第6章 日期函数的生成和转化

不管在哪个领域中，时间序列（time series）数据都是一种重要的结构化数据形式。在多个时间点观察或测量到的任何事物都可以形成一段时间序列。很多时间序列是固定频率的，也就是说，数据点是根据某种规律定期出现的。时间序列也可以是不定期的。Pandas是在金融建模的背景下开发的，所以它包含了一套用于处理日期、时间和时间索引数据的工具。本章主要从以下几部分展开：

一、日期和时间数据类型及工具

- datatime模块
- calendar模块
- 字符串与datetime相互转换 二、时间序列基础
- 生成日期范围
- 索引 三、时区处理 四、时期period及其算术运算 五、日期的频率以及移动
- 频率和偏移量
- 时移shift 六、重采样及频率转换
- resample方法
- 降采样
- 升采样和插值 七、移动窗口函数 八、指数加权函数

[69]
```
from datetime import datetime
from datetime import timedelta
from datetime import date
from datetime import time
import calendar
```

[3]
```
import pandas as pd
import numpy as np
```

一、日期和时间数据类型及工具

Python标准库包含用于日期（date）和时间（time）数据的数据类型，而且还有日历方面的功能。 datetime.datetime是用得最多的数据类型。我们主要介绍datetime模块、calendar模块以及字符串与datetime的转化。

datetime 模块

datetime 模块提供了可以通过多种方式操作日期和时间的类

```
datetime中的数据类型
date:以公历形式存储日历日期
time：以时间存储为时，分，秒，毫秒
datetime：存储日期和时间
timedelta：表示两个datetime之间的差（日，秒，毫秒）
```

date

```
class datetime.date(year, month, day)
date.year年
date.month月
date.day日
date.replace(year[, month[, day]])生成并返回一个新的日期对象，原日期对象不变
date.timetuple()返回日期对应的time.struct_time对象
date.toordinal()返回日期是是自 0001-01-01 开始的第多少天
date.weekday()返回日期是星期几，[0, 6]，0表示星期一
date.isoweekday()返回日期是星期几，[1, 7]，1表示星期一
date.isocalendar()返回一个元组，格式为：(year, weekday, isoweekday)
date.isoformat()返回‘YYYY-MM-DD’格式的日期字符串
date.strftime(format)返回指定格式的日期字符串
```

[4]
```
print("date对象所能表示的最大日期",date.max)
print("date对象所能表示的最小日期",date.min)
d = date.today()
print("今天是：",d)
print("生成一个新的日期对象，更改年：",d.replace(2016))
print("生成一个新的日期对象，更改年月：",d.replace(2016, 3))
print("生成一个新的日期对象，更改年月日：",d.replace(2016, 3, 2))
print("日期对应的time.struct_time对象：\n",d.timetuple())
print("自 0001-01-01 开始的第",d.toordinal(),"天")
print("(0表示星期一)星期",d.weekday())
print("(1表示星期一)星期",d.isoweekday())
print("格式为：(year, weekday, isoweekday)的元组：",d.isocalendar())
print("‘YYYY-MM-DD’格式的日期字符串：",d.isoformat())
print("date对象所能表示的最大日期",d.ctime())
print("指定格式的日期字符串：",d.strftime('%Y/%m/%d'))
```

```
date对象所能表示的最大日期 9999-12-31
date对象所能表示的最小日期 0001-01-01
今天是： 2019-08-20
```

```
生成一个新的日期对象，更改年： 2016-08-20
生成一个新的日期对象，更改年月： 2016-03-20
生成一个新的日期对象，更改年月日： 2016-03-02
日期对应的time.struct_time对象：
 time.struct_time(tm_year=2019, tm_mon=8, tm_mday=20, tm_hour=0,
tm_min=0, tm_sec=0, tm_wday=1, tm_yday=232, tm_isdst=-1)
自 0001-01-01 开始的第 737291 天
(0表示星期一)星期 1
(1表示星期一)星期 2
格式为：(year, weekday, isoweekday)的元组： (2019, 34, 2)
‘YYYY-MM-DD’格式的日期字符串： 2019-08-20
date对象所能表示的最大日期 Tue Aug 20 00:00:00 2019
指定格式的日期字符串： 2019/08/20
```

time

```
class datetime.time(hour,minute,second,microsecond,tzinfo)
time.max    time类所能表示的最大时间：time(23, 59, 59, 999999)
time.min    time类所能表示的最小时间：time(0, 0, 0, 0)
time.resolution   时间的最小单位，即两个不同时间的最小差值：1微秒
t.replace(hour,minute,second,microsecond,tzinfo)     生成并返回一个新的时间
对象，原时间对象不变
t.isoformat()    返回一个‘HH:MM:SS.%f’格式的时间字符串
t.strftime()     返回指定格式的时间字符串，与time模块的strftime(format,
struct_time)功能相同
```

