第 5 章

Pandas 文本与日期

5.1 字符串处理

5.1.1 字符串处理流程

在 Python 数据分析过程中，经常要对字符串数据进行处理，例如对字符串的删除、拆分、组合、查找、匹配、替换、计数等应用，这其中的很多应用具备正则表达式的功能。当相关字符串方法已知可用正则表达式时它的默认参数为 regex＝True，可用 regex＝False 关掉正则表达式。Pandas 中字符串方法的整体使用流程及说明如图 5-1 所示。

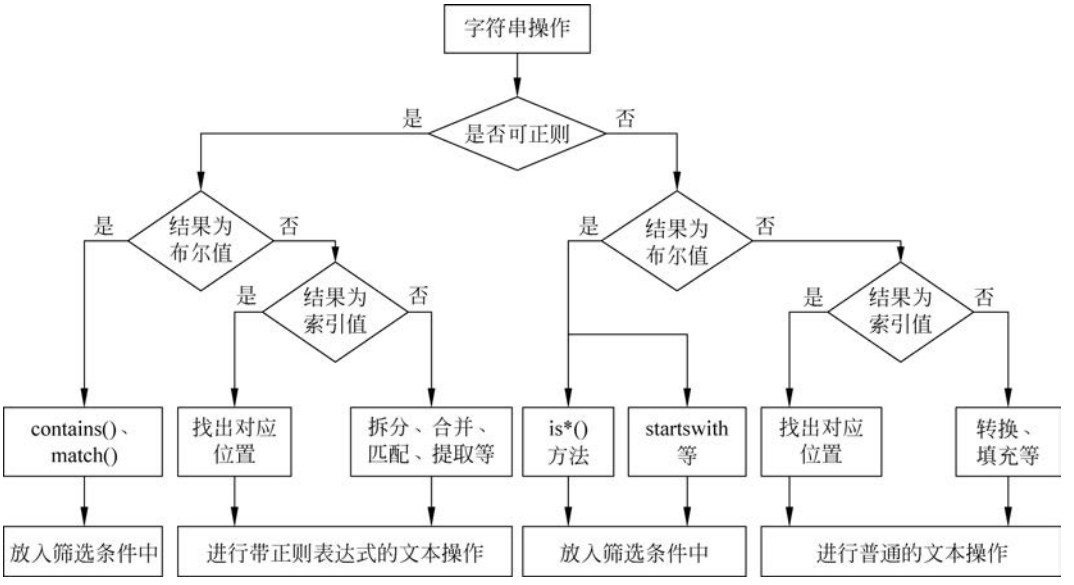

图 5-1 Pandas 中字符串方法的整体使用流程及说明

在 Pandas 中主要是以 Series 为单位来处理字符串的，它的语法为 Series. str. str 方法()。例如 Series. str. split()，代表的是 Pandas 中以某列为单位进行字符串拆分。在 Series 中，

直接对列用 split()来分列是不允许的(系统会报错 AttributeError: 'Series' object has no attribute 'Split'),但如果先用.str 将这一列转换为类似字符串的格式,则不会有问题。假如要将这一列转换为数值列,直接对数值列进行 str 的相关操作也会报错,必须先执行 astype(str),然后进行对应的字符串操作,这样就不会有问题了,即 Series.astype(str).str.str 方法()。

在 Series.str 的 str 方法中,str 是转换器。在 Pandas 中,有 3 个功能强大的转换器(或称访问器):str、dt、cat。str 转换器用于处理字符串对象,后面常用于连接字符串方法;dt 转换器用于处理时间对象,后面常用于连接时间属性;cat 转换器用于处理分类对象的数据。在 Pandas 中,除 replace()方法以 DataFrame 及 Series 为处理对象外,其他的字符串处理方法一般以 Series 为处理对象。

str 转换器相关方法见表 5-1。

表 5-1 str 转换器相关方法

方 法	语法说明	作 用
str.isdigit()	是否只由数字组成	判断是否
str.isdecimal()	是否只包含十进制字符	判断是否
str.isnumeric()	是否只由数字组成	判断是否
str.isalnum()	是否由字母和数字组成	判断是否
str.isalpha()	字符串至少包含一个字符且所有字符都是字母(汉字)	判断是否
str.islower()	至少包含一个小写字母,且不包含大写字母	判断是否
str.isupper()	至少包含一个大写字母,且不包含小写字母	判断是否
str.istitle()	所有单词以大写字母开头,其余小写	判断是否
str.isspace()	只包含空白符	判断是否
str.startswith()	以某指定的字符或字符串开头	判断是否
str.endswith()	以某指定的字符或字符串结尾	判断是否
str.get._dummies	用于数据的离散特征取值,返回的值为 0 或 1	数值计算
str.len()	计算字符串中每个元素的长度	数值计算
str.index()	计算字符串首次出现的索引位置	位置计算
str.rindex()	计算字符串最后一次出现的位置	位置
str.find()	找到字符串首次出现的索引位置,如果未找到,则返回-1	位置
str.rfind()	找到字符串首次出现的索引位置,如果未找到,则返回-1	位置
str.get()	从字符串中提取元素	提取
str.strip()	删除字符串左右两边的空白字符(含换行符)	剪切
str.lstrip()	删除字符串左边的空白字符(含换行符)	剪切
str.rstrip()	删除字符串右边的空白字符(含换行符)	剪切
str.slice()	按下标截取字符串	剪切
str.slice_replace()	按下标替换	剪切
str.removeprefix()	删除字符串中的前缀	剪切
str.removesuffix()	删除字符串中的后缀	剪切
str.repeat()	按指定的次数重复字符串	重复

续表

方 法	语 法 说 明	作 用
str.partition()	在分隔符第 1 次出现的地方拆分字符串	拆分
str.rpartition()	在分隔符最后出现的地方拆分字符串	拆分
str.join()	以指定的字符串为分隔符并生成一个新的字符串	拼接
str.cat()	合并多列的字符串	拼接
str.capitalize()	首字母大写	转换
str.title()	(有分隔符分隔时)字符串内所有单词的首字母大写	转换
str.lower()	字母全部小写	转换
str.upper()	字母全部大写	转换
str.swapcase()	大小写互换	转换
str.swap()	按指定的行宽换行字符串	转换
str.casefold()	将指定的字符串转换为大小写折叠	转换
str.translate()	通过给定的映射表映射字符串中的所有字符	转换
str.normalize()	返回字符串的 Unicode 标准形式	转换
str.pad()	字符串的左右补齐	填充
str.center()	字符串居中填充	填充
str.ljust()	字符串左对齐填充	填充
str.rjust()	字符串右对齐填充	填充
str.zfill()	在字符串前面填充	填充
str.repeat()	对字符串指定重复的次数	填充
str.warp()	在指定的位置加回车符	填充
str.decode()	使用指定的编码解码字符串	解码
str.encode()	使用指定的编码对字符串进行编码	编码

以 str.cat()的应用为例。Pandas 中的 str.cat()的用法如下：

```
Series.str.cat(others = None, sep = None, na_rep = None, join = 'left')
```

