子'})           #用字典创建 Series
s                                                            #返回 Series 数据
```

运行结果如下：

```
A 张三
B 李四
C 王麻子
dtype: object
```

3. 创建 Series 数据时设置索引

如果在创建 Series 数据时，需要设置对应的索引，则可以在 index 参数中设置，示例代码如下：

```
#chapter3\3-5\3-5-3.ipynb
import pandas as pd                                                    #导入 Pandas 库,并命名为 pd
s = pd.Series(['张三','李四','王麻子'],index = ['A','B','C'])          #创建 Series 时设置索引
s                                                                      #返回 Series 数据
```

运行结果如下：

```
A 张三
B 李四
C 王麻子
dtype: object
```

4. 创建 Series 数据时设置类型

如果在创建 Series 数据时，需要设置数据类型，则可以在 dtype 参数中设置，示例代码如下：

```
#chapter3\3-5\3-5-4.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
s = pd.Series([2,3,5],dtype = 'float')                 #创建字典时设置数据类型
s                                                      #返回 Series 数据
```

运行结果如下：

```
0    2.0
1    3.0
2    5.0
dtype: float64
```

3.6　DataFrame 表格的创建

本应该在 2.5 节讲解 DataFrame 表格的创建，为什么要放在本章的最后一节学习呢？原因在于之前没有讲解 NumPy 数组，而用数组创建 DataFrame 表又是非常重要的方式，因此在讲解数组之后，再系统地讲解 DataFrame 表格的创建。

虽然我们对 DataFrame 表格结构已不陌生，因为在前面的章节中已学习过通过获取外部文件数据来生成 DataFrame 表格，但还没尝试通过创建的方式来生成 DataFrame 表格。在实际的数据处理环境中，可能会经常构造 DataFrame 表格，所以非常有必要学习 DataFrame 表格的创建，创建 DataFrame 表格使用 pd.DataFrame()函数，该函数的参数说明如下所示。

pd.DataFrame(data＝None,index＝None,columns＝None, dtype＝None, copy＝False)

data：提供创建 DataFrame 表格的数据，可以为数组、列表和字典。
index：提供 DataFrame 表格的行索引数据，默认为 RangeIndex(0,1,2,…,n)。
columns：提供 DataFrame 表格的列索引数据，默认为 RangeIndex(0,1,2,…,n)。
dtype：数据类型，只允许设置单个数据类型，如果没有设置则自动推断。
copy：是否从输入复制数据。仅影响 DataFrame/二维数组输入。

3.6.1 使用 NumPy 数组创建 DataFrame 表格

将提供的数组放置在 pd.DataFrame()函数的 data 参数中，并且在 columns 参数中指定表的列索引(列标题)，示例代码如下：

```
#chapter3\3-6\3-6-1.ipynb
import numpy as np,pandas as pd          #导入 NumPy 库和 Pandas 库,并分别命名为 np 和 pd
arr = np.array([
    ['张三','男',28],
    ['李四','女',25],
    ['王五','女',19]
])                                       #二维数组
df = pd.DataFrame(
    data = arr,                          #提供的二维数组
    columns = ['姓名','性别','年龄']      #设置列索引
)
df                                       #返回 DataFrame 表格
```

运行结果如图 3-4 所示。

	姓名	性别	年龄
0	张三	男	28
1	李四	女	25
2	王五	女	19

图 3-4 使用 NumPy 数组创建 DataFrame 表格

3.6.2 使用 Python 列表创建 DataFrame 表格

将提供的列表放置在 pd.DataFrame()函数的 data 参数中，并且在 columns 参数中指定表的列索引(列标题)，示例代码如下：