[5]
```
print("time类所能表示的最大时间:",time.max)
print("time类所能表示的最小时间:",time.min)
print("时间的最小单位:",time.resolution)
t = time(20, 5, 40, 8888)
print("(时，分，秒，微秒，
tzinfo)：",t.hour,t.minute,t.second,t.microsecond,t.tzinfo)
print("生成新的时间对象，代替时",t.replace(21))
print("HH:MM:SS.%f’格式的时间字符串",t.isoformat())
print("返回指定格式的时间字符串",t.strftime('%H%M%S'))
```

```
time类所能表示的最大时间： 23:59:59.999999
time类所能表示的最小时间： 00:00:00
时间的最小单位： 0:00:00.000001
(时，分，秒，微秒，tzinfo)： 20 5 40 8888 None
生成新的时间对象，代替时 21:05:40.008888
HH:MM:SS.%f’格式的时间字符串 20:05:40.008888
返回指定格式的时间字符串 200540
```

datetime

```
class datetime.datetime(year, month, day, hour=0, minute=0, second=0,
microsecond=0, tzinfo=None)
datetime.today()返回一个表示当前本期日期时间的datetime对象
datetime.now([tz])返回指定时区日期时间的datetime对象，如果不指定tz参数则结果同上
datetime.utcnow()返回当前utc日期时间的datetime对象
datetime.strftime(date_str, format)将datetime对象转换为时间字符串
datetime.strptime(date_str, format)将时间字符串转换为datetime对象
```

[6]
```
now = datetime.now()
print("属性:
year,month,day,hour,minute,second,microsecond,tzinfo.")
print("
",now.year,now.month,now.day,now.hour,now.minute,now.second,now.m
icrosecond,now.tzinfo)
print(datetime.today())
print(datetime.utcnow())
print(now.isoformat())#返回一个 ISO 8601 格式的字符串， 'YYYY-MM-
DD'。
print(now.strftime('%Y-%m-%d'))#接收一个时间元组，并返回以可读字符串表示
的当地时间，格式由参数决定。
print(datetime.strptime('2019/08/19 20:49', '%Y/%m/%d %H:%M'))#将
字符串转为datetime
```

```
属性:year,month,day,hour,minute,second,microsecond,tzinfo.
      2019 8 20 12 43 7 246369 None
2019-08-20 12:43:07.246369
2019-08-20 04:43:07.246369
2019-08-20T12:43:07.246369
2019-08-20
2019-08-19 20:49:00
```

timedelta

如果有人问你昨天是几号，这个很容易就回答出来了。但是如果问你200天前是几号，就不是那么容易答出来。而在Python中datetime模块中的timedelta就可以很轻松给出答案。
timedelta 对象表示两个 date 或者 time 的时间间隔。
class datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, hours=0, weeks=0)
datetime.datetime.now() 返回当前本地时间（datetime.datetime对象实例）

datetime.datetime.fromtimestamp(timestamp)　返回指定时间戳对应的时间（datetime.datetime对象实例）
datetime.timedelta()　　返回一个时间间隔对象，可以直接与datetime.datetime对象做加减操作

```
[7]  print("一年包含的总秒数:",timedelta(365).total_seconds())
     dt = datetime.now()
     print("3天后:",dt + timedelta(3))
     print("3天前:",dt + timedelta(-3))
     print("3小时后:",dt + timedelta(hours=3))
     print("3小时前:",dt + timedelta(hours=-3))
     print("3小时30秒后:",dt + timedelta(hours=3, seconds=30))
```

```
一年包含的总秒数: 31536000.0
3天后: 2019-08-23 12:43:07.327154
3天前: 2019-08-17 12:43:07.327154
3小时后: 2019-08-20 15:43:07.327154
3小时前: 2019-08-20 09:43:07.327154
3小时30秒后: 2019-08-20 15:43:37.327154
```

calendar模块

calendar是与日历相关的模块，calendar模块文件里定义了很多类型，主要有Calendar，TextCalendar以及HTMLCalendar类型。其中，Calendar是TextCalendar与HTMLCalendar的基类。该模块文件还对外提供了很多方法，例如：calendar，month，prcal，prmonth之类的方法。

