

Python 数据分析与科学计算基础

篇 2

Pandas 图解

第1章 Pandas 简介

1.1 学习 Pandas 前置知识点

- 1、Python 基础语言
- 2、重点 Python 内置数据结构：列表 (list)、字符串 (str)、元组 (tuple)、集合 (set) 和字典 (dict)。
- 3、序列的索引和切片操作。
- 4、列表推导式、集合推导式、字典推导式
- 5、lambda 表达式
- 6、Python 三个函数式编程基础的函数：filter()、map() 和 reduce()。
- 7、NumPy 库

1.2 pandas 是什么？

pandas 是一个开源的 Python 数据分析库。

1、pandas 基于 NumPy 库，整合 NumPy、SciPy (科学计算) 和 Matplotlib (绘图库) 功能。

2、pandas 官网：<http://pandas.pydata.org/>

3、pandas 源代码：<https://github.com/pandas-dev/pandas>

4、pandas 一词源自 “Python and data analysis” 和 “panel data” 字

母缩略写。

5、pandas 广泛应用学术和商业领域, 包括金融, 经济学, 统计学, 广告, 网络分析等。

1.3 为什么选择 pandas

1、Python 写出易读、整洁并且缺陷最少的代码。

2、使用 pandas 可以完成数据处理和分析中的五个典型步骤: 数据加载、数据准备、数据操作、数据建模和数据分析。

3、pandas 提供了快速高效的 Series 和 DataFrame 数据结构。

4、pandas 数据结构基于 NumPy 数组, 而 NumPy 底层是用 C 语言实现速度快。

5、可以加载来自不同文件格式的数据到内存中。

6、可以处理数据对齐和缺失数据。

7、支持基于标签(索引)下标和切片操作, 可以处理大数据集。

8、按数据分组以进行聚合和转换。

9、高性能的数据合并和连接。

10、支持时间序列功能。

1.4 课后练习

1、访问 pandas 官网。

2、使用《pandas 用户指南文档》。

3、使用《pandas API 文档》。

第2章 环境搭建

2.1 环境方案 1：手动安装

Python 要求 Python 3.5 以及以上版本。

2.1.1 使用 pip 安装 pandas 等库

还需要安装的库 numpy、scipy、matplotlib。

在 Windows 命令提示符中使用 pip 安装指令：

```
pip install pandas
```

2.1.2 安装 IPython

IPython 是一个加强版本的 Python 解释器。

在命令提示符或终端中使用 pip 安装 IPython 指令：

```
pip install ipython
```

2.2 环境方案 2：安装 Anaconda

Anaconda 指的是一个开源的 Python 发行版本，其包含了 conda、Python 等 1500 多个科学包。

1、Anaconda 官网：<https://www.anaconda.com>

2、清华大学开源软件镜像站：

<https://mirrors.tuna.tsinghua.edu.cn/help/anaconda/>

❑ Anaconda 安装包：

<https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/>

❑ Miniconda 安装包，只包含了 python 和 conda：

<https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/>

2.2.1 Windows 安装 Anaconda

<https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/>

2.2.2 Linux 安装 Anaconda

<https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/>

2.2.3 macOS 安装 Anaconda

<https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/>

2.3 开发工具

Python shell、Python IDEL、IPython shell 、IDE 工具 (Pycharm、Eclipse Pydev 插件、Visual Studio Code、Spyder) 、Jupyter Notebook

2.3.1 IPython shell

1、启动

在命令提示符或终端中使用如下指令：

```
ipython
```

2、获得帮助

通过?或??获取获得帮助，示例如下：

3、用 Tab 键语法补全

示例如下：

4、IPython shell 中的快捷键

表 2-1 IPython shell 中常用的快捷键

快捷键	说明
Ctrl + p (或向上箭头)	获取前一个历史命令
Ctrl + n (或向下箭头)	获取后一个历史命令
Ctrl + r	对历史命令的反向搜索
Ctrl + L	清除终端屏幕的内容
Ctrl + d	退出 IPython 会话

示例如下：

5、IPython shell 魔法命令

表 2-2 IPython shell 常用魔法命令

魔法命令	说明
%run	执行外部代码
%timeit	计算代码运行时间
%magic	获得所有可用魔法命令的列表
?	某个魔法命令帮助

示例如下：

2.3.2 Jupyter Notebook

Jupyter Notebook 是 IPython shell 基于浏览器的图形界面。Jupyter Notebook 不仅可以执行 Python/IPython 语句，还允许用户编写科技文章。

1、安装

- ☐ Anaconda 完整安装包括 Jupyter Notebook。
- ☐ 手动使用 pip 安装指令：

```
pip install jupyter
```

2、启动

在命令提示符或终端中使用如下指令：

```
jupyter notebook
```

示例如下：

2.3.3 Spyder

Spyder(Scientific Python Development Environment)是一个强大的交互式 Python 语言 IDE 开发环境, 支持包括 Windows、Linux 和 macOS 系统。Spyder 还集成了很多流行科学软件包, 包括 NumPy, SciPy, Pandas, IPython, QtConsole, Matplotlib, SymPy 等。

Spyder 官网: www.spyder-ide.org

1、安装

- ❑ Anaconda 完整安装包括 Spyder。
- ❑ 手动使用 pip 安装指令:

```
pip install spyder
```

手动使用安装 Spyder 不推荐!!!!

2、基本用法

2.4 课后练习

- 1、如何使用 pip 或 conda 指令查询 pandas 版本。
- 2、写一个程序来获取 pandas 版本。

参考答案:

```
import pandas as pd
print(pd.__version__)
```


第3章 Series 数据结构

3.1 Pandas 数据结构概述

两种主要的数据结构：

- ❑ 一维数据结构 Series
- ❑ 二维数据结构 DataFrame

这些数据结构都是带有索引(标签)的，DataFrame 是由 Series 构成。

Series			Series			DataFrame		
	apples			oranges			apples	oranges
0	3	+	0	0	=	0	3	0
1	2		1	3		1	2	3
2	0		2	7		2	0	7
3	1		3	2		3	1	2

数据结构的操作如下：

插入、删除、修改、查询、分组、连接、合并、汇总

3.2 理解 Series 数据结构

- ❑ Series 结构是一种带有标签一维数组对象。
- ❑ 能够保存任何数据类型。
- ❑ 一个 Series 对象又包含两数组：数据和数据索引（即标签）。
- ❑ 数据部分是 numpy 的数组（ndarray）类型。

	apples
0	3
1	2
2	0
3	1

索引 数据

图 3-1 Series 结构

```
In [49]: import pandas as pd

In [50]: apples = pd.Series([3,2,0,1])

In [51]: apples
Out[51]:
0    3
1    2
2    0
3    1
dtype: int64

In [52]: type(apples)
Out[52]: pandas.core.series.Series

In [53]: type(apples.values)
Out[53]: numpy.ndarray

In [54]: type(apples.index)
Out[54]: pandas.core.indexes.range.RangeIndex
```

3.3 创建 Series 对象

Series 构造函数语法格式如下:

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

- ❑ data 是 Series 数据部分, 可以是列表、NumPy 数组、标量值 (常数)、字典。
- ❑ index 是 Series 索引 (即标签) 部分, 与数据的长度相同。默认 `np.arange(n)`。
- ❑ dtype 用于数据类型。如果没有则推断数据类型。

1、使用列表创建 Series:

```
In [49]: import pandas as pd
In [50]: apples = pd.Series([3,2,0,1])
In [51]: apples
Out[51]:
0    3
1    2
2    0
3    1
dtype: int64
```

2、使用 NumPy 数组创建 Series:

```
In [49]: import pandas as pd
In [50]: import numpy as np
In [51]: a = np.array([3,2,0,1])
In [52]: apples = pd.Series(a)
In [53]: apples
Out[54]:
0    3
1    2
```

```
2    0
3    1
dtype: int64
```

3、指定索引:

```
In [27]: apples = pd.Series([3,2,0,1], index=['a','b','c','d'])

In [28]: apples
Out[28]:
a    3
b    2
c    0
d    1
dtype: int64
```

4、使用标量创建 Series:

```
In [29]: apples = pd.Series(2, index=['a','b','c','d'])
...: apples
Out[29]:
a    2
b    2
c    2
d    2
dtype: int64
```

5、使用字典创建 Series:

```
In [31]: data = {'a' : 3, 'b' : 2, 'c' : 0, 'd' : 1}

In [32]: apples = pd.Series(data)

In [33]: apples
Out[33]:
```

```
a    3
b    2
c    0
d    1
dtype: int64
```

6、指定索引与数据数组长度不同:

```
In [34]: data = {'a' : 3, 'b' : 2, 'c' : 0}

In [35]: apples = pd.Series(data, index=['a','b','c','d'])

In [36]: apples
Out[36]:
a    3.0
b    2.0
c    0.0
d    NaN
dtype: float64
```

提示 指定索引与数据数组长度不同时, 按照索引长度创建, 不足的数据部分用NaN补齐。

3.4 访问 Series 数据

3.4.1 Series 标签与位置区别

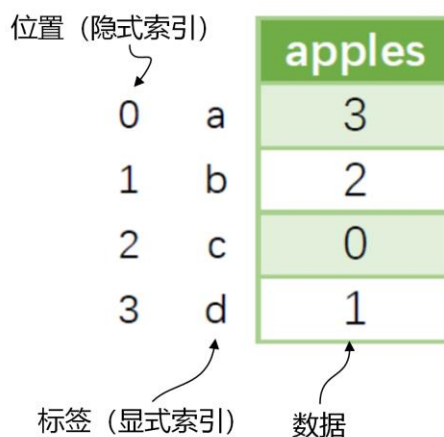


图 3-2 标签和位置

3.4.2 通过下标访问 Series 数据

1、通过标签下标访问数据：

```
In [43]: data = {'a' : 3, 'b' : 2, 'c' : 0, 'd' : 1}
...: apples = pd.Series(data)
...: # 通过标签下标访问数据
...: apples['a']
...:
Out[43]: 3
```

2、通过位置下标访问数据：

```
In [41]: apples[0]
Out[41]: 3
```


3.4.3 通过切片访问 Series 数据

1、通过标签切片访问数据：

```
In [60]: apples['a':'c']
```

```
Out[60]:
```

```
a    3
```

```
b    2
```

```
c    0
```

```
dtype: int64
```

```
In [61]: apples['a':'d']
```

```
Out[61]:
```

```
a    3
```

```
b    2
```

```
c    0
```

```
d    1
```

```
dtype: int64
```

```
In [62]: apples[:, 'd']
```

```
Out[62]:
```

```
a    3
```

```
b    2
```

```
c    0
```

```
d    1
```

提示 标签切片包括结束标签数据。

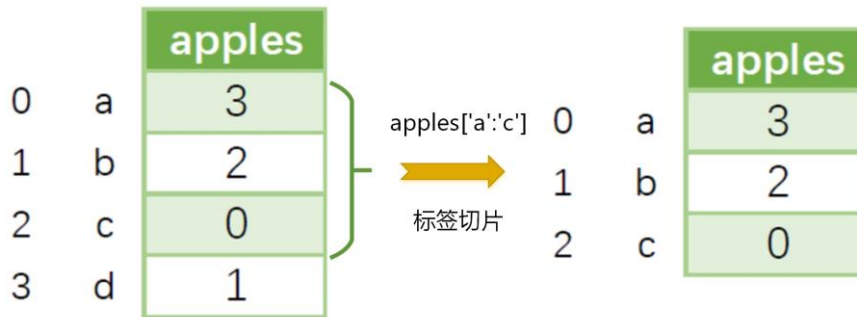


图 3-3 标签切片

2、通过位置切片访问数据：

```
In [63]: apples[:3]
Out[63]:
a    3
b    2
c    0
dtype: int64

In [66]: apples[0:3]
Out[66]:
a    3
b    2
c    0
dtype: int64
```

提示 位置切片不包括结束位置数据。

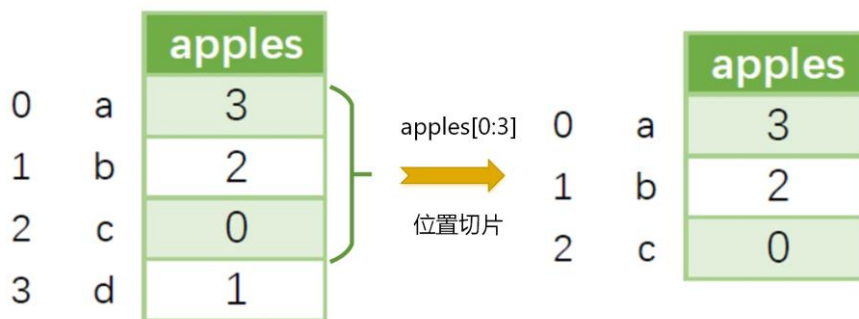


图 3-4 位置切片

3.4.4 通过布尔数组访问 Series 数据

使用布尔数组（或列表）可以访问 Series 数据。

```
In [71]: b = [True, False, True, False]
```

```
In [72]: apples[b]
```

```
Out[72]:
```

```
a    3
```

```
c    0
```

```
dtype: int64
```

```
In [73]: apples[[True, False, True, False]]
```

```
Out[73]:
```

```
a    3
```

```
c    0
```

```
dtype: int64
```

```
In [74]: apples[ apples >=2 ]
```

```
Out[74]:
```

```
a    3
```

```
b    2
```

```
dtype: int64
```