参数说明：others 参数为 Series、Index、DataFrame、np.ndarray 或 list-like；sep 参数为 str，默认值为''。na_rep 参数为 str 或 None，默认值为 None。join 参数为'left'、'right'、'outer'、'inner'，拼接方式的默认值为 left。

str.cat()中 others 参数的常见用法，如图 5-2 所示。

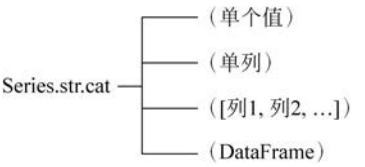

图 5-2 cat()方法的常见用法

以拼接多个列为例，代码如下：

```
#ch05d001.ipynb
import pandas as pd
df = pd.read_excel(
    r'D:\数据源\B 文件\文本处理.xlsx',
    sheet_name = 1
).head(3)
```

```
df['合并列'] = df['运单编号'].str.cat([df['城市'],df['楼牌号']],sep = '、')
df.iloc[:,[0,4]]
```

返回的值如下：

```
    运单编号                  合并列
0  YD001  YD001、北京、AA02 幢 02 楼 201
1  YD003  YD003、上海、aA03 幢 03 楼 301
2  YD006  YD006、广州、Aa04 幢 04 楼 401
```

以拼接 DataFrame 为例，代码如下：

```
df1 =  df.iloc[:,[0,4]]
df['合并列 1'] = df['字符标识'].str.cat(df1,sep = '、')
df.iloc[:,[0,5]]
```

返回的值如下：

```
    运单编号                                  合并列 1
0  YD001     ABC123、YD001、YD001、北京、AA02 幢 02 楼 201
1  YD003     b1A32C、YD003、YD003、上海、aA03 幢 03 楼 301
2  YD006  1c2a3abb、YD006、YD006、广州、Aa04 幢 04 楼 401
```

5.1.2 正则表达式

正则是 Regular Expression(正则表达式)的简写，正则表达式通过一些特定的元字符实现强大、便捷与高效的文本匹配、查找、替换等功能，因此，正则表达式已经成为所有主流编程语言的必备项。很值得去认真学习与了解。

正则表达式由“元字符”和其他“普通文本字符”两部分组成，其中，正则表达式中的“元字符”主要分为基本元字符、数字元字符、位置元字符、特殊元字符等。

1. 基本元字符

基本元字符及其语法说明见表 5-2。

表 5-2 基本元字符及语法说明

元 字 符	语 法 说 明
.	(除换行符以外的)任意字符
|	逻辑或
[]	字符集合中的任一字符
[^]	不是字符集合中的任一字符
-	区间定义
\	转义符
()	生成子表达式

2. 数字元字符

常见的数字元字符及其语法说明见表 5-3。

表 5-3 数字元字符及其语法说明

元 字 符	语 法 说 明
*	零次或多次(贪婪模式)
*?	* 的懒惰模式
+	一次或多次(贪婪模式)
+?	+ 的懒惰模式
?	前一字符的零次或一次
{n}	n 次重复
{m,n}	重复 m 到 n 次
{n,}	重复 n 次到更多次
{n,}?	{n,} 的懒惰模式

3. 特殊元字符

特殊元字符及其语法说明见表 5-4。

表 5-4 特殊元字符及其语法说明

元 字 符	语 法 说 明
\d	任意数字,等价于[0-9]
\D	不是数字,等价于[^0-9]
\s	空白字符,等价于[\n\r\t\v]
\S	不是空白字符,等价于[^\n\r\t\v]
\w	任意字母、数字或下画线,等价于[a-zA-Z0-9_]
\W	不是任意字母、数字或下画线,等价于[^a-zA-Z0-9_]
\f	换页符
\n	换行符
\r	回车符
\t	制表符

4. 位置元字符

位置元字符及其语法说明见表 5-5。

表 5-5 位置元字符及其语法说明

元 字 符	语 法 说 明
^	开始
$	结束
\A	字符串的开头(忽略 re.M)
\Z	字符串的结尾(忽略 re.M)
\b	单词的边界
\B	不是\b

5. 追溯与查找

各类正则断言均属于分组不捕获,匹配的结果为(零宽度的)位置,其作用是给指定位置

添加限定的条件，如图 5-3 所示。

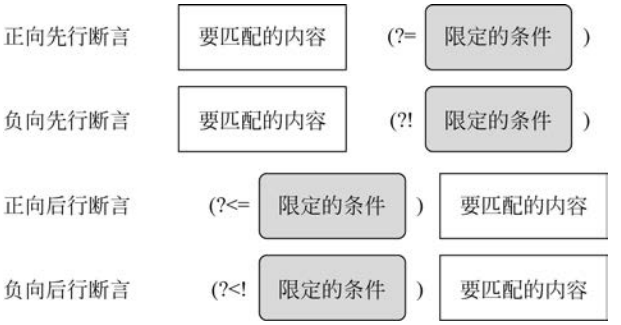

图 5-3　正则断言

追溯与查找元字符及相关语法说明见表 5-6。

表 5-6　追溯与查找元字符及相关语法说明

元　字　符	语 法 说 明
?=	名称：正向先行断言(正前瞻) 语法：(?=pattern) 作用：匹配 pattern 表达式前面的内容，不返回本身 举例：a(?=b)，先行断言，a 只有在 b 前面才匹配
?!	名称：负向先行断言(负前瞻) 语法：(?!pattern) 作用：匹配非 pattern 表达式前面的内容，不返回本身 举例：a(?!b)，先行否定断言，a 只有不在 b 前面才匹配
?<=	名称：正向后行断言(正后顾) 语法：(?<=pattern) 作用：匹配 pattern 表达式后面的内容，不返回本身 举例：(?<=b)a，后行断言，a 只有在 b 后面才匹配
?<!	名称：负向后行断言(负后顾) 语法：(?<!pattern) 作用：匹配非 pattern 表达式后面的内容，不返回本身。 举例：(?<!b)a，后行否定断言，a 只有不在 b 后面才匹配

5.1.3　文本正则应用

1. 包含

Pandas 中的 str.contains()用法：

```
Series.str.contains(pat, case = True, flags = 0, na = None, regex = True)
```

参数说明：pat 参数为字符串或正则表达式。case 参数的默认值为 True(对大小写敏感)。flags 参数的默认值为 0，可用值为 re.I、re.M、re.S、re.U、re.A 等。na 参数的默认值

为 None,用于 na 值的替换。regex 的默认值为 True(启用正则表达式)。返回的值为 True 或 False。

在收货地址列查看是否存在 3 个连续的数字,代码如下:

```
#ch05d002.ipynb
import pandas as pd
df = pd.read_excel(r'D:\数据源\B 文件\文本处理.xlsx').head(3)
df['收货地址'].str.contains('\d{3}')
```

返回的值如下:

```
0    True
1    True
2    True
Name:收货地址, dtype: bool
```

布尔值用作[]访问器的筛选条件返回 DataFrame。应用举例,代码如下:

```
df[df['收货地址'].str.contains('\d{3}')]
```