calendar模块方法
calendar.setfirstweekday(firstweekday)指定一周的第一天，0是星期一，…，6为星期日
calendar.firstweekday()返回一周的第一天，0是星期一，…，6为星期日
calendar.isleap(year)：判断指定是否是闰年，闰年为True，平年为False
calendar.leapdays(y1, y2)：返回y1与y2年份之间的闰年数量，y1与y2皆为年份。包括起始年，不包括结束年：
calendar.weekday(year, month, day)：获取指定日期为星期几
calendar.weekheader(n)：返回包含星期的英文缩写，n表示英文缩写所占的宽度
calendar.monthrange(year, month)：返回一个由一个月第一个天的星期与当前月的天数组成的元组
calendar.monthcalendar(year, month)：返回一个月中天数列表(不是当前月份的天数为0)，按周划分，为一个二维数组。包括月份开始那周的所有日期和月份结束那周的所有日期
calendar.prmonth(year,month, w=0, l=0)：打印一个月的日历
calendar.month(year,month, w=0, l=0)：返回一个月的日历的多行文本字符串。
calendar.prcal(year, w=0, l=0, c=6, m=3)：打印一年的日历，
calendar.calendar(year, w=2, l=1, c=6, m=3)：以多行字符串形式返回一年的日历
参数说明：
w:每个单元格宽度，默认0，内部已做处理，最小宽度为2