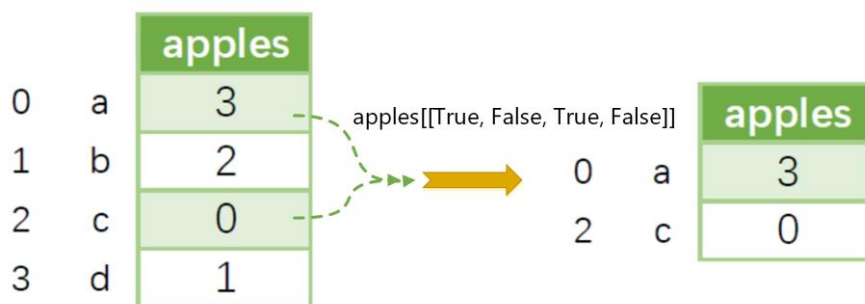


图 3-5 布尔下标访问

注意 不能是`apples[True, False, True, False]`

3.4.5 通过花式下标访问 Series 数据

使用**标签数组（或列表）或位置数组（或列表）**，作为下标，这称为“花式下标”。

1、**标签数组（或列表）**“花式下标”

```
In [79]: apples[['b','d']]
Out[79]:
b    2
d    1
dtype: int64
```

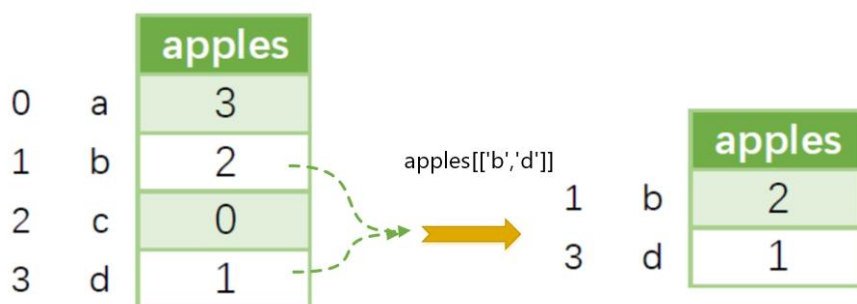


图 3-6 标签花式下标访问

注意 不能是`apples['b','d']`

2、位置数组（或列表）“花式下标”

```
In [80]: apples[[1,3]]
Out[80]:
b    2
d    1
dtype: int64
```

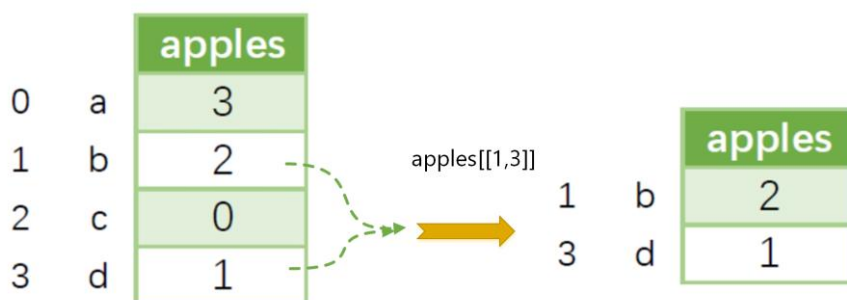


图 3-7 位置花式下标访问

注意 不能是`apples[1,3]`

3.5 课后练习

1、如何从列表、NumPy 数组和字典创建一个 Series ？

参考答案：

```
import pandas as pd
import numpy as np
mylist = list('abcdefghijklmnopqrstuvwxyz')
myarr = np.arange(26)
mydict = dict(zip(mylist, myarr))
ser = pd.Series(mydict)
```

2、编写一个 pandas 程序，根据给定条件获得 Series 子集。

参考答案：

```
import pandas as pd
s = pd.Series([0, 1,2,3,4,5,6,7,8,9,10])
print("原始数据：")
print(s)
print("获得子集：")
new_s = s[s < 6]
print(new_s)
```

3、编写一个 pandas 程序，将 NumPy 数组转换为 Series 对象。

参考答案：

```
import numpy as np
import pandas as pd
np_array = np.array([10, 20, 30, 40, 50])
print("NumPy array: ")
```

```
print(np_array)
new_series = pd.Series(np_array)
print("转换后的Series对象: ")
print(new_series)
```

4、编写一个 pandas 程序将字典转换为 Series 对象。

参考答案:

```
import pandas as pd
d1 = {'a': 100, 'b': 200, 'c':300, 'd':400, 'e':800}
print("原始数据: ")
print(d1)
new_series = pd.Series(d1)
print("转换后的Series对象: ")
print(new_series)
```

第4章 DataFrame 数据结构

4.1 理解 DataFrame 数据结构

- ❑ 由多个 Series 结构构成二维表格对象。
- ❑ 每一个列可以不同数据类型。
- ❑ 行和列是带有标签的轴。
- ❑ 行和列可变的。

The diagram illustrates the DataFrame structure as a 2D table. The columns are labeled 'apples', 'oranges', and 'bananas' at the top. The rows are indexed 0, 1, 2, and 3 on the left. Arrows point to the column headers with the label '列标签 (列索引)', to the row indices with '行标签 (行索引)', and to a data cell with '数据'.

	apples	oranges	bananas
0	3	0	1
1	2	1	2
2	0	2	1
3	1	3	0

图 4-1 DataFrame 数据结构

4.2 创建 DataFrame 对象

DataFrame 构造函数语法格式如下：

```
pandas.DataFrame( data, index, columns, dtype, ...)
```


- ❑ data 是 DataFrame 数据部分，可以是列表、NumPy 数组、字典、Series 对象和其他的 DataFrame 对象。
- ❑ index 是行索引（即行标签），默认 np.arange(n)。
- ❑ columns 是列索引（即列标签），默认 np.arange(n)。
- ❑ dtype 用于数据类型。如果没有则推断数据类型。

1、使用列表创建 DataFrame：

```
In [10]: L = [[3,0,1], [2,1,2], [0,2,1], [1,3,0]]
```

```
In [11]: df = pd.DataFrame(L)
```

```
In [12]: df
```

```
Out[12]:
```

```
   0  1  2
0  3  0  1
1  2  1  2
2  0  2  1
3  1  3  0
```



	0	1	2
0	3	0	1
1	2	1	2
2	0	2	1
3	1	3	0

图 4-2 默认行标签和列标签

2、指定行标签和列标签：

```
In [8]: L = [[3,0,1],
...:        [2,1,2],
...:        [0,2,1],
...:        [1,3,0]]
```

```
In [9]: df = pd.DataFrame(L, columns=['apples', 'oranges', 'bananas'])
```

```
In [10]: df
```

```
Out[10]:
```

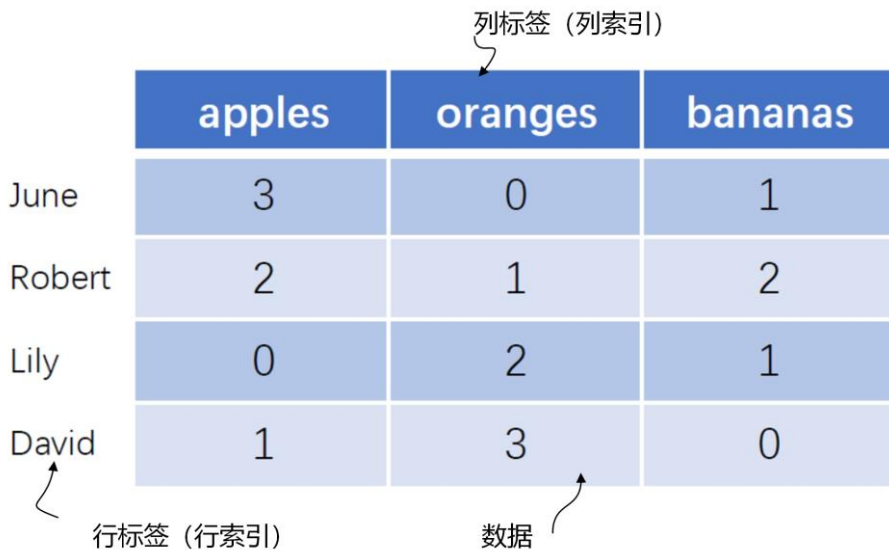
	apples	oranges	bananas
0	3	0	1
1	2	1	2
2	0	2	1
3	1	3	0

```
In [11]: df = pd.DataFrame(L,
...:                        columns=['apples', 'oranges', 'bananas'],
...:                        index=['June', 'Robert', 'Lily', 'David'])
```

```
In [12]: df
```

```
Out[12]:
```

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0



The diagram shows a pandas DataFrame with four rows and three columns. The columns are labeled 'apples', 'oranges', and 'bananas'. The rows are labeled 'June', 'Robert', 'Lily', and 'David'. The data values are as follows:

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0

Annotations in the diagram:

- An arrow points to the column headers with the label "列标签 (列索引)".
- An arrow points to the row labels with the label "行标签 (行索引)".
- An arrow points to the data values in the 'oranges' column with the label "数据".

图 4-3 指定行标签和列标签

3、使用字典创建 DataFrame:

```
In [11]: data = {
...:     'apples': [3, 2, 0, 1],
...:     'oranges': [0, 1, 2, 3],
...:     'bananas': [1, 2, 1, 0]
...: }
```

```
In [12]: df = pd.DataFrame(data)
```

```
In [13]: df
Out[13]:
```

	apples	oranges	bananas
0	3	0	1
1	2	1	2
2	0	2	1
3	1	3	0

```
In [14]: df = pd.DataFrame(data,
...:     index=['June', 'Robert', 'Lily', 'David'])
```

```
In [15]: df
```

```
Out[15]:
```

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0

4、使用列表嵌套字典创建 DataFrame：

```
In [13]: data = [ {'apples': 3, 'oranges': 0, 'bananas': 1},
...:              {'apples': 2, 'oranges': 1, 'bananas': 2},
...:              {'apples': 0, 'oranges': 2, 'bananas': 1},
...:              {'apples': 1, 'oranges': 3, 'bananas': 0} ]
...:
```

```
In [14]: df = pd.DataFrame(data)
```

```
In [15]: df
```

```
Out[15]:
```

	apples	bananas	oranges
0	3	1	0
1	2	2	1
2	0	1	2
3	1	0	3

```
In [16]: df = pd.DataFrame(data,
...:                        index=['June', 'Robert', 'Lily', 'David'])
```

```
In [17]: df
```

```
Out[17]:
```

	apples	bananas	oranges
June	3	1	0
Robert	2	2	1
Lily	0	1	2
David	1	0	3

5、使用字典嵌套 Series 创建 DataFrame:

```
In [10]: data = {
...:     'apples': pd.Series([3, 2, 0, 1]),
...:     'oranges': pd.Series([0, 1, 2, 3]),
...:     'bananas': pd.Series([1, 2, 1, 0])
...: }
...:
```

```
In [11]: df = pd.DataFrame(data)
```

```
In [12]: df
Out[12]:
```

	apples	oranges	bananas
0	3	0	1
1	2	1	2
2	0	2	1
3	1	3	0

```
In [13]: data = {
...:     'apples': pd.Series([3, 2, 0, 1], index=['June', 'Robert', 'Lily', 'David']),
...:     'oranges': pd.Series([0, 1, 2, 3], index=['June', 'Robert', 'Lily', 'David']),
...:     'bananas': pd.Series([1, 2, 1, 0], index=['June', 'Robert', 'Lily', 'David'])
...: }
```

```
In [14]: df = pd.DataFrame(data)
```

```
In [15]: df
Out[15]:
```

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0

4.3 DataFrame 标签与位置区别

		列位置 (隐式列索引)	列标签 (显式列索引)	
		0	1	2
		apples	oranges	bananas
0	June	3	0	1
1	Robert	2	1	2
2	Lily	0	2	1
3	David	1	3	0

图 4-4 标签和位置

4.4 访问 DataFrame 列

使用[]运算符访问 DataFrame 列，有两种主要形式：

- ❑ 单个标签下标，返回表示某列的 Series 对象。
- ❑ 多个标签列表（或数组）下标，返回包含多列的 DataFrame 对象。

提示 访问列是不能使用位置下标，只能使用标签下标。

4.4.1 单个标签下标访问 DataFrame 列

1、使用默认列标签：

```
In [30]: L = [[3,0,1],
...:         [2,1,2],
...:         [0,2,1],
...:         [1,3,0]]
...: df = pd.DataFrame(L)
...: df
Out[30]:
   0  1  2
0  3  0  1
1  2  1  2
2  0  2  1
3  1  3  0

In [31]: df[0]
Out[31]:
0    3
1    2
2    0
3    1
Name: 0, dtype: int64

In [32]: type(df[0])
Out[32]: pandas.core.series.Series
```