返回的值如下:

```
    运单编号          收货地址      字符标识
0  YD001 北京路 AA02 幢 02 楼 201    ABC123
1  YD003 上海路 aA03 幢 03 楼 301    b1A32C
2  YD006 广州路 Aa04 幢 04 楼 401  1c2a3abb
```

2. 匹配

str.match()在字符串的开始进行匹配,str.fullmatch()对字符串从头匹配到尾,如果匹配成功,则返回正则表达式对象,语法如下:

```
Series.str.match(pat, case = True, flags = 0, na = None)
#Series.str.fullmatch(pat, case = True, flags = 0, na = None)
```

参数说明:pat 参数为字符串或正则表达式。case 参数的默认值为 True。flags 参数的默认值为 0,na 参数的默认值为 None。返回的值为 True 或 False。

字符串匹配。应用举例如下:

```
df['收货地址'].str.match('\w{3}')
```

返回的值如下:

```
0    True
1    True
2    True
Name:收货地址, dtype: bool
```

3. 计数

Pandas 中的 str.count()的用法如下:

```
Series.str.count(pat, flags = 0)
```

参数说明：pat 为字符串或正则表达式，flags 参数的默认值为 0。用于统计 Series 或 Index 中所匹配的字符出现的次数。

在收货地址列统计三位数的数值个数，代码如下：

```
df = pd.read_excel(r'D:\数据源\B 文件\文本处理.xlsx').head(3)
df['个数统计'] = df['收货地址'].str.count('\d{3}')
df
```

返回的值如下：

```
    运单编号          收货地址      字符标识  个数统计
0  YD001 北京路 AA02 幢 02 楼 201    ABC123     1
1  YD003 上海路 aA03 幢 03 楼 301    b1A32C     1
2  YD006 广州路 Aa04 幢 04 楼 401  1c2a3abb     1
```

4. 拆分

Pandas 中的 str.split()的用法如下：

```
Series.str.split(pat = None, n = -1, expand = False)
```

参数说明：pat 为字符串或正则表达式。n 的默认值为－1，按最大可拆分次数进行拆分。expand 的默认值为 False；当 expand＝True 时，返回的值为 DataFrame。与 split()的用法类似，但方向相反的是 rsplit()，拆分的方向由右向左。

按分隔符拆分。以“路”为分隔符，代码如下：

```
#ch05d003.ipynb
import pandas as pd
df = pd.read_excel(r'D:\数据源\B 文件\文本处理.xlsx').head(3)
df['收货地址'].str.split('路')
```

返回的值如下：

```
0    [北京, AA02 幢 02 楼 201]
1    [上海, aA03 幢 03 楼 301]
2    [广州, Aa04 幢 04 楼 401]
Name:收货地址, dtype: object
```

如果需要扩展到 DataFrame 并命名列名，则代码如下：

```
s = df['收货地址'].str.split('路', expand = True)
df['城市'] = s[0]
df['楼牌号'] = s[1]
df
```

或者采用以下自定义函数的写法，代码如下：

```
def y(x):
```

```
    x['城市'],x['楼牌号'] = x['收货地址'].split('路')
    return x
df = df.apply(y,axis = 1)
df
```

apply()函数调用的语法说明见下一章节。以上代码返回的值如下：

```
  运单编号        收货地址        字符标识  城市      楼牌号
0  YD001  北京路 AA02 幢 02 楼 201  ABC123    北京  AA02 幢 02 楼 201
1  YD003  上海路 aA03 幢 03 楼 301  b1A32C    上海  aA03 幢 03 楼 301
2  YD006  广州路 Aa04 幢 04 楼 401  1c2a3abb  广州  Aa04 幢 04 楼 401
```

以字母为分隔符，代码如下：

```
df['收货地址'].str.split('[a-zA-Z]+', expand = True)
```

返回的值如下：

```
    0          1
0 北京路   02 幢 02 楼 201
1 上海路   03 幢 03 楼 301
2 广州路   04 幢 04 楼 401
```

Power Query菜单中的几种拆分应用。采用Pandas正则拆分列“按字符数”，代码如下：

```
df['收货地址'].str.split('(?<=...)',1)
```

拆分列方式“按位置”，代码如下：

```
df['收货地址'].str.split('...',1)
```

拆分列方式“从小写到大写”，代码如下：

```
df['收货地址'].str.split('(?<=[a-z])(?=[A-Z])')
```

拆分列方式“从大写到小写”，代码如下：

```
df['收货地址'].str.split('(?<=[A-Z])(?=[a-z])')
```

拆分列方式“非数字到数字”，代码如下：

```
df['收货地址'].str.split('(?<=\D)(?=\d)')
```

拆分列方式“数字到非数字”，代码如下：

```
df['收货地址'].str.split('(?<=\d)(?=\D)')
```

5. 查找全部

Pandas中的str.findall()的用法如下：

```
Series.str.findall(pat, flags=0)
```

参数说明：str.findall()返回的值为列表。参数 flags 是正则表达式的匹配方式，例如 re.I、re.M、re.S、re.U、re.A 等。

采用常规匹配方式，获取收货地址中的幢、楼信息，代码如下：

```
#ch05d004.ipynb
import pandas as pd
df = pd.read_excel(r'D:\数据源\B文件\文本处理.xlsx').head(3)
df['收货地址'].str.findall('\d+\D+')
```

返回的值如下：

```
0    [02幢, 02楼]
1    [03幢, 03楼]
2    [04幢, 04楼]
Name:收货地址, dtype: object
```

添加一个捕获组，将分组内的内容返回列表，代码如下：

```
df['收货地址'].str.findall('(\d+)\D+')
```

返回的值如下：

```
0    [02, 02]
1    [03, 03]
2    [04, 04]
Name:收货地址, dtype: object
```

添加多个捕获组，返回的是由元组组成的列表，每个元组中的元素是每个分组所返回的值，代码如下：

```
df['收货地址'].str.findall('(\d+)(\D+)')
```

返回的值如下：

```
0    [(02,幢), (02, 楼)]
1    [(03,幢), (03, 楼)]
2    [(04,幢), (04, 楼)]
Name:收货地址, dtype: object
```

6. 提取

Pandas 中的 str.extract()的用法如下：

```
Series.str.extract(pat, flags=0, expand=True)
```

参数说明：pat 参数为字符串或正则表达式。参数 flags 是正则表达式的匹配方式，例如 re.I、re.M、re.S、re.U、re.A 等。expand 的默认值为 True，当 expand＝True 时返回 DataFrame，每个捕获组有一列；当 expand＝False 时，如果有一个捕获组，则返回一个 Series/Index；如果有多个捕获组，则返回 DataFrame。

导入数据，提取收货地址中的第 1 组数值，代码如下：

```
#ch05d005.ipynb
import pandas as pd
df = pd.read_excel(r'D:\数据源\B文件\文本处理.xlsx').head(3)
df['收货地址'].str.extract('(\d+)\D+')
```