l:每列换l行，默认为0，内部已做处理，至少换行1行
c:表示月与月之间的间隔宽度，默认为6，内部已做处理，最小宽度为2
m:表示将12个月分为m列

[76]
```
calendar.setfirstweekday(firstweekday=6)
print(calendar.firstweekday())
print(calendar.isleap(2019))
print("2008年到2019年之间闰年数量：",calendar.leapdays(2008, 2019))
print("2019-8-8是星期：",calendar.weekday(2019, 8, 8))# 2019-08-08
正是星期四，千万别忘记3代表的是星期四
print("包含星期的英文缩写:",calendar.weekheader(4))
print("2019年8月天数列表:",calendar.monthcalendar(2019, 8))
print("2019年8月的日历:",calendar.prmonth(2019,8))
print("2019年8月的日历的多行文本字符串:\n",calendar.month(2019, 8))
print("2019年的日历:",calendar.prcal(2019, m=4))
print("2019年的日历的多行文本字符串:\n",calendar.calendar(2018, m=4))
```

```
6
False
2008年到2019年之间闰年数量： 3
2019-8-8是星期： 3
包含星期的英文缩写: Sun  Mon  Tue  Wed  Thu  Fri  Sat
2019年8月天数列表: [[0, 0, 0, 0, 1, 2, 3], [4, 5, 6, 7, 8, 9, 10], [11,
12, 13, 14, 15, 16, 17], [18, 19, 20, 21, 22, 23, 24], [25, 26, 27, 28,
29, 30, 31]]
    August 2019
Su Mo Tu We Th Fr Sa
                1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
2019年8月的日历: None
2019年8月的日历的多行文本字符串:
     August 2019
Su Mo Tu We Th Fr Sa
                1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

                                  2019

      January                   February                  March       ...
April
Su Mo Tu We Th Fr Sa      Su Mo Tu We Th Fr Sa      Su Mo Tu We Th Fr ...
Su Mo Tu We Th Fr Sa
       1  2  3  4  5                      1  2                        1 ...
1  2  3  4  5  6
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       3  4  5  6  7  8 ...
```

```
7  8  9 10 11 12 13
13 14 15 16 17 18 19      10 11 12 13 14 15 16      10 11 12 13 14 15 ...
14 15 16 17 18 19 20
20 21 22 23 24 25 26      17 18 19 20 21 22 23      17 18 19 20 21 22 ...
21 22 23 24 25 26 27
27 28 29 30 31            24 25 26 27 28            24 25 26 27 28 29 ...
28 29 30
                                                    31

        May                       June                      July      ...
August
Su Mo Tu We Th Fr Sa      Su Mo Tu We Th Fr Sa      Su Mo Tu We Th Fr ...
Su Mo Tu We Th Fr Sa
          1  2  3  4                         1          1  2  3  4  5 ...
1  2  3
 5  6  7  8  9 10 11       2  3  4  5  6  7  8       7  8  9 10 11 12 ...
4  5  6  7  8  9 10
12 13 14 15 16 17 18       9 10 11 12 13 14 15      14 15 16 17 18 19 ...
11 12 13 14 15 16 17
19 20 21 22 23 24 25      16 17 18 19 20 21 22      21 22 23 24 25 26 ...
18 19 20 21 22 23 24
26 27 28 29 30 31         23 24 25 26 27 28 29      28 29 30 31       ...
25 26 27 28 29 30 31
                          30

     September                 October                   November    ...
December
Su Mo Tu We Th Fr Sa      Su Mo Tu We Th Fr Sa      Su Mo Tu We Th Fr ...
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7              1  2  3  4  5                      1 ...
1  2  3  4  5  6  7
 8  9 10 11 12 13 14       6  7  8  9 10 11 12       3  4  5  6  7  8 ...
8  9 10 11 12 13 14
15 16 17 18 19 20 21      13 14 15 16 17 18 19      10 11 12 13 14 15 ...
15 16 17 18 19 20 21
22 23 24 25 26 27 28      20 21 22 23 24 25 26      17 18 19 20 21 22 ...
22 23 24 25 26 27 28
29 30                     27 28 29 30 31            24 25 26 27 28 29 ...
29 30 31


2019年的日历: None
2019年的日历的多行文本字符串:
                                        2018

      January                   February                  March       ...
April
Su Mo Tu We Th Fr Sa      Su Mo Tu We Th Fr Sa      Su Mo Tu We Th Fr ...
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6                  1  2  3                   1  2 ...
1  2  3  4  5  6  7
 7  8  9 10 11 12 13       4  5  6  7  8  9 10       4  5  6  7  8  9 ...
8  9 10 11 12 13 14
```

```
14 15 16 17 18 19 20      11 12 13 14 15 16 17      11 12 13 14 15 16 ...
15 16 17 18 19 20 21
21 22 23 24 25 26 27      18 19 20 21 22 23 24      18 19 20 21 22 23 ...
22 23 24 25 26 27 28
28 29 30 31               25 26 27 28               25 26 27 28 29 30 ...
29 30

        May                       June                      July      ...
August
Su Mo Tu We Th Fr Sa      Su Mo Tu We Th Fr Sa      Su Mo Tu We Th Fr ...
Su Mo Tu We Th Fr Sa
       1  2  3  4  5                    1  2       1  2  3  4  5  6 ...
1  2  3  4
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       8  9 10 11 12 13 ...
5  6  7  8  9 10 11
13 14 15 16 17 18 19      10 11 12 13 14 15 16      15 16 17 18 19 20 ...
12 13 14 15 16 17 18
20 21 22 23 24 25 26      17 18 19 20 21 22 23      22 23 24 25 26 27 ...
19 20 21 22 23 24 25
27 28 29 30 31            24 25 26 27 28 29 30      29 30 31          ...
26 27 28 29 30 31

     September                 October                   November    ...
December
Su Mo Tu We Th Fr Sa      Su Mo Tu We Th Fr Sa      Su Mo Tu We Th Fr ...
Su Mo Tu We Th Fr Sa
                  1          1  2  3  4  5  6                  1  2 ...
1
 2  3  4  5  6  7  8       7  8  9 10 11 12 13       4  5  6  7  8  9 ...
2  3  4  5  6  7  8
 9 10 11 12 13 14 15      14 15 16 17 18 19 20      11 12 13 14 15 16 ...
9 10 11 12 13 14 15
16 17 18 19 20 21 22      21 22 23 24 25 26 27      18 19 20 21 22 23 ...
16 17 18 19 20 21 22
23 24 25 26 27 28 29      28 29 30 31               25 26 27 28 29 30 ...
23 24 25 26 27 28 29
30                                                                    ...
30 31
```

字符串与datetime的相互转换

datetime格式定义：

%Y:4位数的年
%y:2位数的年

%m:2位数的月[01,12]
%d:2位数的日[01,31]
%H:23小时制[00,23]
%I:12小时制[01,12]
%M:2位数的分[00,59]
%S:秒[00,61]秒60和61用于闰秒
%w:用整数表示的星期几[0(星期日),6]
%U:每年的第几周[00,53]。星期天被认为是每周的第一天，每年的第一个星期天之前的几天是第0周
%W:每年的第几周[00,53]。星期一被认为是每周的第一天，每年的第一个星期一之前的几天是第0周
%z:以+HHMM或-HHMM表示的UTC时间对象，如果时区为naive，则返回空字符串
%F: %Y-%m-%d简写形式，例如2012-04-18
%D:%m%d%y简写形式，例如04/18/12