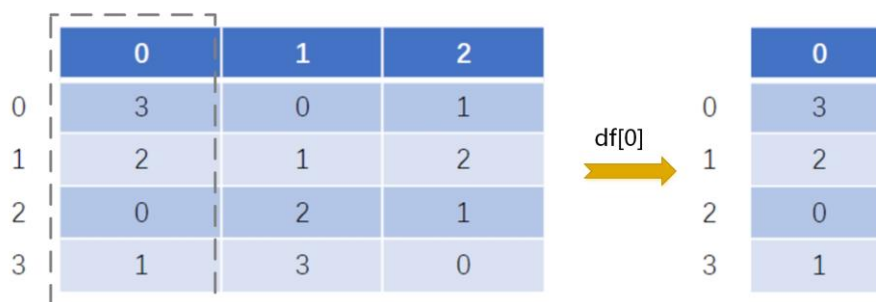


图 4-5 默认列标签

2、使用指定列标签：

```
In [33]: data = {
...:     'apples': [3, 2, 0, 1],
...:     'oranges': [0, 1, 2, 3],
...:     'bananas': [1, 2, 1, 0]
...: }
...: df = pd.DataFrame(data,
...: index=['June', 'Robert', 'Lily', 'David'])
...: df
```

```
Out[33]:
```

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0

```
In [34]: df['apples']
```

```
Out[34]:
```

June	3
Robert	2
Lily	0
David	1

Name: apples, dtype: int64

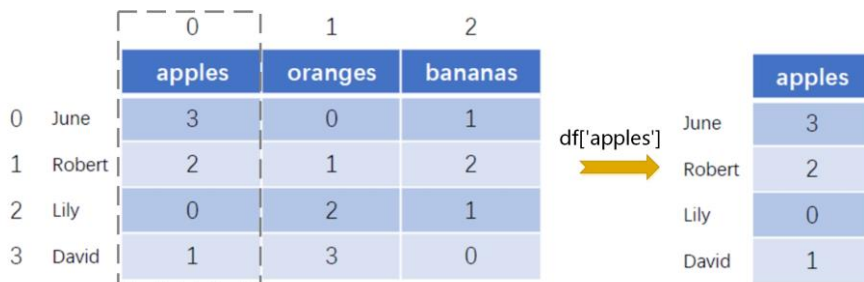


图 4-6 指定列标签

4.4.2 多个标签下标访问 DataFrame 列

1、使用默认列标签:

```
In [30]: L = [[3,0,1],
...:         [2,1,2],
...:         [0,2,1],
...:         [1,3,0]]
...: df = pd.DataFrame(L)
...: df
Out[30]:
   0  1  2
0  3  0  1
1  2  1  2
2  0  2  1
3  1  3  0

In [31]: df[[0,2]]
Out[31]:
   0  2
0  3  1
1  2  2
2  0  1
3  1  0

In [32]: type(df[[0,2]])
Out[32]: pandas.core.frame.DataFrame
```

	0	1	2			0	2
0	3	0	1	df[[0,2]] →	0	3	1
1	2	1	2		1	2	2
2	0	2	1		2	0	1
3	1	3	0		3	1	0

图 4-7 默认列标签

2、使用指定列标签：

```
In [33]: data = {
...:     'apples': [3, 2, 0, 1],
...:     'oranges': [0, 1, 2, 3],
...:     'bananas': [1, 2, 1, 0]
...: }
...: df = pd.DataFrame(data,
...: index=['June', 'Robert', 'Lily', 'David'])
...: df
```

```
Out[33]:
      apples  oranges  bananas
June        3        0        1
Robert       2        1        2
Lily         0        2        1
David        1        3        0
```

```
In [39]: df[['apples', 'bananas']]
```

```
Out[39]:
      apples  bananas
June        3        1
Robert       2        2
Lily         0        1
David        1        0
```

		0	1	2
		apples	oranges	bananas
0	June	3	0	1
1	Robert	2	1	2
2	Lily	0	2	1
3	David	1	3	0

→

df[['apples', 'bananas']]			
		apples	bananas
	June	3	1
	Robert	2	2
	Lily	0	1
	David	1	0

图 4-8 指定列标签

4.5 访问 DataFrame 行

访问 **DataFrame** 行也可以使用 `[]` 运算符访问，有两种主要形式：

- ❑ 切片
- ❑ 布尔数组

4.5.1 通过切片访问 DataFrame 行

1、使用行位置切片：

```
In [30]: L = [[3,0,1],
...:         [2,1,2],
...:         [0,2,1],
...:         [1,3,0]]
...: df = pd.DataFrame(L)
...: df
Out[30]:
   0  1  2
0  3  0  1
1  2  1  2
2  0  2  1
```

```
3 1 3 0
```

```
In [31]: df[0:3]
```

```
Out[31]:
```

```
0 1 2
```

```
0 3 0 1
```

```
1 2 1 2
```

```
2 0 2 1
```

```
In [32]: df[:3]
```

```
Out[32]:
```

```
0 1 2
```

```
0 3 0 1
```

```
1 2 1 2
```

```
2 0 2 1
```

提示 行位置切片不包括结束行数据。



图 4-9 行位置切片

2、使用行标签切片：

```
In [33]: data = {
...:     'apples': [3, 2, 0, 1],
...:     'oranges': [0, 1, 2, 3],
...:     'bananas': [1, 2, 1, 0]
...: }
...: df = pd.DataFrame(data,
```

```
index=['June','Robert','Lily','David'])
...: df
Out[33]:
```

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0

```
In [34]: df['June':'David']
```

```
Out[34]:
```

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0

```
In [35]: df[0:3]
```

```
Out[35]:
```

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1

提示 行标签切片包括结束行数据。

	0	1	2			0	1	2
	apples	oranges	bananas			apples	oranges	bananas
June	3	0	1	→	June	3	0	1
Robert	2	1	2		Robert	2	1	2
Lily	0	2	1		Lily	0	2	1
David	1	3	0		David	1	3	0

图 4-10 行标签切片

4.5.2 通过布尔数组访问 DataFrame 行

```
In [33]: data = {
...:     'apples': [3, 2, 0, 1],
...:     'oranges': [0, 1, 2, 3],
...:     'bananas': [1, 2, 1, 0]
...: }
...: df = pd.DataFrame(data,
...:     index=['June', 'Robert', 'Lily', 'David'])
...: df
```

```
Out[33]:
```

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0

```
In [34]: df[[True, False, True, False]]
```

```
Out[34]:
```

	apples	oranges	bananas
June	3	0	1
Lily	0	2	1

```
In [35]: b = [True, False, True, False]
```

```
In [36]: df[b]
```

```
Out[36]:
```

	apples	oranges	bananas
June	3	0	1
Lily	0	2	1

	0	1	2	
	apples	oranges	bananas	
June	3	0	1	→
Robert	2	1	2	
Lily	0	2	1	
David	1	3	0	

df[[True, False, True, False]]			
	0	1	2
apples	oranges	bananas	
June	3	0	1
Lily	0	2	1

图 4-11 布尔数组访问

4.5.3 通过 query 方法访问 DataFrame 行

```
In [33]: data = {
...:     'apples': [3, 2, 0, 1],
...:     'oranges': [0, 1, 2, 3],
...:     'bananas': [1, 2, 1, 0]
...: }
...: df = pd.DataFrame(data,
...: index=['June', 'Robert', 'Lily', 'David'])
...: df
```

```
In [34]: df.query("apples > 1")
```

```
Out[34]:
```

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2

```
In [35]: df.query("apples > 1 and bananas <= 2")
```

```
Out[35]:
```

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2

```
In [36]: df.query("apples > 1 or bananas < 1")
```

```
Out[36]:
```

	apples	oranges	bananas
--	--------	---------	---------

June	3	0	1
Robert	2	1	2
David	1	3	0

	0	1	2
	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0

df.query("apples > 1 and bananas <= 2")

	0	1	2
	apples	oranges	bananas
0 June	3	0	1
2 Robert	2	1	2

图 4-12 query 方法访问

4.5.4 通过 head 和 tail 方法访问 DataFrame 行

- ❑ head(n) 方法，返回前 n 行，省略 n 返回前 5 行。
- ❑ tail(n) 方法，返回后 n 行，省略 n 返回后 5 行。

```
In [33]: df = pd.read_csv("planets.csv")
In [4]: df.head()
Out[4]:
```

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011
3	Radial Velocity	1	326.030	19.40	110.62	2007
4	Radial Velocity	1	516.220	10.50	119.47	2009

```
In [5]: df.tail()
Out[5]:
```

	method	number	orbital_period	mass	distance	year
--	--------	--------	----------------	------	----------	------


```

1030 Transit      1      3.941507  NaN    172.0  2006
1031 Transit      1      2.615864  NaN    148.0  2007
1032 Transit      1      3.191524  NaN    174.0  2007
1033 Transit      1      4.125083  NaN    293.0  2008
1034 Transit      1      4.187757  NaN    260.0  2008

In [6]: df.head(1)
Out[6]:
      method  number  orbital_period  mass  distance  year
0  Radial Velocity      1           269.3   7.1      77.4  2006

In [7]: df.tail(2)
Out[7]:
      method  number  orbital_period  mass  distance  year
1033  Transit      1      4.125083  NaN    293.0  2008
1034  Transit      1      4.187757  NaN    260.0  2008

```

4.6 使用 DataFrame 存取器

4.6.1 使用 DataFrame 存取器 loc[]

DataFrame 提供了存取器 loc[], 语法如下:

```
DataFrame.loc[n, m]
```

- ❑ n 可以是单个行标签、多行标签数组（或列表）、行标签切片、布尔数组。
- ❑ m 可以是单个列标签、多列标签数组（或列表）、列标签切片、布尔数组。

```

In [33]: data = {
...:     'apples': [3, 2, 0, 1],

```

```
...:     'oranges': [0, 1, 2, 3],
...:     'bananas': [1, 2, 1, 0]
...: }
...: df = pd.DataFrame(data,
...: index=['June', 'Robert', 'Lily', 'David'])
...: df
```

```
In [81]: df.loc['David', 'apples']
```

```
Out[81]: 1
```

```
In [82]: df.loc[['David', 'Robert'], 'apples']
```

```
Out[82]:
```

```
David    1
```

```
Robert   2
```

```
Name: apples, dtype: int64
```

```
In [84]: b = [True, False, True, False]
```

```
In [85]: df.loc[b, 'apples']
```

```
Out[85]:
```

```
June     3
```

```
Lily     0
```

```
Name: apples, dtype: int64
```

```
In [86]: df.loc[b, 'apples:']
```

```
Out[86]:
```

	apples	oranges	bananas
June	3	0	1
Lily	0	2	1

	0	1	2
	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0

$b = [\text{True}, \text{False}, \text{True}, \text{False}]$
 $\text{df.loc}[b, \text{'apples'}]$

	0	1	2
	apples	oranges	bananas
0 June	3	0	1
2 Lily	0	2	1

图 4-13 使用 DataFrame 存取器 loc[]

4.6.2 使用 DataFrame 存取器 iloc[]

iloc[]用法与 loc[]类似，区别只是 iloc 其中的参数都是位置，语法如下：

```
DataFrame.iloc[n, m]
```

- ❑ n 可以是单个行位置、多行位置数组（或列表）、行位置切片、布尔数组。
- ❑ m 可以是单个列位置、多列位置数组（或列表）、列位置切片、布尔数组。

```
In [33]: data = {
...:     'apples': [3, 2, 0, 1],
...:     'oranges': [0, 1, 2, 3],
...:     'bananas': [1, 2, 1, 0]
...: }
...: df = pd.DataFrame(data,
...:     index=['June', 'Robert', 'Lily', 'David'])
...: df
```

```
In [90]: df.iloc[3, 0]
```

```
Out[90]: 1
```

```
In [91]: df.iloc[[3, 1], 0]
```

```
Out[91]:
```

```
David    1
```

```
Robert   2
```

```
Name: apples, dtype: int64
```

```
In [92]: b = [True, False, True, False]
```

```
In [93]: df.iloc[b, 0]
```

```
Out[93]:
```

```
June     3
```

```
Lily     0
```

```
Name: apples, dtype: int64
```

```
In [94]: df.iloc[b, 0:]
```

```
Out[94]:
```

	apples	oranges	bananas
June	3	0	1
Lily	0	2	1

```
In [95]: df.iloc[:, 0:2]
```

```
Out[95]:
```

	apples	oranges
June	3	0
Robert	2	1
Lily	0	2
David	1	3

```
In [96]: df.iloc[1:3, 0:2]
```

```
Out[96]:
```

	apples	oranges
Robert	2	1
Lily	0	2

		0	1	2
		apples	oranges	bananas
0	June	3	0	1
1	Robert	2	1	2
2	Lily	0	2	1
3	David	1	3	0

		0	1
		apples	oranges
1	Robert	2	1
2	Lily	0	2

图 4-14 使用 DataFrame 存取器 iloc[]

4.6.3 使用 DataFrame 存取器 at[]和 iat[]

存取器 at[]和 iat[]可以访问 DataFrame 对象中的单个值。他们的语法如下：

```
DataFrame.at[n, m]
```

- ❑ n 是行标签。
- ❑ m 是列标签。

```
DataFrame.iat[idx_n, idx_m]
```

- ❑ idx_n 是行位置。
- ❑ idx_m 是列位置。

```
In [33]: data = {
...:     'apples': [3, 2, 0, 1],
...:     'oranges': [0, 1, 2, 3],
...:     'bananas': [1, 2, 1, 0]
...: }
...: df = pd.DataFrame(data,
```

```
index=['June','Robert','Lily','David'])
```

```
In [108]: df.at['Robert', 'apples']
```

```
Out[108]: 2
```

```
In [110]: df.iat[1, 0]
```

```
Out[110]: 2
```