返回的值如下：

```
    0
0  02
1  03
2  04
```

提取收货地址中第1组数值与第1组文本，代码如下：

```
df['收货地址'].str.extract('(\d+)(\D+)')
```

返回的值如下：

```
    0  1
0  02  幢
1  03  幢
2  04  幢
```

在分组内添加分组命名。命名分组是Python中正则的一种用法，其语法结构为“(?P<名称>正则表达式)”。应用举例，代码如下：

```
df['收货地址'].str.extract('(?P<幢数>\d+)(?P<幢>\D+)')
```

返回的值如下：

```
  幢数  幢
0  02  幢
1  03  幢
2  04  幢
```

采用命名分组形式，提取收货地址中的幢、楼信息；未被命名的分组会以数值的形式显示，代码如下：

```
df['收货地址'].str.extract('(?P<幢数>\d+)(?P<幢>\D+)(\d+)(\D+)(\d+)')
```

返回的值如下：

```
  幢数  幢   2  3    4
0  02  幢  02  楼  201
1  03  幢  03  楼  301
2  04  幢  04  楼  401
```

7. 提取全部

Pandas中的str.extractall()的用法如下：

```
Series.str.extractall(pat, flags=0)
```

str.extract()方法用于向列方向扩展。str.extractall()方法用于向行方向扩展。应用举例如下：

```
df['收货地址'].str.extractall('(?P<数字>\d+)(?P<文本>\D+)')
```

返回的值如下：

```
          数字 文本
  match
0 0       02  幢
  1       02  楼
1 0       03  幢
  1       03  楼
2 0       04  幢
  1       04  楼
```

8. 替换

Pandas 中的 str.replace()的用法如下：

```
Series.str.replace(pat, repl, n = - 1, case = None, flags = 0, regex = None)
```

str.replace()方法用于替换字符串中出现的每个模式或正则表达式，等价于 str.replace()或 re.sub()。

应用举例如下：

```
df['收货地址'].str.replace('路\w{5}','一线城市')
```

返回的值如下：

```
0  北京一线城市 02 楼 201
1  上海一线城市 03 楼 301
2  广州一线城市 04 楼 401
Name:收货地址, dtype: object
```

5.2 日期和时间

Pandas 继承了 NumPy 库和 datetime 库的时间相关模块，提供了 Timestamp、Period、Timedelta、DatetimeIndex、PeriodIndex、TimedeltaIndex 这 6 种时间相关的类，使其能更高效地处理时间序列数据。Pandas 支持时区，使用的是 datetime64[ns]数据类型，可以精确到毫秒、纳秒，能够轻松处理金融等对时间精度有要求的行业。

在这 6 个类中，Timestamp 是 Pandas 中最基础的，也是最常用的时间序列类型，它是以时间戳为索引的 Series。当创建一个带有 DatetimeIndex 的 Series 时，Pandas 就会知道对象是一个时间序列。

Pandas 中这 6 个时间序列类型的数据结构与数据类型的对照见表 5-7。

表 5-7 Pandas 时间序列数据结构

概 念	标 量 类	数 据 类 型	索 引	创 建 方 法
时间点	Timestamp	datetime64[ns]，datetime64[ns,tz]	DatetimeIndex	pd. to_datetime、pd. date_range
时间段	Period	period[freq]	PeriodIndex	df. to_period、pd. period_range
时间差	Timedelta	Timedelta64[ns]	TimedeltaIndex	pd. to_timedelta、pd. timedelta_range
时间偏移	DateOffset	None	None	pd. DateOffset

以下是表 5-7 的内容说明。

(1) 时间点(Datetime)，也可以称为时间戳(Timestamp)。一系列的时间戳构成了 DatetimeIndex。在 Series 中这些数据的类型为 datetime64[ns]，如果存在时区设置，则在 Series 中这些数据的类型为 datetime64[ns,tz]。

(2) 持续的时间段(Period)都由 start 和 end 两部分组成。一系列的时间段构成了 PeriodIndex。

(3) 两个时间点的差值代表的是时间差(Timedelta)，代表的是某事件的持续时间。一系列的时间戳构成了 TimedeltaIndex。

(4) 如果需要在某一日期的基础上进行未知日期的计算，则可以采用日期偏移(DateOffset)。

Pandas 中时序分析常用的流程如图 5-4 所示。

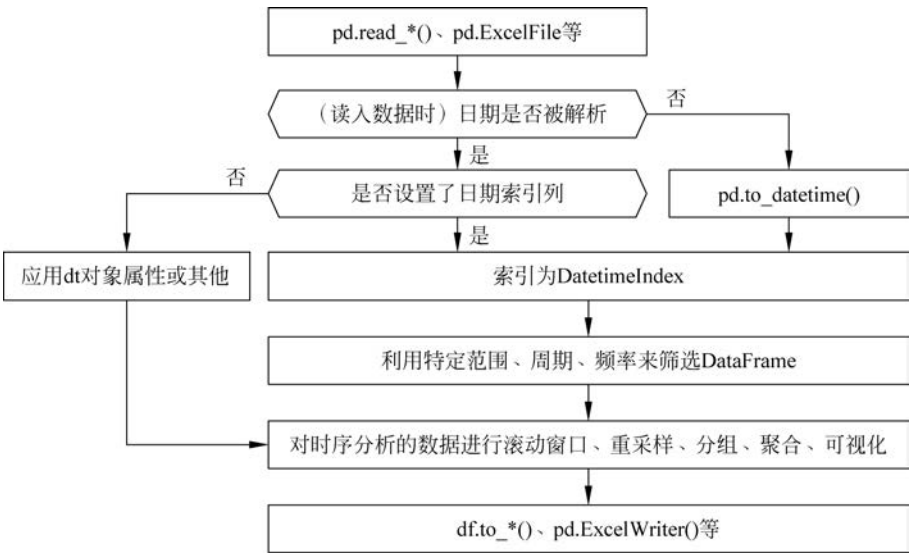

图 5-4 Pandas 中时序分析常用的流程

5.2.1 时间点

1. 时间戳

在 Pandas 中，可用 pd.Timestamp()进行解析、转换、创建单个时间戳。以 pd.Timestamp 为例，代码如下：

```
#ch05d006.ipynb
import pandas as pd
pd.Timestamp(2022,12,13)
#pd.Timestamp('2022/12/13')
#pd.Timestamp(year = 2022,month = 12,day = 13)
#pd.Timestamp(1670889600,unit = 's')
```

一个完整的时间戳是由年、月、日、时、分、秒等组成，返回的值如下：

```
Timestamp('2022-12-13 00:00:00')
```

利用 pd.Timestamp()获取当前时间点，代码如下：

```
pd.Timestamp('now')
pd.Timestamp('today')
```

利用 pd.Timestamp()获取当日日期、当前时间及星期。应用举例，代码如下：

```
pd.Timestamp('now').date()
pd.Timestamp('today').time()
pd.Timestamp('today').day_name()
```