特定于当年环境的日期格式：

%a：星期几的简写%A星期几的全称
%b：月份的简写%B月份的全称
%c：完整的日期和时间，例如“Tue 01May 201204：20：57PM”
%p：不同环境中的AM或PM
%x：适合于当前环境的日期格式，例如，在美国，“May 1，2019”会产生“05/01/2019”
%X：适合于当前环境的时间格式，例如“04：24：12PM”

利用str或strftime方法（传入一个格式化字符串），datetime对象和pandas的Timestamp对象（稍后就会介绍）可以被格式化为字符串：

[8]
```
stamp=datetime(2019,7,3)
str(stamp),stamp.strftime('%Y-%m-%d')
```

```
('2019-07-03 00:00:00', '2019-07-03')
```

[9]
```
#对于一些常见的日期格式每次要编写格式定义是很麻烦的事情，这种情况下，可以用dateutil这个第三方包中的parser.parse方法：
from dateutil.parser import parse
parse('2019-01-03'),parse('6/12/2011',dayfirst=True)#日期在前，dayfirst=True
```

```
(datetime.datetime(2019, 1, 3, 0, 0), datetime.datetime(2011, 12, 6, 0, 0))
```

Pandas提供Timestamp类型为时间戳。关联的索引结构是DatetimeIndex.
Pandas提供Period类型为时间段。这将编码一个固定频率间隔，该间隔基于numpy.datetime64..关联的索引结构是PeriodIndex.

Pandas提供Timedelta类型为时间三角洲或持续时间。Timedelta是Python本机更有效的替代品datetime.timedelta类型，关联的索引结构是TimedeltaIndex.

这些日期/时间对象中最基本的是Timestamp和DatetimeIndex。虽然可以直接调用这些类对象，但使用pd.to_datetime()函数可以解析多种格式。将一个日期传递给pd.to_datetime()产生一个Timestamp；默认情况下传递一系列日期将产生DatetimeIndex

[10]
```
#在pandas中可以使用to_datetime方法处理成组日期,解析多种不同的日期表现形式
date = pd.to_datetime("4th of July, 2015")
date
```

```
Timestamp('2015-07-04 00:00:00')
```

[11]
```
#此外，我们可以直接对同一个对象执行NumPy风格的矢量化操作：
date + pd.to_timedelta(np.arange(12), 'D')
```

```
DatetimeIndex(['2015-07-04', '2015-07-05', '2015-07-06', '2015-07-07',
               '2015-07-08', '2015-07-09', '2015-07-10', '2015-07-11',
               '2015-07-12', '2015-07-13', '2015-07-14', '2015-07-15'],
              dtype='datetime64[ns]', freq=None)
```

[12]
```
#构造一个具有时间索引数据的Series
index = pd.DatetimeIndex(['2014-07-04', '2014-08-04',
                          '2015-07-04', '2015-08-04'])
data = pd.Series([0, 1, 2, 3], index=index)
data
```

```
2014-07-04    0
2014-08-04    1
2015-07-04    2
2015-08-04    3
dtype: int64
```

[13]
```
dates = pd.to_datetime([datetime(2015, 7, 3), '4th of July,
2015',
                        '2015-Jul-6', '07-07-2015', '20150708'])
dates
```

```
DatetimeIndex(['2015-07-03', '2015-07-04', '2015-07-06', '2015-07-07',
               '2015-07-08'],
              dtype='datetime64[ns]', freq=None)
```

[14]
```
#任何DatetimeIndex可以转换为PeriodIndex:to_period()函数,使用'D'指示每日频率:
dates.to_period('D')
```

```
PeriodIndex(['2015-07-03', '2015-07-04', '2015-07-06', '2015-07-07',
             '2015-07-08'],
            dtype='period[D]', freq='D')
```

[15]
```
#当从另一个日期减去日期时,生成一个TimedeltaIndex
dates - dates[0]
```

```
TimedeltaIndex(['0 days', '1 days', '3 days', '4 days', '5 days'],
dtype='timedelta64[ns]', freq=None)
```

二、时间序列基础

当数据是以时间戳作为索引的话，可以将其理解为一个时间序列数据。我们主要介绍生成日期范围与时间序列的索引。

生成日期范围

```
pd.date_range(start,end,periods)
```
为了更方便地创建日期序列，Pandas为此提供了一些功能：pd.date_range()时间戳，pd.period_range()一段时间内，以及pd.timedelta_range()。
我们都知道，Python的range()和NumPy的np.arange()将起始点、端点和可选步长转换为序列。同样，pd.date_range()接受开始日期、结束日期和可选频率代码，以创建定期日期序列。默认情况下，频率为某一天。