4.7 DataFrame 行添加和删除

4.7.1 DataFrame 行添加

行添加使用 append 方法

```
In [22]: data = {
...:     'apples': [3, 2, 0, 1],
...:     'oranges': [0, 1, 2, 3],
...:     'bananas': [1, 2, 1, 0]
...: }
...: df = pd.DataFrame(data,
...: index=['June','Robert','Lily','David'])
...: df
```

```
Out[22]:
```

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0

```
In [23]: L = [[1,3,1],
```

```
...:     [2,4,1]]
```

```
...:
```

```
...: df2 = pd.DataFrame(L,
```

```
...: columns=['apples','oranges','bananas'],  
index=['Robert','Tom'])  
...:
```

```
In [24]: df2
```

```
Out[24]:
```

	apples	oranges	bananas
Robert	1	3	1
Tom	2	4	1

```
In [25]: df3 = df.append(df2)
```

```
In [26]: df3
```

```
Out[26]:
```

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0
Robert	1	3	1
Tom	2	4	1

4.7.2 DataFrame 行删除

行删除 `drop(labels=None)`

```
In [69]: df4 = df3.drop('Robert')
```

```
In [70]: df4
```

```
Out[70]:
```

	apples	oranges	bananas
June	3	0	1
Lily	0	2	1

David	1	3	0
Tom	2	4	1

4.8 DataFrame 列添加和删除

4.8.1 DataFrame 列添加

```
In [71]: data = {
...:     'apples': [3, 2, 0, 1],
...:     'oranges': [0, 1, 2, 3]
...: }
...: df = pd.DataFrame(data,
...: index=['June', 'Robert', 'Lily', 'David'])
...: df
Out[71]:
   apples  oranges
June      3        0
Robert    2        1
Lily      0        2
David     1        3

In [72]: df['banbanas'] = pd.Series([1, 2, 1, 0],
...: index=['June', 'Robert', 'Lily', 'David'])

In [73]: df
Out[73]:
   apples  oranges  banbanas
June      3        0         1
Robert    2        1         2
Lily      0        2         1
David     1        3         0
```


4.8.2 DataFrame 列删除

删除可以使用：

- ❑ `del` 语句
- ❑ `pop` 方法，`pop` 方法删除选择列并返回删除的列

```
In [74]: del df['banbanas']
```

```
In [75]: df
```

```
Out[75]:
```

	apples	oranges
June	3	0
Robert	2	1
Lily	0	2
David	1	3

```
In [76]: df.pop('apples')
```

```
Out[76]:
```

June	3
Robert	2
Lily	0
David	1

Name: apples, dtype: int64

```
In [77]: df
```

```
Out[77]:
```

	oranges
June	0
Robert	1
Lily	2
David	3

4.9 更改列标题

使用 rename 方法

```
In [78]: data = {
...:     'apples': [3, 2, 0, 1],
...:     'oranges': [0, 1, 2, 3],
...:     'bananas': [1, 2, 1, 0]
...: }
...: df = pd.DataFrame(data,
...: index=['June', 'Robert', 'Lily', 'David'])
...: df
```

```
Out[78]:
```

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0

```
In [79]: df = df.rename(columns={'apples': '苹果', 'oranges': '桔子',
...: 'bananas': '香蕉'})
```

```
In [80]: df
```

```
Out[80]:
```

	苹果	桔子	香蕉
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0

4.10 课后练习

1、编写一个 pandas 程序，将 DataFrame 的第一列转换为 Series。

参考答案:

```
import pandas as pd
d = {'col1': [1, 2, 3, 4, 7, 11], 'col2': [4, 5, 6, 9, 5, 0],
     'col3': [7, 5, 8, 12, 1, 11]}
df = pd.DataFrame(data=d)
print("原始数据: ")
print(df)
print("获得df第col1列的Series对象: ")
s1 = df['col1']
print(s1)
print(type(s1))
```

2、编写 pandas 程序, 指定字典创建 DataFrame 对象, 并具有指定行标签。

参考答案:

```
import pandas as pd

exam_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James',
                     'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],
             'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
             'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
             'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes',
                        'no', 'no', 'yes']}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

df = pd.DataFrame(exam_data , index=labels)
print(df)
```

3、在第 2 题基础上获得 DataFrame 对象的前 3 行。

参考答案:

```
print(df.iloc[:3])
或
print(df.head(3))
```

4、在第 2 题基础上从 DataFrame 对象中选中两列。

参考答案：

```
print(df[['name', 'score']])
```

5、在第 2 题基础上从 DataFrame 对象中选中 name 和 score 两列的 1、3、5、6 行数据。

参考答案：

```
print(df.iloc[[1, 3, 5, 6], [1, 3]])
```

6、在第 2 题基础上从 DataFrame 对象中选择分数在 15 到 20 之间（包括 15 和 20）的行。

参考答案：

```
print(df.query("score>=15 and score<=20"))  
或  
print(df[df['score'].between(15, 20)])
```

7、在第 2 题基础上从 DataFrame 对象中插入新列。

参考答案：

```
color =  
['Red', 'Blue', 'Orange', 'Red', 'White', 'White', 'Blue', 'Green', 'Green',  
 'Red']  
df['color'] = color  
print(df)
```

8、在第 2 题基础上从 DataFrame 对象中删除一列。

参考答案:

```
df.pop('attempts')
print(df)
```

9、编写一个 pandas 程序来重新给定 DataFrame 对象的列。

参考答案:

```
import pandas as pd
d = {'col1': [1, 2, 3], 'col2': [4, 5, 6], 'col3': [7, 8, 9]}
df = pd.DataFrame(data=d)
print("原始DataFrame对象: ")
print(df)
df.columns = ['Column1', 'Column2', 'Column3']
df = df.rename(columns={'col1': 'Column1', 'col2': 'Column2',
                        'col3': 'Column3'})
print("重命名DataFrame列名:")
print(df)
```

10、编写一个 pandas 程序，根据某些列中的值从给定的 DataFrame 对象中选择行。

参考答案:

```
import pandas as pd
d = {'col1': [1, 4, 3, 4, 5], 'col2': [4, 5, 6, 7, 8], 'col3': [7, 8, 9, 0, 1]}
df = pd.DataFrame(data=d)
print("原始DataFrame对象: ")
print(df)
print("column1值的行== 4 :")
print(df.loc[df['col1'] == 4])
```

11、在第 2 题基础上变更 DataFrame 对象列顺序。

参考答案:

```
df = df[['col3', 'col2', 'col1']]  
print(df)
```

12、在第 2 题基础上从 DataFrame 对象中选中指定行。

参考答案：

```
print(df.iloc[3])
```

13、在第 2 题基础上从 DataFrame 对象中选择除某列外的其他列。

参考答案：

```
df = df.loc[:, df.columns != 'col3']  
print(df)
```

第5章 索引

5.1 Index 对象

5.1.1 一级索引 Index 对象

1、从 Series 对象中获得 Index 对象：

```
In [1]: data = {'a' : 3, 'b' : 2, 'c' : 0, 'd' : 1}
In [2]: s = pd.Series(data)
In [3]: s
Out[3]:
a    3
b    2
c    0
d    1
dtype: int64

In [4]: s.index
Out[4]: Index(['a', 'b', 'c', 'd'], dtype='object')
```

2、从 DataFrame 对象中获得 Index 对象：

```
In [33]: data = {
...:     'apples': [3, 2, 0, 1],
...:     'oranges': [0, 1, 2, 3],
...:     'bananas': [1, 2, 1, 0]
...: }
...: df = pd.DataFrame(data,
...: index=['June', 'Robert', 'Lily', 'David'])

In [34]: df.index
```

```
Out[34]: Index(['June', 'Robert', 'Lily', 'David'], dtype='object')

In [35]: df.columns
Out[35]: Index(['apples', 'oranges', 'bananas'], dtype='object')
```

5.1.2 创建 Index 对象

1、在 Series 对象中使用 Index 对象：

```
In [1]: labels = pd.Index(['a', 'b', 'c', 'd', 'e'])

In [2]: s = pd.Series(np.arange(5), index=labels)

In [3]: s
Out[3]:
a    0
b    1
c    2
d    3
e    4
dtype: int32
```

2、在 DataFrame 对象中使用 Index 对象：

```
In [7]: L = [[3,0,1],
...:         [2,1,2],
...:         [0,2,1],
...:         [1,3,0]]
...: df = pd.DataFrame(L)
...: df
Out[7]:
   0  1  2
0  3  0  1
```



```

1  2  1  2
2  0  2  1
3  1  3  0

```

```
In [8]: row_labels = pd.Index(['June', 'Robert', 'Lily', 'David'])
```

```
In [9]: col_labels = pd.Index(['apples', 'oranges', 'bananas'])
```

```
In [10]: df = pd.DataFrame(L, index=row_labels, columns=col_labels)
```

```
In [11]: df
```

```
Out[11]:
```

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0

5.1.3 重建索引

重建索引使用 `reindex()` 方法。

1、在 Series 对象中重建索引:

```
In [1]: import pandas as pd
...: import numpy as np
```

```
In [2]: labels = pd.Index(['a', 'b', 'c', 'd', 'e'])
```

```
...: s = pd.Series(np.arange(5), index=labels)
```

```
...: s
```

```
Out[2]:
```

```

a    0
b    1
c    2
d    3
e    4

```

```
dtype: int32

In [3]: s1 = s.reindex(['a', 'b', 'c'])
...: s1
Out[3]:
a    0
b    1
c    2
dtype: int32

In [4]: s2 = s.reindex(['a', 'b', 'c', 'd', 'e', 'f'])
...: s2
Out[4]:
a    0.0
b    1.0
c    2.0
d    3.0
e    4.0
f     NaN
dtype: float64
```

2、在 DataFrame 对象中重建索引:

```
In [6]: L = [[3,0,1],
...:         [2,1,2],
...:         [0,2,1],
...:         [1,3,0]]
...: df = pd.DataFrame(L,
...:                    columns=['apples', 'oranges', 'bananas'],
...:                    index=['June', 'Robert', 'Lily', 'David'])

In [7]: df2 = df.reindex(['June', 'Lily', 'Robert'])
...: df2
Out[7]:
   apples  oranges  bananas
```

June	3	0	1
Lily	0	2	1
Robert	2	1	2

```
In [8]: df3 = df.reindex(columns=['apples', 'bananas', 'oranges'])
...: df3
```

```
Out[8]:
```

	apples	bananas	oranges
June	3	1	0
Robert	2	2	1
Lily	0	1	2
David	1	0	3

5.2 MultiIndex 对象

5.2.1 创建多级索引对象

一维 Series 对象表示二维数据:

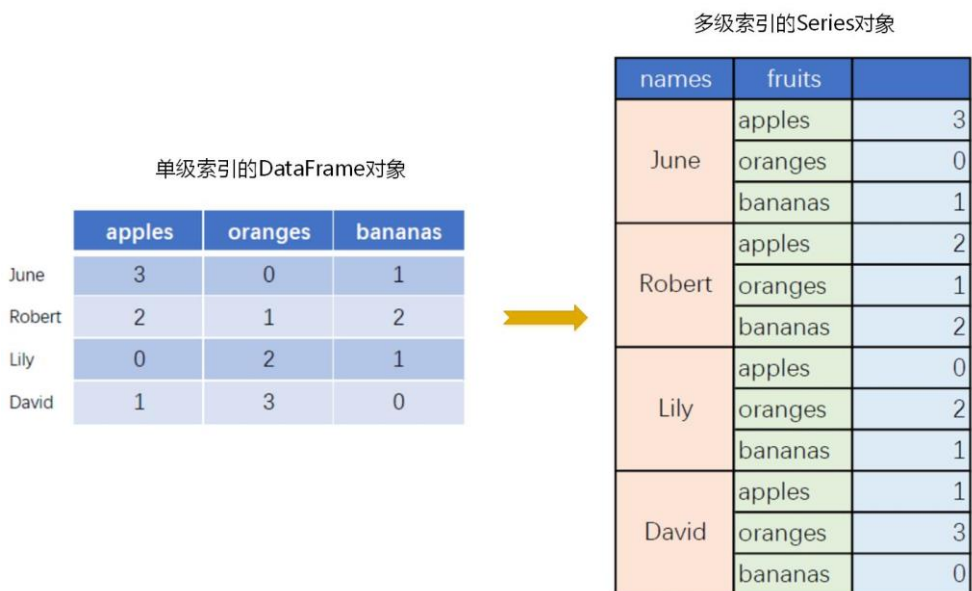


图 5-1 多级索引对象

```
In [78]: keys = [('June', 'apples'), ('June', 'oranges'),
...:            ('June', 'bananas'),
...:            ('Robert', 'apples'), ('Robert', 'oranges'),
...:            ('Robert', 'bananas'),
...:            ('Lily', 'apples'), ('Lily', 'oranges'),
...:            ('Lily', 'bananas'),
...:            ('David', 'apples'), ('David', 'oranges'),
...:            ('David', 'bananas')]

In [81]: index = pd.MultiIndex.from_tuples(keys,
...:                                       names=['names', 'fruits'])

In [82]: index
Out[82]:
MultiIndex(levels=[['David', 'June', 'Lily', 'Robert'],
...:               ['apples', 'bananas', 'oranges']],
...:       labels=[[1, 1, 1, 3, 3, 3, 2, 2, 2, 0, 0, 0], [0, 2,
...:               1, 0, 2, 1, 0, 2, 1]],
...:       names=['names', 'fruits'])

In [87]: data = [3,0,1,
...:              2,1,2,
...:              0,2,1,
...:              1,3,0]
...:

In [89]: s = pd.Series(data, index=index)

In [90]: s
Out[90]:
names  fruits
June   apples    3
       oranges    0
       bananas    1
Robert apples    2
       oranges    1
       bananas    2
Lily   apples    0
       oranges    2
```

```
      bananas    1
David  apples    1
      oranges    3
      bananas    0
dtype: int64
```