2. 创建日期时间

在 Pandas 中，可用 pd.to_datetime()创建日期时间。pd.to_datetime 的第 1 个参数可为 str、int、float、datetime、list、1-d array、Series、DataFrame、dict-like、tuple 等。以列表中的字符串全部转换为日期时间为例，代码如下：

```
pd.to_datetime(['2022-12-13','12/13/2022','2022.12.13','13/12/2022','Dec 13, 2022'])
```

列表中存在多个时间戳值，从而构成了一个 DatetimeIndex，数据类型为 datetime64[ns]，freq 为 None，返回的值如下：

```
DatetimeIndex(['2022-12-13', '2022-12-13', '2022-12-13', '2022-12-13',
               '2022-12-13'], dtype = 'datetime64[ns]', freq = None)
```

3. 日期范围

pd.date_range()、pd.bdate_range()用于生成指定长度的 DatetimeIndex；二者的区别在于是否包含周六、周日。4 个主要参数如表 5-8 所示。

表 5-8 date_range 的参数说明

参 数	参 数 说 明
Start	开始日期,可省
End	结束日期,可省
Periods	固定时期,取值为整数或 None
Freq	日期偏移量(频率),取值为 string 或 DateOffset

利用 pd.date_range()函数创建 5 个日期列,代码如下:

```
#ch05d007.ipynb
import pandas as pd
df = pd.DataFrame(
   {
       'Date1':pd.date_range(start = '2022-12-12',periods = 3),
       'Date2':pd.date_range(end = '2022-12-15',periods = 3, freq = 'D'),
       'Date3':pd.date_range(end = '2022-12-15',periods = 3, freq = 'A'),
       'Date4':pd.date_range(end = '2022-12-15',periods = 3, freq = 'QS'),
       'Date5':pd.date_range(end = '2022-12-15',periods = 3, freq = 'MS'),
   }
)
df
```

pd.date_range()类似于 Power Query 中 List.Dates()的功能,用于生成一个给定开始时间和持续时长的日期列表。返回的值如下:

```
       Date1      Date2      Date3      Date4      Date5
0 2022-12-12  2022-12-13 2019-12-31 2022-04-01 2022-10-01
1 2022-12-13  2022-12-14 2020-12-31 2022-07-01 2022-11-01
2 2022-12-14  2022-12-15 2021-12-31 2022-10-01 2022-12-01
```

利用 dtypes 属性查看各列的数据类型,代码如下:

```
df.dtypes
```

返回的值如下:

```
Date1    datetime64[ns]
Date2    datetime64[ns]
Date3    datetime64[ns]
Date4    datetime64[ns]
Date5    datetime64[ns]
dtype: object
```

在 DataFrame 中利用 pd.date_range()创建 DatetimeIndex,代码如下:

```
#ch05d008.ipynb
import pandas as pd
df = pd.read_excel(
   r'D:\数据源\B 文件\订单表.xlsx',
   usecols = ['接单时间','产品','订单数'],
```

```
    index_col = '接单时间'
).head(3)

pd.date_range(df.index.min(),df.index.max(),freq = '4M')
```

返回的 freq 为'4M',返回的值如下：

```
DatetimeIndex(['2020 - 04 - 30', '2020 - 08 - 31', '2020 - 12 - 31', '2021 - 04 - 30',
    '2021 - 08 - 31'],dtype = 'datetime64[ns]', freq = '4M')
```

在以上代码中 freq 参数的对照表见表 5-9。

表 5-9　时间序列频率对照表

别　　名	英 文 描 述	中 文 描 述
D/B	Day/BusinessDay	日历日的每天/工作日的每天
H/T(或 Min)/S	Hour/Minute/Second	时/分/秒
M/BM	MonthEnd/BusinessMonthEnd	日历日的月末/工作日的月末
MS/BMS	MonthStart/BusinessMonthStart	日历日的月初/工作日的月初
W-MON	Week-Monday	每周从星期一开始计算,其他的有 W-TUE……
WOM-1MON	WeekOfMonth	在本月的第 1 周创建按周分隔的日期,例如 WOM-3FRI 代表每月的第 3 个星期五
Q-JAN/BQ-JAN	QuarterEnd/BusinessQuarterEnd	JAN 表示月份结束的季度,也可以是 FEB……
QS-JAN/QBS-JAN	QuarterStart/ BusinessQuarterStart	JAN 表示月份结束的季度,也可以是 FEB……
A-JAN/BA-JAN	BusinessYearEnd/YearStart	JAN 表示月份结束的季度,也可以是 FEB……
AS-JAN/BAS-JAN	YearStart/BusinessYearStart	JAN 表示月份结束的季度,也可以是 FEB……

表 5-9 中的各类频率组合,可用图 5-5 加深理解与记忆。例如 A/Q/M 单字母时代表的是其对应频率的 End,加上 s 时则为其对应频率的 start。

组合（元素）								
B	S	—	代表	频率（别名）	频率（别名）组合			
Business	Start		yeAr	A	BA	AS	BAS	A-JAN
			Quarter	Q	BQ	QS	QBS	Q-JAN
			Month	M	BM	MS	BMS	

图 5-5　时间频率组合规律说明

为加深理解,仍以 2022-12 为例,表 5-10 是 M、BM、MS、BMS 的对照说明。

表 5-10　时间序列频率对照表

别　　名	代表的频率	(2022-12)举例
M	月末	2022-12-31
BM	月末的工作日	2022-12-30
MS	月初	2022-12-1
BMS	月初工作日	2022-12-1

4. 解析日期

在 Pandas 中，在 pd.read_excel()读取过程中，可通过 parse_dates 手动设置或系统自动解析，将对应的文本型日期时间数据解析为日期时间数据；甚至可以将导入 DataFrame 的日期时间数据设置为索引列，代码如下：

```
#ch05d009.ipynb
import pandas as pd
df = pd.read_excel(
    r'D:\数据源\B文件\订单表.xlsx',
    usecols=['接单时间','产品','订单数'],
    #parse_dates=True,
    index_col='接单时间'
).head(3)
df.index
```

生成 DatetimeIndex，dtype 为 datetime64[ns]，freq 为 None。如果以上代码返回的数据类型为 Object，则可以启用 parse_dates=True 进行日期解析，返回的值如下：

```
DatetimeIndex(['2020-04-28', '2020-05-31', '2020-06-30'], dtype='datetime64[ns]',
name='接单时间', freq=None)
```

将数据导入 DataFrame 时，日期时间数据会被自动解析为 datetime64[ns]，代码如下：

```
df = pd.read_excel(
    r'D:\数据源\B文件\订单表.xlsx',
    #parse_dates=True,
    usecols=['接单时间','产品','订单数'],)
df.dtypes
```

查看数据类型，返回的值如下：

```
接单时间    datetime64[ns]
产品              object
订单数             int64
dtype: object
```

将接单时间列设置为索引列并查看索引列，代码如下：

```
df1 = df.set_index(df['接单时间'])
df1.head(3).index
```

DatetimeIndex 的 freq 为 None，返回的值如下：

```
DatetimeIndex(['2020-04-28', '2020-05-31', '2020-06-30'], dtype='datetime64[ns]',
name='接单时间', freq=None)
```