```
#chapter3\3-6\3-6-2.ipynb
import numpy as np,pandas as pd          #导入 NumPy 库和 Pandas 库,并分别命名为 np 和 pd
lst = [
    ['张三','男',28],
    ['李四','女',25],
    ['王五','女',19]
]                                        #嵌套的列表
df = pd.DataFrame(
    data = lst,                          #提供的列表
    columns = ['姓名','性别','年龄']      #设置列索引
)
df                                       #返回 DataFrame 表格
```

运行结果如图 3-5 所示。

	姓名	性别	年龄
0	张三	男	28
1	李四	女	25
2	王五	女	19

图 3-5　使用 Python 列表创建 DataFrame 表格

3.6.3　使用 Python 字典创建 DataFrame 表格

Python 中的字典是由键值对组成的，也可以使用字典创建 DataFram 表格，并且创建方式更为多样化。用字典创建 DataFrame 表格的固定格式是：{列索引:列数据}，也就是说字典的键对应 DataFrame 表格的列索引，值对应列数据，列数据可以是列表、数组和 Series。由于字典的键是 DataFrame 表格的列索引，所以不需要在 columns 参数中指定列索引，不过可以设置 index 参数，这样 DataFrame 表格的行索引和列索引就都是自定义的了。

1. 字典值为列表

字典的值是列表，创建 DataFrame 表格时，示例代码如下：

```
#chapter3\3-6\3-6-3.ipynb
import numpy as np,pandas as pd                    #导入 NumPy 库和 Pandas 库,并分别命名为 np 和 pd
dic = {
    '姓名':['张三','李四','王五'],
     '性别':['男','女','女'],
     '年龄':[28,25,19]
}                                                  #值为列表的字典
df = pd.DataFrame(
    data = dic,                                    #提供的字典
    index = ['NED001','NED002','NED003']           #设置行索引
)
df                                                 #返回 DataFrame 表格
```

运行结果如图 3-6 所示。

	姓名	性别	年龄
NED001	张三	男	28
NED002	李四	女	25
NED003	王五	女	19

图 3-6　字典值为列表，创建 DataFrame 表格

2. 字典值为数组

字典的值是数组，创建 DataFrame 表格时，示例代码如下：

```
#chapter3\3-6\3-6-4.ipynb
import numpy as np                                 #导入 NumPy 库,并命名为 np
import pandas as pd                                #导入 Pandas 库,并命名为 pd
dic = {
    '姓名':np.array(['张三','李四','王五']),
```

```
    '性别':np.array(['男','女','女']),
    '年龄':np.array([23,25,19])
}                                                    #值为数组的字典
df = pd.DataFrame(
    data = dic,                                      #提供的字典
    index = ['NED001','NED002','NED003']             #设置行索引
)
df                                                   #返回 DataFrame 表格
```

运行结果如图 3-7 所示。

	姓名	性别	年龄
NED001	张三	男	28
NED002	李四	女	25
NED003	王五	女	19

图 3-7　字典值为数组，创建 DataFrame 表格

3. 字典值为 Series

字典的值是 Series，因为 Series 数据只是加了索引的数组而已，所以 Series 的索引就是 DataFrame 表格的行索引。pd.DataFrame()函数的 index 和 columns 两个参数都不用设置，也能达到自定义行索引和列索引的效果，示例代码如下：

```
#chapter3\3-6\3-6-5.ipynb
import numpy as np                                   #导入 NumPy 库,并命名为 np
import pandas as pd                                  #导入 Pandas 库,并命名为 pd
dic = {
    '姓名':pd.Series(['张三','李四','王五'],['NED001','NED002','NED003']),
    '性别':pd.Series(['男','女','女'],['NED001','NED002','NED003']),
    '年龄':pd.Series([23,25,19],['NED001','NED002','NED003'])
}                                                    #值为 Series 数据的字典
df = pd.DataFrame(data = dic)                        #提供的字典
df                                                   #返回 DataFrame 表格
```

运行结果如图 3-8 所示。

	姓名	性别	年龄
NED001	张三	男	28
NED002	李四	女	25
NED003	王五	女	19

图 3-8　字典值为 Series，创建 DataFrame 表格