创建 MultiIndex 对象方法:

- ❑ pandas.MultiIndex.from_arrays, 从数组创建。
- ❑ pandas.MultiIndex.from_product, 从笛卡尔积创建。
- ❑ pandas.MultiIndex.from_tuples, 从元组创建。
- ❑ pandas.MultiIndex.from_frame, 另外 DataFrame 对象创建。

5.2.2 多级索引行列转换

1、多级索引转化普通索引

unstack() 方法可以快速将一个多级索引的 Series 转化为普通索引的 DataFrame:

```
In [125]: s.unstack()
Out[125]:
fruits  apples  bananas  oranges
names
David      1         0         3
June       3         1         0
Lily       0         1         2
Robert     2         2         1
```

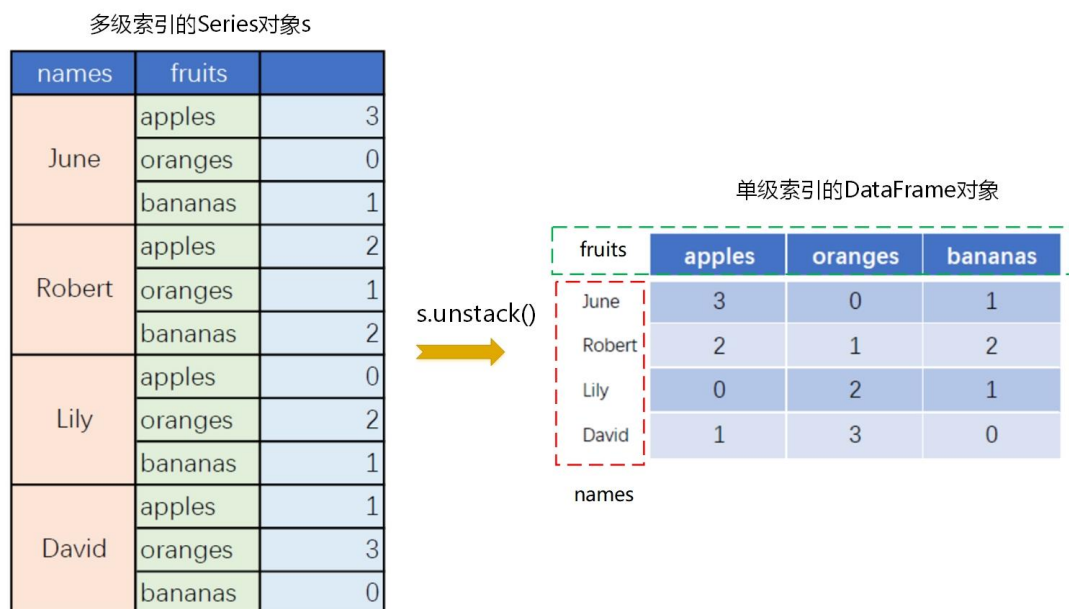


图 5-2 多级索引转化普通索引

2、stack() 方法可以将一个普通索引的 DataFrame 转化为多级索引的 Series:

```
In [33]: data = {
...:     'apples': [3, 2, 0, 1],
...:     'oranges': [0, 1, 2, 3],
...:     'bananas': [1, 2, 1, 0]
...: }
...: df = pd.DataFrame(data,
...: index=['June', 'Robert', 'Lily', 'David'])
```

```
In [34]: df.stack()
```

```
Out[34]:
```

```
June    apples    3
        oranges    0
        bananas    1
Robert  apples    2
        oranges    1
        bananas    2
Lily    apples    0
        oranges    2
```

```

        bananas    1
David  apples     1
        oranges    3
        bananas    0
dtype: int64

```

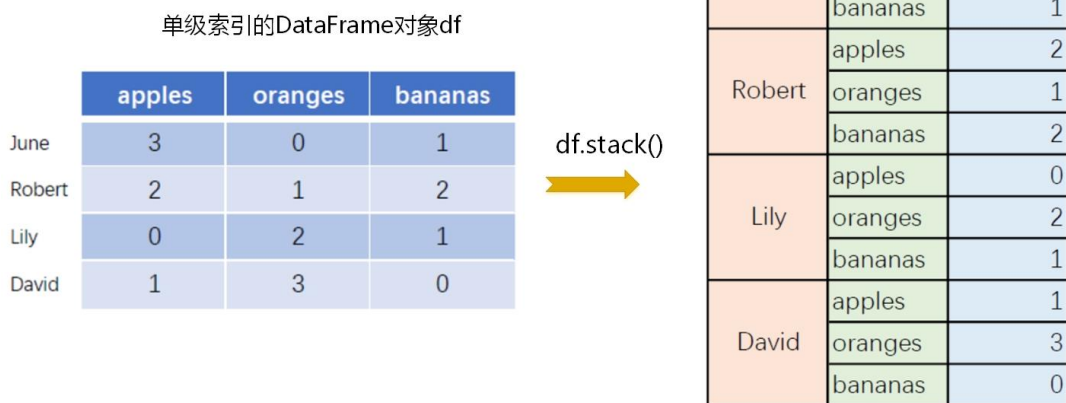


图 5-3 普通索引转化多级索引

5.2.3 多级索引数据存取

多级索引数据存取可以使用[]运算符，也可以使用 loc 等存取器。语法：

[第 1 级索引的标签（或切片），第 2 级索引的标签（或切片），...，第 n 级索引的标签（或切片）]

loc 存取器还可以使用标签列表。

```

In [15]: s[:, 'apples']
Out[15]:
names
June     3
Robert   2

```

```

Lily      0
David     1
dtype: int64

In [16]: s.loc[:, 'apples']
Out[16]:
names
June      3
Robert    2
Lily      0
David     1
dtype: int64

In [17]: s['June', 'apples']
Out[17]: 3

In [18]: s.loc['June', 'apples']
Out[18]: 3

In [19]: s['June', :]
Out[19]:
fruits
apples    3
oranges   0
bananas   1
dtype: int64

In [20]: s.loc['June', :]
Out[20]:
fruits
apples    3
oranges   0
bananas   1
dtype: int64

```


5.3 课后练习

1、编写一个 pandas 程序来改变 Series 对象索引。

参考答案:

```
import pandas as pd
s = pd.Series(data = [1,2,3,4,5], index = ['A', 'B', 'C', 'D', 'E'])
print("原始数据: ")
print(s)
s = s.reindex(index = ['B', 'A', 'C', 'D', 'E'])
print("改变Series对象索引:")
print(s)
```

2、创建多级索引对象，保存城市信息，包括所在国家（country）、面积（area）和人口（population）。

参考答案:

```
import pandas as pd
cities = [('Vienna', 'country'),
          ('Vienna', 'area'),
          ('Vienna', 'population'),

          ('Hamburg', 'country'),
          ('Hamburg', 'area'),
          ('Hamburg', 'population'),

          ('Berlin', 'country'),
          ('Berlin', 'area'),
          ('Berlin', 'population'),

          ('Zürich', 'country'),
          ('Zürich', 'area'),
          ('Zürich', 'population')]
index = pd.MultiIndex.from_tuples(cities)
或者
```

```
index =  
pd.MultiIndex.from_product([['Vienna', 'Hamburg', 'Berlin', 'Zürich'],  
                             ['country', 'area', 'population']])
```

3、在第 2 题基础上，创建具有多级索引的 Series 对象。

参考答案：

```
data = ["Austria", 414.60,    1805681,  
        "Germany", 755.00,    1760433,  
        "Germany", 891.85,    3562166,  
        "Switzerland", 87.88, 378884]  
city_series = pd.Series(data, index=index)
```

4、在第 3 题基础上访问 Series 对象，获取城市数据、城市面积、城市人口。

参考答案：

```
data = ["Austria", 414.60,    1805681,  
        "Germany", 755.00,    1760433,  
        "Germany", 891.85,    3562166,  
        "Switzerland", 87.88, 378884]  
city_series = pd.Series(data, index=index)  
  
print('城市数据', city_series['Vienna'])  
print("城市面积", city_series["Vienna", "area"])  
print("城市人口", city_series["Vienna", "population"])
```

第6章 数据读写操作

数据包括格式：文本格式和二进制格式。

- ❑ 文本格式：CSV、HTML、JSON 等。
- ❑ 二进制格式：Excel、Pickle、HDF5 等。

6.1 读写 Excel 文件

6.1.1 读取 Excel 文件数据

读取 Excel 文件数据函数是 `pandas.read_excel()`，该函数返回值是 **DataFrame 对象**，该函数语法格式：

```
pandas.read_excel(io, sheet_name=0, header=0, index_col=None,
skiprows=None, skipfooter=0)
```

主要的参数如下：

- ❑ `io`：是输入 Excel 文件。可以是字符串、文件对象、`ExcelFile` 对象，可以是本地文件，也可以是网络 URL。
- ❑ `sheet_name`：是 Excel 文件工作表名，可以是字符串、整数（基于 0 的工作表位置索引）、列表（选择多个工作表）。
- ❑ `header`：用作 DataFrame 对象列标签的行号，默认是 0（第一行）；如果设置 `None`，则没有指定列标签。
- ❑ `index_col`：用作 DataFrame 对象的行标签的列号，默认是 `None`。
- ❑ `skiprows`：忽略文件头部行数，默认是 `None`。
- ❑ `skipfooter`：忽略文件尾部行数，默认是 0。

6.1.2 示例：从 Excel 文件读取全国总人口数据

数据来源国家统计局 (<http://www.stats.gov.cn/>)

读取【全国总人口数据.xls】文件

1、基本参数

```
In [13]: file_path = 'data\\'

In [14]: df10 = pd.read_excel(file_path+'全国总人口10年数据.xls')

In [15]: df10
Out[15]:
```

	1	...	Unnamed: 9	Unnamed: 10	数据库: 年度数据	Unnamed:
0	NaN	...	NaN	NaN	时间: 最近10年	
1	2010年		2009年		指标	2018年 ...
2	139538	...	134091	133450	年末总人口(万人)	
3	71351	...	68748	68647	男性人口(万人)	
4	68187	...	65343	64803	女性人口(万人)	
5	83137	...	66978	64512	城镇人口(万人)	
6	56401	...	67113	68938	乡村人口(万人)	
7	注: 1981年及以前人口数据为户籍统计数; 1982、1990、2000、2010年数据为当年...					
8	NaN	...	NaN	NaN	数据来源: 国家统计局	

[9 rows x 11 columns]

2、忽略文件头部和尾部的行数

```
In [21]: df10 = pd.read_excel(file_path+'全国总人口数据.xls',skiprows=2,skipfooter=2)

In [22]: df10
Out[22]:
```

	指标	2018年	2017年	2016年	2015年	2014年	2013年
	2012年	2011年	2010年	2009年			
0	年末总人口(万人)	139538	139008	138271	137462	136782	136072
	135404	134735	134091	133450			
1	男性人口(万人)	71351	71137	70815	70414	70079	69728
	69395	69068	68748	68647			

2	女性人口(万人)	66009	65667	65343	67871	64803	67456	67048	66703	66344
3	城镇人口(万人)	71182	69079	66978	83137	81347	79298	77116	74916	73111
4	乡村人口(万人)	64222	65656	67113	56401	57661	58973	60346	61866	62961

3、指定行标签

```
In [69]: df10 = pd.read_excel(file_path+'全国总人口数据.xls', skiprows=2, skipfooter=2, index_col=0)
```

```
In [70]: df10
```

```
Out[70]:
```

	2018年	2017年	2016年	2015年	2014年	2013年	2012年
2011年	2010年	2009年					
指标							
年末总人口(万人)	139538	139008	138271	137462	136782	136072	
	135404	134735	134091	133450			
男性人口(万人)	71351	71137	70815	70414	70079	69728	69395
	69068	68748	68647				
女性人口(万人)	68187	67871	67456	67048	66703	66344	66009
	65667	65343	64803				
城镇人口(万人)	83137	81347	79298	77116	74916	73111	71182
	69079	66978	64512				
乡村人口(万人)	56401	57661	58973	60346	61866	62961	64222
	65656	67113	68938				

4、指定工作表

```
In [71]: df20 = pd.read_excel(file_path+'全国总人口数据.xls',
    sheet_name='20年数据', skiprows=2, skipfooter=
    ...: 2, index_col=0)
```

```
In [72]: df20
```

```
Out[72]:
```

	年末总人口(万人)	男性人口(万人)	女性人口(万人)	城镇人口(万人)	乡村人口(万人)
时间					
2018年	139538	71351	68187	83137	56401