在 Pandas 中，可使用 dt 转换器访问 datetime 对象并对 year、month、day、hour、minute、second、quarter 等属性进行访问，见表 5-11。

表 5-11 Pandas 时间戳属性

细　分	内　容
常用	year、month、day、hour、minute、second、value
偶用	asm8、day_of_week、day_of_year、dayofweek、dayofyear、days_in_month、daysinmonth、freq、fold、freqstr、is_leap_year、is_month_end、is_month_start、is_quarter_end、is_quarter_start、is_year_end、is_year_start、microsecond、nanosecond、quarter、tz、week、weekofyear、start_time、end_time

以 is 开头的属性均为判断是否满足条件，返回的值为 True 或 False。

在 Pandas 中，可使用 dt 转换器访问 datetime 对象并对其显示格式进行设置。Python 中 datetime 的常见格式说明见表 5-12。

表 5-12 datetime 的常见格式说明

类　型	格 式 说 明
%F	%Y-%m-%d 的简写，解析到纳秒
%D	%m%d%y 的简写
%Y	四位的年份
%y	两位的年份
%m	两位的月份
%d	两位的日期
%H	24h 制的小时
%I	12h 制的小时
%M	两位的分钟
%S	秒
%w	星期几(星期天为 0)
%W	一年中第几周(星期一为每周的第一天)
%U	一年中第几周(星期天为每周的第一天)
%z	以＋HHMM 或－HHMM 的 UTC 时区偏移，如果没有时区，则为空

(1) 利用 dt 转换器进行属性访问。利用 assign()方法创建连续的多列，代码如下：

```
df = df.assign(
    年月 = df.接单时间.dt.strftime('%Y-%m'),
    年 = df.接单时间.dt.year,
    季 = df.接单时间.dt.quarter,
    月 = df.接单时间.dt.month
).head(3)
df
```

当需解析的时间数据因某些原因无法正确解析时，可采用 strftime()的参数进行强制格式转换，返回的值如下：

```
      接单时间   产品  订单数   年月     年  季  月
0 2020-04-28  蛋糕纸     2  2020-04  2020  2  4
1 2020-05-31  蛋糕纸     2  2020-05  2020  2  5
2 2020-06-30  苹果醋     2  2020-06  2020  2  6
```

(2) 利用 dt 转换器的连接方法。继续创建多列,代码如下:

```
df = df.assign(
    月名 = df.接单时间.dt.month_name(),
    星期 = df.接单时间.dt.day_name())
df
```

返回的值如下:

```
      接单时间   产品  订单数    年月     年  季  月   月名     星期
0 2020-04-28  蛋糕纸    2  2020-04  2020  2  4  April  Tuesday
1 2020-05-31  蛋糕纸    2  2020-05  2020  2  5    May   Sunday
2 2020-06-30  苹果醋    2  2020-06  2020  2  6   June  Tuesday
```

5. 日期时间索引

DatetimeIndex 具备以下优点:①可以通过切片方式快速索引;②可以通过索引的属性快速访问数据;③同频率数据的快速合并等。

(1) 当创建一个带有 DatetimeIndex 的 Series 时,Pandas 就会知道对象是一个时间序列,可以进行与索引相关操作,代码如下:

```
#ch05d010.ipynb
import pandas as pd
df = pd.read_excel(
    r'D:\数据源\B 文件\订单表.xlsx',
    usecols = ['接单时间','产品','订单数'],
    #parse_dates = True,
)

df = df.set_index(pd.to_datetime(df['接单时间']))
#df = df.set_index(df['接单时间'])

df.sort_index().loc['2020'].count()
```

需要提醒的是:为了避免索引列中数据可能存在的乱序情形,在索引与切片之前先对 DatetimeIndex 进行索引排序,返回的值如下:

```
接单时间    13
产品      13
订单数     13
dtype: int64
```

也可以对具体的年月进行访问,代码如下:

```
df = df.set_index(pd.to_datetime(df['接单时间']))
#df = df.set_index(df['接单时间'])
df.sort_index().loc['2020-10']
```

返回的值如下:

```
                接单时间   产品  订单数
接单时间
2020-10-02 2020-10-02  油漆     9
2020-10-04 2020-10-04  包装绳    7
2020-10-04 2020-10-04  钢化膜   11
```

或对具体的日期进行访问,代码如下:

```
df = df.set_index(pd.to_datetime(df['接单时间']))
#df = df.set_index(df['接单时间'])
df.loc['2020-10-4']
```

返回的值如下:

```
                接单时间   产品  订单数
接单时间
2020-10-04 2020-10-04  包装绳    7
2020-10-04 2020-10-04  钢化膜   11
```

对第 4 季的数据进行索引,代码如下:

```
df.loc['2020Q4']
```

返回的值如下:

```
                接单时间   产品  订单数
接单时间
2020-10-02 2020-10-02  油漆     9
2020-10-04 2020-10-04  包装绳    7
2020-10-04 2020-10-04  钢化膜   11
```

(2) 当不存在时间索引列时,也可以利用 dt 转换器进行访问,代码如下:

```
df = pd.read_excel(
   r'D:\数据源\B文件\订单表.xlsx',
   usecols=['接单时间','订单数','入库数'],
   parse_dates=True,
)
df.loc[(df.接单时间.dt.year==2020) & (df.接单时间.dt.quarter==4)]
```

以上代码返回的值如下:

```
      接单时间  订单数  入库数
10 2020-10-02    9    9
11 2020-10-04    7    7
12 2020-10-04   11   11
```

5.2.2 时间段

1. 期间

pd.Period()用于创建一个具体的时段,代码如下:

```
pd.Period(2021,freq='A-Feb')
```

返回的值如下：

```
Period('2021', 'A-FEB')
```

以 Q 为频率，代码如下：

```
pd.Period('2022-12-13', freq = "Q")
```

返回的值如下：

```
Period('2022Q4', 'Q-DEC')
```

2. 创建期间

在 Pandas 中有 Series.to_period()和 DataFrame.to_period()两种常见用法，s.to_period()或 df.to_period()用于描述该日期处于哪个时期。Series 的主要参数为 freq，DataFrame 中还有 axis 参数。s.to_period()用法的应用举例，代码如下：

```
A = pd.date_range('2022-12-12',periods=3,freq='3M')
A.to_period()
```

返回的值如下：

```
PeriodIndex(['2022-12', '2023-03', '2023-06'], dtype='period[3M]')
```

df.to_period()用法的应用举例，以时间段为分组计算依据，代码如下：

```
df = pd.read_excel(
    r'D:\数据源\B文件\订单表.xlsx',
    usecols=['接单时间','订单数','入库数'],
    parse_dates=True,
    index_col='接单时间'
)
df.to_period('Q').head(3)
```

返回的值如下：

```
          订单数  入库数
接单时间
2020Q2     2     2
2020Q2     2     2
2020Q2     2     2
```

利用索引及其属性作为分组的依据，然后计算订单与入库数，代码如下：

```
df.groupby(df.index.year).sum()
```

或者利用 to_period()的频率(频率为年的简码为 A)，代码如下：

```
df.to_period('A').reset_index().groupby('接单时间').sum()
```