2017年	139008	71137	67871	81347	57661
2016年	138271	70815	67456	79298	58973
2015年	137462	70414	67048	77116	60346
2014年	136782	70079	66703	74916	61866
2013年	136072	69728	66344	73111	62961
2012年	135404	69395	66009	71182	64222
2011年	134735	69068	65667	69079	65656
2010年	134091	68748	65343	66978	67113
2009年	133450	68647	64803	64512	68938
2008年	132802	68357	64445	62403	70399
2007年	132129	68048	64081	60633	71496
2006年	131448	67728	63720	58288	73160
2005年	130756	67375	63381	56212	74544
2004年	129988	66976	63012	54283	75705
2003年	129227	66556	62671	52376	76851
2002年	128453	66115	62338	50212	78241
2001年	127627	65672	61955	48064	79563
2000年	126743	65437	61306	45906	80837
1999年	125786	64692	61094	43748	82038

sheet_name='20 年数据' 可以替换为 sheet_name=1

6.1.3 写入数据到 Excel 文件

写入数据到 Excel 文件数据是通过 Series 和 DataFrame 对象的 to_excel() 方法实现，该方法语法格式：

```
to_excel(excel_writer, sheet_name='Sheet1', header=True, index=True)
```

主要的参数如下：

- ❑ excel_writer: 是写入 Excel 文件。可以是字符串、ExcelWriter 对象。
- ❑ sheet_name: 是 Excel 文件工作表名，字符串类型，默认是 'Sheet1'。
- ❑ header: 是写入列名。可以是布尔类型 (False 是不写入列名，True 是将 Series 和 DataFrame 对象列标签作为列名)；也可以字符串列表

Python 数据分析与科学计算基础篇 2: pandas 图解

(自定义列名)。

❑ index: 是否写入行名。该参数只能是布尔类型，默认是 True。

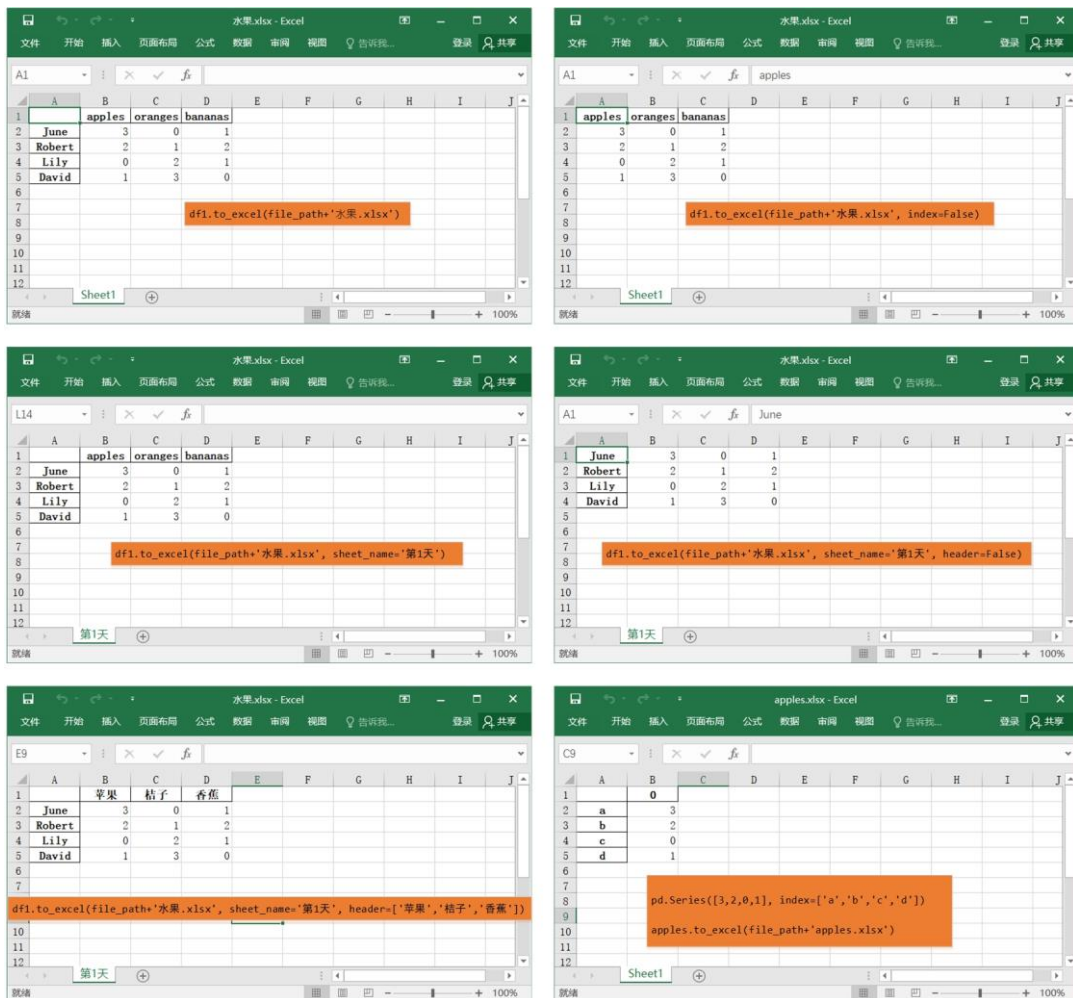


图 6-1 写入 Excel 文件

6.1.4 示例：写入水果数据到 Excel 文件

```
In [13]: file_path = 'data\\'
```

```
In [38]: data1 = {  
    ...:     'apples': [3, 2, 0, 1],  
    ...:     'oranges': [0, 1, 2, 3],
```

```

...:     'bananas': [1, 2, 1, 0]
...: }
...: df1 = pd.DataFrame(data,
...: index=['June', 'Robert', 'Lily', 'David'])
...: df1
Out[38]:
      apples  oranges  bananas
June        3         0         1
Robert       2         1         2
Lily         0         2         1
David        1         3         0

In [39]: df1.to_excel(file_path+'水果.xlsx')
In [40]: df1.to_excel(file_path+'水果.xlsx', index=False)
In [41]: df1.to_excel(file_path+'水果.xlsx', sheet_name='第1天')
In [42]: df1.to_excel(file_path+'水果.xlsx', sheet_name='第1天',
...: header=False)
In [43]: df1.to_excel(file_path+'水果.xlsx', sheet_name='第1天',
...: header=['苹果', '桔子', '香蕉'])
In [44]: apples = pd.Series([3,2,0,1], index=['a', 'b', 'c', 'd'])
In [45]: apples.to_excel(file_path+'apples.xlsx')

In [46]: with pd.ExcelWriter(file_path+'水果.xlsx') as writer:
...:     df1.to_excel(writer, sheet_name='第1天')
...:     apples.to_excel(writer, sheet_name='第2天')

```

6.2 读写 CSV 文件

6.2.1 读取 CSV 文件数据

读取 CSV 文件数据函数是 `pandas.read_csv()`，该函数返回值是 `DataFrame` 对象，该函数语法格式：

```

pandas.read_csv(filepath_or_buffer, sep=',', delimiter=None,
header='infer', index_col=None, skiprows=None, skipfooter=0)

```


主要的参数如下：

- ❑ `filepath_or_buffer`: 是输入 CSV 文件。可以是字符串、文件对象，可以是本地文件，可以是网络 URL。
- ❑ `sep` 或 `delimiter`: 用于分隔每行字段的字符或正则表达式。
- ❑ `header`: 用作 DataFrame 对象列标签的行号，默认是 'infer'（自动推断）。
- ❑ `index_col`: 用作 DataFrame 对象的行标签的列号，默认是 None。
- ❑ `skiprows`: 忽略文件头部行数，默认是 None。
- ❑ `skipfooter`: 忽略文件尾部行数，默认是 0。
- ❑ `engine`: 解析引擎，取值是有 `c` 和 `python`，默认是 `c`。`c` 不支持 `skipfooter` 参数。

6.2.2 示例：从 CSV 文件读取全国总人口数据

数据来源国家统计局 (<http://www.stats.gov.cn/>)

1、读取【全国总人口 10 年数据.csv】文件

```
df10 = pd.read_csv(file_path+'全国总人口10年数据.csv',skiprows=2,skipfooter=2,engine='python')
df10 = pd.read_csv(file_path+'全国总人口10年数据.csv',skiprows=2,skipfooter=2,index_col=0, engine='python')
df10 = pd.read_csv(file_path+'全国总人口10年数据.csv',header=2,skipfooter=2,index_col=0, engine='python')
df10
```

2、使用 `sep` 参数

```
In [4]: df1 = pd.read_csv(file_path+'pandas_tutorial_read.csv',
    sep=';',header=None)
In [5]: df1.head()
Out[5]:
    2018-01-01 00:01:01 read country_7 2458151261      SEO North
    America
0  2018-01-01 00:03:20 read country_7 2458151262      SEO South
    America
```

1	2018-01-01 00:04:01	read	country_7	2458151263	AdWords	
	Africa					
2	2018-01-01 00:04:02	read	country_7	2458151264	AdWords	
	Europe					
3	2018-01-01 00:05:03	read	country_8	2458151265	Reddit	North
	America					
4	2018-01-01 00:05:42	read	country_6	2458151266	Reddit	North
	America					

6.2.3 写入数据到 CSV 文件

写入数据到 CSV 文件数据是通过 Series 和 DataFrame 对象的 `to_csv()` 方法实现，该方法语法格式：

```
to_csv(path_or_buf=None, sep=',', header=True, index=True,
encoding=None)
```

主要的参数如下：

- ❑ `path_or_buf`：是写入 CSV 文件。可以是字符串、文件对象。
- ❑ `sep`：用于分隔每行字段的字符。
- ❑ `header`：是写入列名。可以是布尔类型（False 是不写入列名，True 是将 Series 和 DataFrame 对象列标签作为列名）；也可以字符串列表（自定义列名）。
- ❑ `index`：是否写入行名。该参数只能是布尔类型，默认是 True。
- ❑ `encoding`：设置字符集，Python2 默认 `ascii`，Python3 默认 `utf-8`。

6.2.4 示例：写入水果数据到 CSV 文件

```
In [13]: file_path = 'data\\'

In [38]: data1 = {
...:     'apples': [3, 2, 0, 1],
...:     'oranges': [0, 1, 2, 3],
...:     'bananas': [1, 2, 1, 0]
...: }
...: df1 = pd.DataFrame(data,
...: index=['June', 'Robert', 'Lily', 'David'])
...: df1
```

Out[38]:

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0

```
In [39]: df1.to_csv(file_path+'水果.csv', index=False)
```

```
In [40]: df1.to_csv(file_path+'水果.csv', header=False)
```

```
In [41]: df1.to_csv(file_path+'水果.csv', header=['苹果','桔子','香蕉',
            ''])
```

```
In [42]: df1.to_csv(file_path+'水果.csv', header=['苹果','桔子','香蕉',
            ''],encoding='gbk')
```

```
In [43]: apples = pd.Series([3,2,0,1], index=['a','b','c','d'])
```

```
In [44]: apples.to_csv(file_path+'apples.csv')
```

关于 CSV 字符集问题:

6.3 课后练习

1、读取 CSV 数据（煤矿生成历史数据 2013.csv）到 DataFrame 对象，并将前 20 行跳过。

参考答案:

```
import pandas as pd

df = pd.read_csv(file_dir+'煤矿生成历史数据2013.csv', skiprows = 20,
engine='python')

df
```

2、读取 CSV 数据（煤矿生成历史数据 2013.csv）到 DataFrame 对象，并显示最后十行。

参考答案:

```
import pandas as pd
df = pd.read_csv(file_dir+'煤矿生成历史数据2013.csv',engine='python')
df.tail(10)
```

3、读取 CSV 数据（煤矿生成历史数据 2013.csv）到 DataFrame 对象，并查找特定的“矿山 ID”。

参考答案：

```
import pandas as pd
df = pd.read_csv(file_dir+'煤矿生成历史数据2013.csv', engine='python')
df[df["矿山ID"]==102901]
```

4、读取 CSV 数据（煤矿生成历史数据 2013.csv）到 DataFrame 对象，并示并查找“工时” > 20000 的详细信息。

参考答案：

```
import pandas as pd
df = pd.read_csv(file_dir+'煤矿生成历史数据2013.csv',engine='python')
df[df["工时"] > 20000]
```

5、读取 Excel 文件数据（雇员.xlsx）到 DataFrame 对象，并转换数据以使用 hire_date 作为索引。

参考答案：

```
import pandas as pd
df = pd.read_excel(file_dir+'雇员.xlsx', index_col=0)
result = df.set_index(['hire_date'])
print(result)
```

6、读取 Excel 文件数据 (雇员.xlsx) 到 DataFrame 对象, 并根据两个特定月份和年份之间 hire_date 的员工列表。

参考答案:

```
import pandas as pd
df = pd.read_excel(file_dir+'雇员.xlsx', index_col=0)
result = df.query("hire_date >='2005-1' and hire_date < '2006-1'")
或
result = df[(df['hire_date'] >='2005-1') & (df['hire_date'] < '2006-1')]
print(result)
```

7、读取 Excel 文件数据 (雇员.xlsx) 到 DataFrame 对象, 并找 hire_date > 2007-01-01 雇员列表。

参考答案:

```
import pandas as pd
df = pd.read_excel(file_dir+'雇员.xlsx', index_col=0)
df[df['hire_date'] >='20070101']
或
df[df['hire_date'] >='2007-1-1']
或
df.query("hire_date >= '2007-1-1'")
```

第7章 数据操作

7.1 算术运算

对象 (Series 和 DataFrame) 中的元素可以进行算术 (加、减、乘、除等) 计算, 当索引相同时对象对应元素进行计算; 如果索引不同, 则返回结果的索引是两个索引的并集。

7.1.1 使用算术运算符

算术运算符包括一元取反 (-), 二元运算符包括: +、-、*、/、%、**和 //。

1、两个 Series 对象算术运算:

```
In [20]: s1 = pd.Series([8, 7, 3], index=list('abc'))

In [21]: s1
Out[21]:
a    8
b    7
c    3
dtype: int64

In [22]: s2 = pd.Series([3, 5, 8], index=list('abc'))

In [23]: s2
Out[23]:
a    3
b    5
c    8
dtype: int64

In [24]: s1+s2
Out[24]:
```

```
a    11
b    12
c    11
dtype: int64

In [25]: s3 = pd.Series([4, 2, 7], index=list('bcd'))

In [26]: s=s1+s3

In [27]: s
Out[27]:
a      NaN
b    11.0
c     5.0
d      NaN
dtype: float64

In [28]: s.fillna(0)
Out[28]:
a     0.0
b    11.0
c     5.0
d     0.0
dtype: float64
```

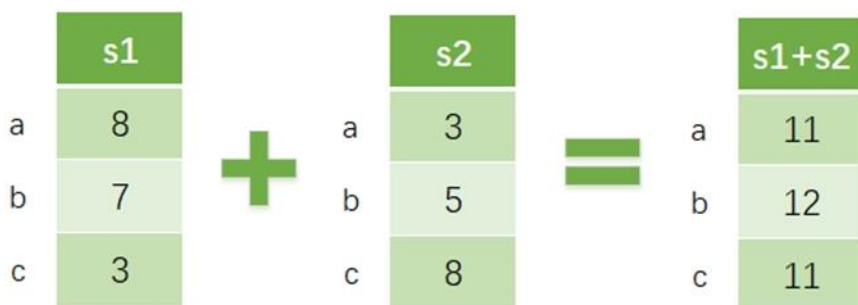


图 7-1 两个 Series 对象算术运算 1

	s1		s3		s1+s3
a	8	+	b	4	NaN
b	7		c	2	11.0
c	3		d	7	5.0
		=			NaN

图 7-2 两个 Series 对象算术运算 2

2、两个 DataFrame 对象算术运算：

```
In [30]: data1 = {'apples': [3, 2, 0, 1],
...:             'oranges': [0, 1, 2, 3] }
```

```
In [31]: df1 = pd.DataFrame(data1,
index=['June', 'Robert', 'Lily', 'David'])
```

```
In [32]: df1
```

```
Out[32]:
```

	apples	oranges
June	3	0
Robert	2	1
Lily	0	2
David	1	3

```
In [33]: data2 = {'apples': [2, 2, 1],
...:             'oranges': [1, 1, 3],
...:             'bananas': [1, 2, 1] }
```

```
In [34]: df2 = pd.DataFrame(data2, index=['June', 'Robert', 'Lily'])
```

```
In [35]: df2
```

```
Out[35]:
```

	apples	oranges	bananas
--	--------	---------	---------


```
June      2      1      1
Robert    2      1      2
Lily      1      3      1
```

```
In [36]: df = df1+df2
```

```
In [37]: df
```

```
Out[37]:
```

```
      apples  bananas  oranges
David     NaN     NaN     NaN
June      5.0     NaN     1.0
Lily      1.0     NaN     5.0
Robert    4.0     NaN     2.0
```

```
In [38]: df.fillna(0)
```

```
Out[38]:
```

```
      apples  bananas  oranges
David     0.0     0.0     0.0
June      5.0     0.0     1.0
Lily      1.0     0.0     5.0
Robert    4.0     0.0     2.0
```

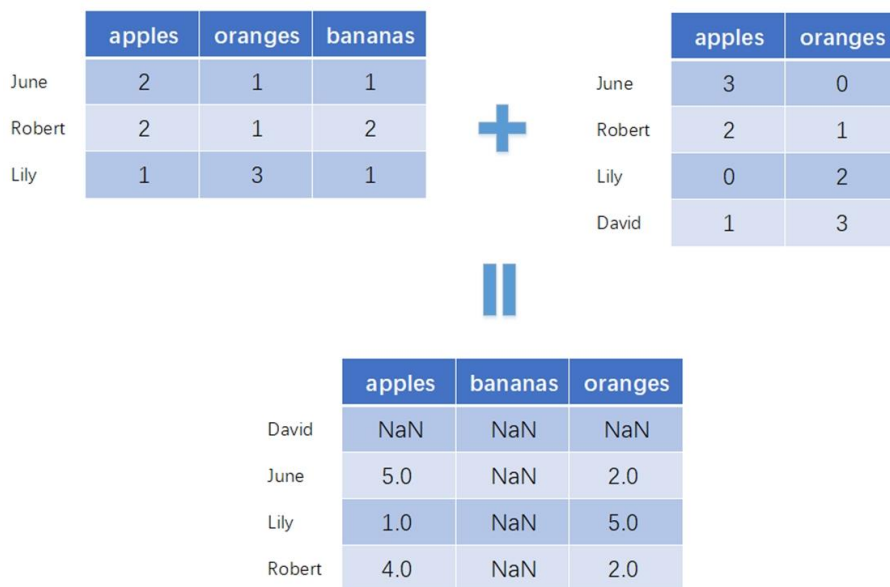


图 7-3 两个 DataFrame 对象算术运算

3、Series 对象和 DataFrame 对象算术运算：

```
In [5]: data1 = { 'apples': [3, 2, 0, 1],
...:             'oranges': [0, 1, 2, 3] }
```

```
In [6]: df1 = pd.DataFrame(data1,
index=['June', 'Robert', 'Lily', 'David'])
```

```
In [7]: df1
Out[7]:
```

	apples	oranges
June	3	0
Robert	2	1
Lily	0	2
David	1	3

```
In [8]: s1 = pd.Series([1, 2, 3], index=['apples', 'oranges',
'bananas'])
```

```
In [9]: s1
Out[9]:
```

apples	1
oranges	2
bananas	3

```
dtype: int64
```

```
In [10]: df1+s1
Out[10]:
```

	apples	bananas	oranges
June	4	NaN	2
Robert	3	NaN	3
Lily	1	NaN	4
David	2	NaN	5

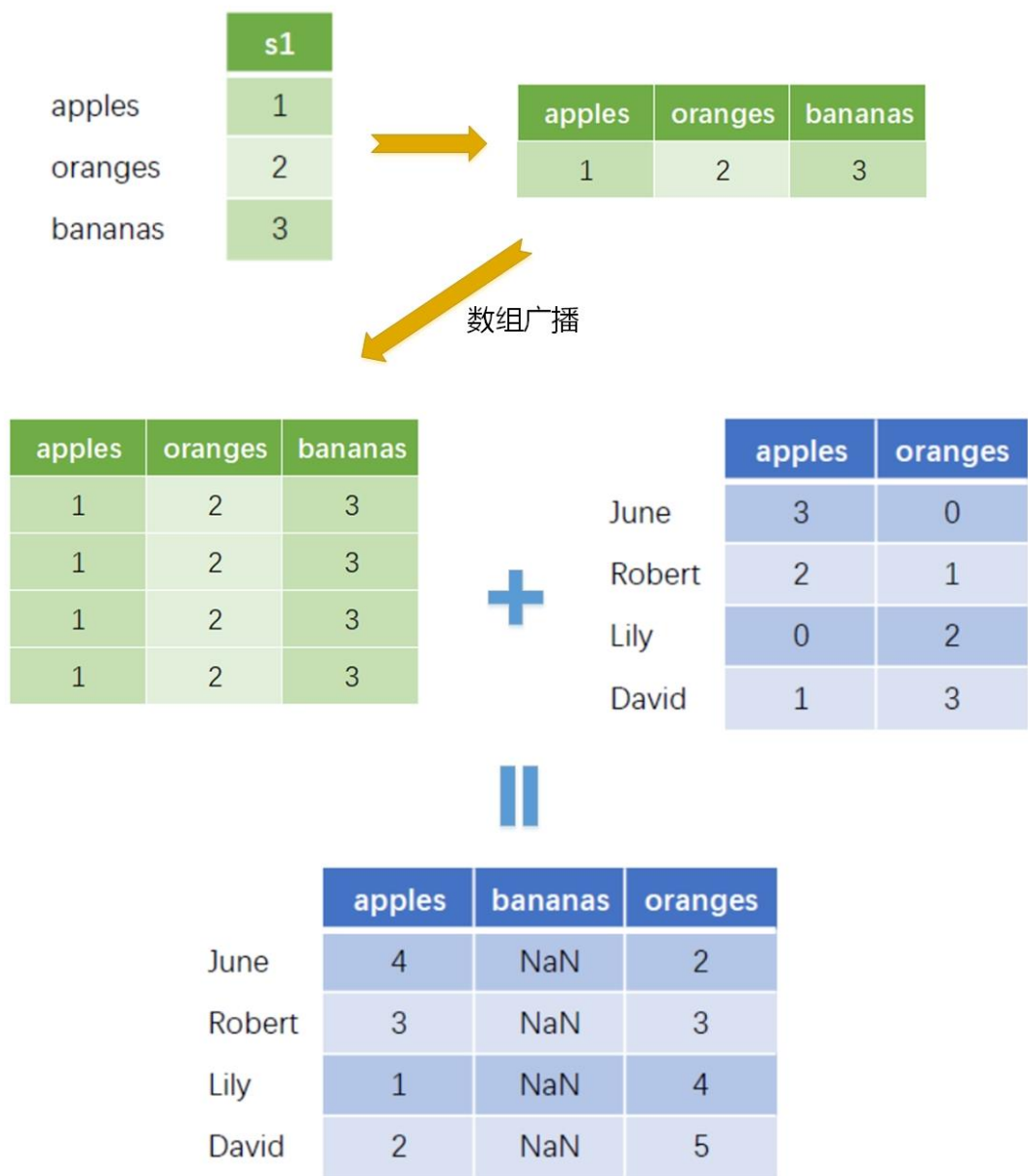


图 7-4 Series 对象和 DataFrame 对象算术运算

7.1.2 使用算术运方法

表 7-1 算术运算方法

方法	说明
add	加法 (+)
sub	减法 (-)
div	除法 (/)
floordiv	整除 (//)
mul	乘法 (*)
pow	幂次方 (**)

```

In [4]: s1+s2
Out[4]:
a    11
b    12
c    11
dtype: int64

In [5]: s1.add(s2)
Out[5]:
a    11
b    12
c    11
dtype: int64

In [6]: s1+s3
Out[6]:
a    NaN
b    11.0
c     5.0
d    NaN
dtype: float64

In [7]: s1.add(s3)
Out[7]:

```

```
a    NaN
b    11.0
c     5.0
d    NaN
dtype: float64
```

```
In [8]: s1.add(s3).fillna(0)
```

```
Out[8]:
```

```
a     0.0
b    11.0
c     5.0
d     0.0
dtype: float64
```

```
In [9]: df1+sf1
```

```
Out[9]:
```

	apples	bananas	oranges
June	4	NaN	2
Robert	3	NaN	3
Lily	1	NaN	4
David	2	NaN	5

```
In [10]: df1.add(sf1)
```

```
Out[10]:
```

	apples	bananas	oranges
June	4	NaN	2
Robert	3	NaN	3
Lily	1	NaN	4
David	2	NaN	5

```
In [11]: df1*df2
```

```
Out[11]:
```

	apples	bananas	oranges
David	NaN	NaN	NaN
June	6.0	NaN	0.0
Lily	0.0	NaN	6.0
Robert	4.0	NaN	1.0

```
In [12]: df1.mul(df2)
Out[12]:
```

	apples	bananas	oranges
David	NaN	NaN	NaN
June	6.0	NaN	0.0
Lily	0.0	NaN	6.0
Robert	4.0	NaN	1.0

```
In [13]: df1//df2
Out[13]:
```

	apples	bananas	oranges
David	NaN	NaN	NaN
June	1.0	NaN	0.0
Lily	0.0	NaN	0.0
Robert	1.0	NaN	1.0

```
In [14]: df1.floordiv(df2)
Out[14]:
```

	apples	bananas	oranges
David	NaN	NaN	NaN
June	1.0	NaN	0.0
Lily	0.0	NaN	0.0
Robert	1.0	NaN	1.0

7.2 描述性统计方法

Pandas 对象有一组常用的数学/统计方法：

- ❑ Series 中的各个元素汇总统计
- ❑ DataFrame 可以沿指定轴进行汇总统计

这些方法主要有 3 个参数：

1. axis 是沿指定轴的索引，默认为 0。DataFrame 对象 0 为行，1 为列；Series 对象只能为 0。
2. skipna 是排除缺失值，默认为 True。注意 count() 方法没有该参