返回的值如下：

```
      订单数   入库数
接单时间
2020    61    57
2021 4   140
```

3. 期间范围

pd.period_range(start＝None，end＝None，periods＝None，freq＝None，name＝None)用于创建固定频率的 PeriodIndex。创建一个 Period_range，代码如下：

```
A = pd.period_range('2022/7/1',periods = 3,freq = 'M')
A
```

生成 PeriodIndex，返回的值如下：

```
PeriodIndex(['2022 - 07', '2022 - 08', '2022 - 09'], dtype = 'period[M]')
```

与 DatetimeIndex 的属性类似，在 PeriodIndex 中，它也有 year、quarter、month、day、hour、minute、second、weekday、weekofyear、dayofyear 等属性。以 year 属性为例，代码如下：

```
A.year
```

返回的值如下：

```
Int64Index([2020, 2020, 2020], dtype = 'int64')
```

以 dayofweek 属性为例，代码如下：

```
A.dayofweek
```

返回的值如下：

```
Int64Index([4, 0, 2], dtype = 'int64')
```

以 end_time 属性为例，代码如下：

```
A.end_time
```

返回的值如下：

```
DatetimeIndex(['2020 - 07 - 31 23:59:59.999999999',
               '2020 - 08 - 31 23:59:59.999999999',
               '2020 - 09 - 30 23:59:59.999999999'],
              dtype = 'datetime64[ns]', freq = None)
```

4. 期间索引

当创建一个带有 DatetimeIndex 的 Series 时，可以进行与索引相关操作，代码如下：

```
#ch05d011.ipynb
import pandas as pd
df = pd.read_excel(
    r'D:\数据源\B文件\订单表.xlsx',
    usecols=['接单时间','产品','订单数'],
    index_col='接单时间'
).to_period('Q')
df.head(3).index
```

返回的值如下：

```
PeriodIndex(['2020Q2', '2020Q2', '2020Q2'], dtype='period[Q-DEC]', name='接单时间')
```

对2020Q4值进行筛选，表达式如下：

```
df.loc['2020Q4']
```

返回的值如下：

```
        产品  订单数
接单时间
2020Q4  油漆     9
2020Q4  包装绳    7
2020Q4  钢化膜   11
```

5.2.3 时间差

1. 时间差

Timedelta用于创建或计算时间差，常用的时间单位有周、日、时、分、秒等。导入数据，代码如下：

```
#ch05d012.ipynb
import pandas as pd
df = pd.read_excel(
    r'D:\数据源\B文件\订单表.xlsx',
    usecols=['入库日期',],nrows=3)
df['时差'] = df['入库日期']- df['入库日期'].min()
df
```

返回的值如下：

```
                入库日期              时差
0 2020-05-26 13:54:24   0 days 00:00:00
1 2020-06-03 06:49:22   7 days 16:54:58
2 2020-07-03 00:03:21 37 days 10:08:57
```

查看数据类型，代码如下：

```
df.dtypes
```

返回的值如下：

```
入库日期      datetime64[ns]
时差       timedelta64[ns]
dtype: object
```

pd.Timedelta()的参数有多种传递方式，利用 assign()创建多列，代码如下：

```
df = df.assign(
    时差 1 = pd.Timestamp(2022,12,13) - pd.to_datetime(df['入库日期']),
    时差 2 = pd.to_datetime(df['入库日期']) - pd.Timedelta('3 days 3 hours 3 minutes'),
    时差 3 = pd.to_datetime('2022.12.13') - pd.Timedelta(5,unit = 'd'),
    时差 4 = pd.to_datetime('2022.12.13') - pd.Timedelta(days = 5),
)
df
```

Timedelta 时间差功能与用法类似于 Power Query 中的 Duration。支持的运算有与标量值相乘、与时间戳加减，以及时间差之间的加、减、乘、除。返回的值如图 5-6 所示。

	入库日期	时差	时差1	时差2	时差3	时差4
0	2020-05-26 13:54:24	0 days 00:00:00	930 days 10:05:36	2020-05-23 10:51:24	2022-12-08	2022-12-08
1	2020-06-03 06:49:22	7 days 16:54:58	922 days 17:10:38	2020-05-31 03:46:22	2022-12-08	2022-12-08
2	2020-07-03 00:03:21	37 days 10:08:57	892 days 23:56:39	2020-06-29 21:00:21	2022-12-08	2022-12-08

图 5-6　时间差

在以上代码中时间差 freq 参数的常用单位见表 5-13。

表 5-13　时间差频率对照表

频　率	代　码
周	W、w、weeks、week
天	D、d、days、day
时	H、h、hours、hour
分	T、m、minutes、minute
秒	S、s、seconds、second
毫秒	I、milli、millis、millisecond、milliseconds
微秒	U、micro、micros、microsecond、microseconds
纳秒	N、ns、nano、nanos、nanosecond、nanoseconds

毫秒、微秒、纳秒等高精度单位在金融等行业应用较多，这样的精度是 Excel 所不具备的。

查看 DataFrame 中各列的数据类型，代码如下：

```
df.dtypes
```

返回的值如下：

```
入库日期      datetime64[ns]
时差       timedelta64[ns]
时差 1      timedelta64[ns]
时差 2       datetime64[ns]
```

```
时差 3      datetime64[ns]
时差 4      datetime64[ns]
dtype: object
```

2. 时间差范围

pandas.timedelta_range(start＝None，end＝None，periods＝None，freq＝None，name＝None，closed＝None)，返回固定频率的 TimedeltaIndex，默认为天数。应用举例如下：

```
A = pd.timedelta_range(start = '1 day', end = '2 days', freq = '16H')
A
```

freq 为 16H，返回的值如下：

```
TimedeltaIndex(['1 days 00:00:00', '1 days 16:00:00'], dtype = 'timedelta64[ns]', freq = '16H')
```

3. 时间差索引

对"时差 1"列中的信息进行提取，代码如下：

```
df['相差天数'] = pd.TimedeltaIndex(df['时差 1']).days
df['间隔天数'] = df['时差 1'].dt.days
df.iloc[:,[0,2,6,7]]
```

常用的时间差属性有 days、seconds 等，返回的值如下：

```
      接单时间          时差 1   相差天数   间隔天数
0 2020 - 04 - 28 28 days 13:54:24     28     28
1 2020 - 05 - 31   3 days 06:49:22      3      3
2 2020 - 06 - 30   3 days 00:03:21      3      3
```

5.2.4 时间偏移

1. 日期偏移

pd.DateOffset()所实现的功能类似于 pd.Timedelta()，但二者实现的方式有所不同。应用举例如下：

```
#ch05d013.ipynb
import pandas as pd
df = pd.read_excel(
   r'D:\数据源\B 文件\订单表.xlsx',
usecols = ['接单时间','入库日期'],nrows = 3)
df['后移一天'] = df['接单时间'] + pd.DateOffset(days = 1)
df['后移二月'] = df['接单时间'] + pd.DateOffset(months = 2)
df.head(3)
```