数。

3. level 如果是多级索引时, 指定索引级别。

表 7-2 常用的统计方法

方法	说明
count()	计算非 NA 值的个数, 注意没有 skipna 参数
sum()	求和
mean()	平均值
std()	标准差
min()、max()	计算最小值、最大值
prod()	所有值的乘积
cumsum()	计算累加值
cumprod()	计算累乘积值

```
In [7]: data = {'apples': [3, 2, 0, 1],
...:           'oranges': [0, 1, 2, 3],
...:           'bananas': [1, 2, 1, np.nan]}
```

```
In [8]: df = pd.DataFrame(data,
index=['June', 'Robert', 'Lily', 'David'])
```

```
In [9]: df
```

```
Out[9]:
   apples  oranges  bananas
June      3        0      1.0
Robert    2        1      2.0
Lily      0        2      1.0
David     1        3      NaN
```

```
In [10]: df.count()
```

```
Out[10]:
apples    4
oranges    4
bananas    3
dtype: int64
```

```
In [11]: df.count(1)
```

```
Out[11]:
```

```
June      3
```

```
Robert    3
```

```
Lily      3
```

```
David     2
```

```
dtype: int64
```

```
In [12]: df.sum()
```

```
Out[12]:
```

```
apples     6.0
```

```
oranges    6.0
```

```
bananas    4.0
```

```
dtype: float64
```

```
In [13]: df.sum(1)
```

```
Out[13]:
```

```
June      4.0
```

```
Robert    5.0
```

```
Lily      3.0
```

```
David     4.0
```

```
dtype: float64
```

```
In [14]: df.sum(skipna=False)
```

```
Out[14]:
```

```
apples     6.0
```

```
oranges    6.0
```

```
bananas    NaN
```

```
dtype: float64
```

7.3 函数应用

要将自定义函数或其他库中的函数应用于 Pandas 对象,有 3 个主要方法:

- ❑ `apply(func)` 方法可以应用于 Series 也可以应用于 DataFrame。
- ❑ `applymap(func)` 方法仅适用于 DataFrame。
- ❑ `map(func)` 方法仅适用于 Series。

7.3.1 函数应用 `apply` 方法

`apply(func)` 方法可以应用于 Series 也可以应用于 DataFrame。func 函数参数，如果是标量（单个元素），可以应用于 Series 和 DataFrame 每一个元素；如果参数是向量（列表、数组、其他 Series 对象）可以应用于 DataFrame 的行或列。

1、应用于 Series 和 DataFrame 对象每一个元素

```
In [3]: def func1(x): # x单个值（标量）
...:     return x*100
...:

In [4]: s = pd.Series([8, 7, 3], index=list('abc'))
...: s
Out[4]:
a      8
b      7
c      3
dtype: int64

In [5]: s.apply(func1)
Out[5]:
a     800
b     700
c     300
dtype: int64

In [6]: data = {'apples': [2, 2, 1],
```

```

...:         'oranges': [1, 1, 3],
...:         'bananas': [1, 2, 1] }
...: df = pd.DataFrame(data, index=['June','Robert','Lily'])
...: df
Out[6]:
      apples  oranges  bananas
June        2        1        1
Robert       2        1        2
Lily         1        3        1

In [7]: df.apply(func1)
Out[7]:
      apples  oranges  bananas
June      200      100      100
Robert    200      100      200
Lily      100      300      100

```

提示 自定义函数也可以使用Lambda表达式。例如：func1函数可以用
lambda x:x*100表达式替代。

```

In [57]: s.apply(lambda x:x*100)
Out[57]:
a    800
b    700
c    300
dtype: int64

In [58]: df.apply(lambda x:x*100)
Out[58]:
      apples  oranges  bananas
June      200      100      100
Robert    200      100      200
Lily      100      300      100

```

2、应用于 DataFrame 对象的行或列

```
In [10]: s.apply(lambda x:x.max() - x.min())
-----
AttributeError: 'int' object has no attribute 'max'

In [11]: df.apply(lambda x:x.max() - x.min())
Out[11]:
apples      1
oranges     2
bananas     1
dtype: int64

In [12]: df.apply(lambda x:x.max() - x.min(), axis=0)
Out[12]:
apples      1
oranges     2
bananas     1
dtype: int64

In [13]: df.apply(lambda x:x.max() - x.min(), axis=1)
Out[13]:
June        1
Robert      1
Lily        2
dtype: int64
```

提示 `lambda x:x.max() - x.min()`表达式用来计算向量, 如`max`和`min`方法。

7.3.2 函数应用 `applymap` 方法

`applymap(func)` 方法仅适用于 DataFrame 对象每一个元素。

```
In [67]: df.applymap(lambda x:x*100)
```

```
Out[67]:
```

	apples	oranges	bananas
June	200	100	100
Robert	200	100	200
Lily	100	300	100

7.3.3 函数应用 map 方法

map(func) 方法仅适用于 Series 对象每一个元素。

```
In [70]: s.map(lambda x:x*100)
```

```
Out[70]:
```

```
a    800
b    700
c    300
dtype: int64
```

7.4 排序

Pandas 中对象 (Series 和 DataFrame) 可以通过标签或元素值进行排序。

7.4.1 通过标签排序

通过标签排序方法是 `sort_index()`，该方法主要的参数如下：

- ❑ `axis`: 指定排序的轴，默认为 0。DataFrame 对象 0 为行，1 为列；Series 对象只能为 0。
- ❑ `ascending`: 排序的顺序，默认为 True。True 为升序，False 为降序。

```
In [8]: s = pd.Series([8, 3, 7], index=list('bac'))
```

```
In [9]: s
```

```
Out[9]:
```

```
b    8
```

```
a    3
```

```
c    7
```

```
dtype: int64
```

```
In [10]: s.sort_index()
```

```
Out[10]:
```

```
a    3
```

```
b    8
```

```
c    7
```

```
dtype: int64
```

```
In [11]: s.sort_index(ascending=False)
```

```
Out[11]:
```

```
c    7
```

```
b    8
```

```
a    3
```

```
dtype: int64
```

```
In [12]: data = {'apples': [3, 2, 0, 1],
```

```
...:          'oranges': [0, 1, 2, 3],
```

```
...:          'bananas': [1, 2, 1, 0]}
```

```
In [13]: df = pd.DataFrame(data,  
index=['June', 'Robert', 'Lily', 'David'])
```

```
In [14]: df
```

```
Out[14]:
```

	apples	oranges	bananas
June	3	0	1
Robert	2	1	2
Lily	0	2	1
David	1	3	0

```
In [15]: df.sort_index()
```

```
Out[15]:
```

	apples	oranges	bananas
David	1	3	0
June	3	0	1
Lily	0	2	1
Robert	2	1	2

```
In [16]: df.sort_index(axis=1)
```

```
Out[16]:
```

	apples	bananas	oranges
June	3	1	0
Robert	2	2	1
Lily	0	1	2
David	1	0	3

```
In [17]: df.sort_index(ascending=False)
```

```
Out[17]:
```

	apples	oranges	bananas
Robert	2	1	2
Lily	0	2	1
June	3	0	1
David	1	3	0

```
In [18]: df.sort_index(axis=1, ascending=False)
```

```
Out[18]:
```

	oranges	bananas	apples
June	0	1	3
Robert	1	2	2
Lily	2	1	0
David	3	0	1

7.4.2 通过元素值排序

元素值排序方法是 `sort_values()`，该方法主要的参数如下：

- ❑ `by`: 指定排序的行标签或列标签，可以是字符串或字符串列表。**注意 Series 对象没有该参数。**
- ❑ `axis`: 指定排序的轴，默认为 0。DataFrame 对象 0 为行，1 为列；Series 对象只能为 0。
- ❑ `ascending`: 排序的顺序，默认为 True。True 为升序，False 为降序。

```
In [67]: s.sort_values()
```

```
Out[67]:
```

```
a    3
c    7
b    8
dtype: int64
```

```
In [68]: s.sort_values(ascending=False)
```

```
Out[68]:
```

```
b    8
c    7
a    3
dtype: int64
```

```
In [71]: df.sort_values('apples')
```

```
Out[71]:
```

	apples	oranges	bananas
Lily	0	2	1
David	1	3	0
Robert	2	1	2
June	3	0	1

```
In [72]: df.sort_values(by='apples')
```

```
Out[72]:
```

	apples	oranges	bananas
Lily	0	2	1

David	1	3	0
Robert	2	1	2
June	3	0	1

```
In [74]: df.sort_values(['bananas','oranges'])
```

```
Out[74]:
```

	apples	oranges	bananas
David	1	3	0
June	3	0	1
Lily	0	2	1
Robert	2	1	2

```
In [75]: df.sort_values('Lily',axis=1)
```

```
Out[75]:
```

	apples	bananas	oranges
June	3	1	0
Robert	2	2	1
Lily	0	1	2
David	1	0	3

7.5 课后练习

1、如何将 Series 中每个元素的第一个字符转换为大写？

参考答案：

```
import pandas as pd
ser = pd.Series(['how', 'to', 'kick', 'ass?'])
ser.map(lambda x: x[0].upper() + x[1:])
```

2、如何计算 Series 中每个单词的字符数？

参考答案：


```
import pandas as pd
ser = pd.Series(['how', 'to', 'kick', 'ass?'])
ser.map(lambda x: len(x))
```

3、计算 Series 数据对象的平均值和标准差？

参考答案：

```
import pandas as pd
s = pd.Series(data = [1,2,3,4,5,6,7,8,9,5,3])
print("原始数据：")
print(s)
print("数据平均值：")
print(s.mean())
print("数据标准偏差：")
print(s.std())
```

4、编写一个 pandas 程序实现两个 Series 对象的加、减、乘、除计算。

参考答案：

```
import pandas as pd
ds1 = pd.Series([2, 4, 6, 8, 10])
ds2 = pd.Series([1, 3, 5, 7, 9])
print("两个Series对象相加：")
ds = ds1 + ds2
print(ds)

print("两个Series对象相减：")
ds = ds1 - ds2
print(ds)

print("两个Series对象相乘：")
ds = ds1 * ds2
print(ds)
```

```
print("两个Series对象相除：")
ds = ds1 / ds2
print(ds)
```

5、编写一个 pandas 程序实现两个 Series 对象的比较计算。

参考答案：

```
import pandas as pd
ds1 = pd.Series([2, 4, 6, 8, 10])
ds2 = pd.Series([1, 3, 5, 7, 10])
print("Series对象1:")
print(ds1)
print("Series对象2:")
print(ds2)
print("比较Series对象:")
print("相等:")
print(ds1 == ds2)
print("大于:")
print(ds1 > ds2)
print("小于:")
print(ds1 < ds2)
```

6、编写一个 pandas 程序计算 DataFrame 对象某一列的平均值。

参考答案：

```
import pandas as pd
exam_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James',
                     'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],
             'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
             'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
             'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes',
                        'no', 'no', 'yes']}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

df = pd.DataFrame(exam_data , index=labels)
```

```
print(df['score'].mean())
```

7、在第 6 题基础上对 DataFrame 对象首先按 name 列降序排列，而且按 score 列升序排列。

参考答案：

```
df.sort_values(by=['name', 'score'], ascending=[False, True])
```

8、在第 6 题基础上对 DataFrame 对象替换 qualify 列 yes 和 no 为 True 和 False。

参考答案：

```
df['qualify'] = df['qualify'].map({'yes': True, 'no': False})
```

9、读取 CSV 数据（煤矿生成历史数据 2013.csv）到 DataFrame 对象，并计算“产能”列的总和、平均值、最大值、最小值。

参考答案：

```
import pandas as pd
df = pd.read_csv(file_dir+'煤矿生成历史数据2013.csv',engine='python')
print("Sum: ",df["产能"].sum())
print("Mean: ",df["产能"].mean())
print("Maximum: ",df["产能"].max())
print("Minimum: ",df["产能"].min())
```

10、读取 CSV 数据（煤矿生成历史数据 2013.csv）到 DataFrame 对象，并示并查找“矿山名”以“P”开通的数据。

参考答案：

```
import pandas as pd
```

```
df = pd.read_csv(file_dir+'煤矿生成历史数据2013.csv', engine='python')
df[df["矿山名"].map(lambda x: x.startswith('P'))]
```

11、读取 Excel 文件数据（雇员.xlsx）到 DataFrame 对象，并根据多个给定列进行排序。

参考答案：

```
import pandas as pd
df = pd.read_excel(file_dir+'雇员.xlsx', index_col=0)
result = df.sort_values(by=['first_name', 'last_name'], ascending=[0,1])
print(result)
```

12、读取 Excel 文件数据（雇员.xlsx）到 DataFrame 对象，并按 hire_date 列对记录进行排序。

参考答案：

```
import pandas as pd
df = pd.read_excel(file_dir+'雇员.xlsx', index_col=0)
result = df.sort_values('hire_date')
print(result)
```

第8章 项目实战：搜狐证券股票数据分析

8.1 获取贵州茅台股票历史数据

`http://q.stock.sohu.com/cn/600519/lshq.shtml`

- 1、通过网络爬虫，爬取数据。
- 2、通过 `pandas.read_html()` 方法。
- 3、通过 `pandas.read_clipboard()` 方法。

8.2 获得特定时间段股票数据

query 方法

布尔数组

8.3 查询时间段内最大【成交量】日期

`max()` 取最大值。

`idxmax()` 取最大值索引。

8.4 将股票信息中【日期】字符串转换为日期类型

8.5 请按照【成交金额】排序

8.6 绘制股票【成交量】折线图

Matplotlib 数据可视化库