返回的值如下：

```
      接单时间              入库日期       后移一天       后移二月
0 2020 - 04 - 28 2020 - 05 - 26 13:54:24 2020 - 04 - 29 2020 - 06 - 28
1 2020 - 05 - 31 2020 - 06 - 03 06:49:22 2020 - 06 - 01 2020 - 07 - 31
2 2020 - 06 - 30 2020 - 07 - 03 00:03:21 2020 - 07 - 01 2020 - 08 - 30
```

2. 偏移

pd.offsets 后面可接的方法较为繁多。简单应用举例,代码如下:

```
df = pd.read_excel(
    r'D:\数据源\B 文件\订单表.xlsx',
   usecols = ['入库日期'],nrows = 3)
df['后移一周'] = df['入库日期'] + pd.offsets.Week()
df['后移三天'] = df['入库日期'] + pd.offsets.Day(3)
df['后移五时'] = df['入库日期'] + pd.offsets.Hour(5)
df['后移一刻'] = df['入库日期'] + pd.offsets.Minute(15)
df
```

返回的值如图 5-7 所示。

	入库日期	后移一周	后移三天	后移五时	后移一刻
0	2020-05-26 13:54:24	2020-06-02 13:54:24	2020-05-29 13:54:24	2020-05-26 18:54:24	2020-05-26 14:09:24
1	2020-06-03 06:49:22	2020-06-10 06:49:22	2020-06-06 06:49:22	2020-06-03 11:49:22	2020-06-03 07:04:22
2	2020-07-03 00:03:21	2020-07-10 00:03:21	2020-07-06 00:03:21	2020-07-03 05:03:21	2020-07-03 00:18:21

图 5-7 返回的值

5.2.5 频率转换

1. 降采样

降采样(resample)用于将高频率数据聚合到低频率,采样的前提是 index 必须为时间序列。在降采样中,目标频率必须是源频率的子时期(subperiod)。以下创建的 DataFrame 用于降采样及升采样,代码如下:

```
#ch05d014.ipynb
import pandas as pd
df = pd.DataFrame(
   data = {"Num":range(1,7)},
   index = pd.date_range('2022/12/9',periods = 6, freq = '2d'))
df
```

以上数据拥有固定的频率。在实际降采样过程中,待聚合的数据可以具有不固定的频率,Pandas 会依据降采样频率自动定义聚合面元的边界,返回的值如下:

```
            Num
2022-12-09    1
2022-12-11    2
2022-12-13    3
2022-12-15    4
2022-12-17    5
2022-12-19    6
```

重新采样的频率为 W,类似于以 W(周)为单位的 group by 对象,可对其聚合运算,代码如下:

```
df.resample('w').sum()
```

2022-12-11对应的周范围为2022-12-5到2020-12-11，2020-12-18对应的周范围为2020-12-12到2020-12-18，2022-12-25对应的周范围为2022-12-19到2022-12-25，返回的值如下：

```
            Num
2022-12-11    3
2022-12-18   12
2022-12-25    6
```

Pandas中常见的聚合函数与方法有count()、sum()、mean()、max()/min()、cummax()/cummin()、idxmax()/argmax()等。当DataFrame或Series中存在时间索引列时，可运用resample()方法对其进行聚合运算，其原理类似于DataFrame或Series对象后接groupby()。当涉及多种聚合运算时，同样可运用agg()方法。应用举例，代码如下：

```
df = pd.read_excel(
   r'D:\数据源\B文件\订单表.xlsx',
   usecols=['订单数','接单时间'],
   index_col='接单时间'
)
df.resample('Q').agg(['count','sum','mean','max','min'])
```

更多有关agg()方法的应用见后续groupby()章节，返回的值如下：

```
            订单数
          count  sum      mean   max  min
接单时间
2020-06-30     3    6  2.000000   2.0  2.0
2020-09-30     7   28  4.000000   9.0  1.0
2020-12-31     3   27  9.000000  11.0  7.0
2021-03-31     0    0       NaN   NaN  NaN
2021-06-30     2    5  2.500000   3.0  2.0
2021-09-30    12  117  9.750000  17.0  2.0
2021-12-31     3   22  7.333333   9.0  4.0
```

当相关日期不是时间索引列时，可通过resample()中的on参数进行指定，代码如下：

```
df = pd.read_excel(
   r'D:\数据源\B文件\订单表.xlsx',
   usecols=['订单数','接单时间'],)
df.resample('4M',on='接单时间').agg(['count','sum','mean'])
```

返回的值如下：

```
            订单数
          count  sum      mean
接单时间
2020-04-30     1    2  2.000000
2020-08-31     5   11  2.200000
2020-12-31     7   48  6.857143
2021-04-30     0    0       NaN
```

```
2021-08-31    13  111  4.538462
2021-12-31     4   33  4.250000
```

2. 升采样

升采样(asfreq)用于将低频率数据转换为高频率数据；在升采样中,目标频率必须是源频率的超时期(superperiod)。

利用 asfreq 对 period 对象进行频率转换,代码如下：

```
a = pd.Period(2021,freq='A-Feb')
a.asfreq('M',how='start')                    #将转换频率为 2019-03
```

运行代码,输出的结果如下：

```
Period('2020-03', 'M')
```

对比 how='start'与 how='end'运行结果的差别,代码如下：

```
a.asfreq('M',how='end')                      #将频率转换为 2020-02
```

运行代码,输出的结果如下：

```
Period('2021-02', 'M')
```

利用 asfreq,对同一频率的不同切割点进行切换,代码如下：

```
a = pd.Period(2021,freq='A-Feb')
a.asfreq('A-JUN')
```

输出的结果如下：

```
Period('2021', 'A-JUN')
```

继续举例,代码如下：

```
#ch05d015.ipynb
import pandas as pd
df = pd.read_excel(
   r'D:\数据源\B文件\订单表.xlsx',
   usecols=['接单时间','产品','订单数'],
   index_col='接单时间'
).head(10)

df.resample('Q').last().resample('M').asfreq()
```

以上代码最后一行采用的是链式写法,其中的 last()方法是时间索引数据中的最后时段,返回的值如下：

```
接单时间
2020-06-30 苹果醋   2.0
2020-07-31   NaN   NaN
2020-08-31   NaN   NaN
2020-09-30 异形件   9.0
```

对比原始数据后会发现：2020-6-30 的订单数 2 所采用的是 2020/6/30 的数据；2020-9-30 的订单数 9 所采用的是 2020/9/27 的数据。如果换成 first()，则代表时间索引数据中的初始时段，代码如下：

```
df.resample('Q').first().resample('M').asfreq()
```

返回的值如下：

```
              产品  订单数
接单时间
2020-06-30  蛋糕纸  2.0
2020-07-31  NaN  NaN
2020-08-31  NaN  NaN
2020-09-30  钢化膜  1.0
```

对比原始数据后会发现：2020-6-30 的蛋糕纸产品的订单数 2 所采用的是 2020/4/28 的数据；2020-9-30 的钢化膜产品的订单数 1 所采用的是 2020/7/28 的数据。