[16]
```
long_ts =
pd.Series(np.random.randn(1000),index=pd.date_range('1/1/2000',periods=1000))
long_ts.head()
```

```
2000-01-01    1.315127
2000-01-02    2.128896
2000-01-03    0.927003
```

```
2000-01-04   -0.880661
2000-01-05   -1.547260
Freq: D, dtype: float64
```

索引

[17]
```
#索引,两种方式结果一样
print(long_ts['2002-9-24':])
print(long_ts[datetime(2002,9,24):])
```

```
2002-09-24   -0.627696
2002-09-25   -1.633077
2002-09-26    1.457194
Freq: D, dtype: float64
2002-09-24   -0.627696
2002-09-25   -1.633077
2002-09-26    1.457194
Freq: D, dtype: float64
```

[18]
```
long_ts['2002-09']#选取2001-05的数据
```

```
2002-09-01   -0.435344
2002-09-02   -0.448500
2002-09-03    0.908174
2002-09-04    1.399495
2002-09-05   -1.199545
2002-09-06    0.980850
2002-09-07   -2.008823
2002-09-08   -0.527086
2002-09-09    0.788963
2002-09-10   -1.606466
2002-09-11   -0.083876
2002-09-12   -1.248117
2002-09-13    0.288755
2002-09-14   -0.293276
2002-09-15   -0.510468
2002-09-16    1.001119
2002-09-17    1.386802
2002-09-18   -0.667894
2002-09-19   -1.245537
2002-09-20    0.024803
2002-09-21    0.307759
2002-09-22   -1.018092
2002-09-23   -1.237862
2002-09-24   -0.627696
```

```
2002-09-25   -1.633077
2002-09-26    1.457194
Freq: D, dtype: float64
```

带有重复索引的时间序列 即一个时间出现多条记录

[19]
```
dates =
pd.DatetimeIndex(['1/1/2000','1/2/2000','1/2/2000','1/2/2000','1/
3/2000'])
dup_ts = pd.Series(np.arange(5),index=dates)
dup_ts
```

```
2000-01-01    0
2000-01-02    1
2000-01-02    2
2000-01-02    3
2000-01-03    4
dtype: int32
```

[20]
```
#检查索引是否唯一
dup_ts.index.is_unique
```

```
False
```

[21]
```
#通过groupby检查
dup_ts.groupby(level=0).count()
```

```
2000-01-01    1
2000-01-02    3
2000-01-03    1
dtype: int64
```

三、时区处理

时间序列处理工作中最麻烦的是对时区的处理。尤其是夏令时（DST）转变，这是一种最常见的麻烦事，就这一点来说，许多人都选择以协调世界时（UTC）来处理时间序列。时区是以UTC偏移量的形式表示的。例如，夏令时期间，纽约比UTC慢4小时，而在全年其他时间则比UTC慢5小时。

在Python中，时区信息来自第三方库pytz，它使Python可以使用Olson数据库。这对历史数据非常重要，这是因为由于各地政府的各种突发奇想，夏令时转变日期（甚至UTC偏移量）已经发生过多次改变了。就拿美国来说，DST转变时间自1900年以来就改变过多次。

[22]
```
#由于pandas包装了pytz的功能，因此你可以不用记忆其API，只要记得时区的名称即可。时区名可以在文档中找到，也可以通过交互的方式查看：
import pytz
pytz.common_timezones[-5:]
```

```
['US/Eastern', 'US/Hawaii', 'US/Mountain', 'US/Pacific', 'UTC']
```

[23]
```
#要从pytz中获取时区对象，使用pytz.timezone即可：
tz=pytz.timezone("US/Eastern")
tz
```

```
<DstTzInfo 'US/Eastern' LMT-1 day, 19:04:00 STD>
```

本地化和转换

默认情况下，pandas中的时间序列是单纯的（naive）时区

```
tz_localize()时区定位/本地化
tz_convert()时区转换
```

[24]
```
rng=pd.date_range('3/9/2012 9:30',periods=6,freq='D')
ts=pd.Series(np.random.randn(len(rng)),index=rng)
print(ts.index.tz)
```

```
None
```

[25]
```
#在生成日期范围的时候还可以加上一个时区集
pd.date_range('3/9/2012 9:30',periods=10,freq='D',tz='UTC')
```

```
DatetimeIndex(['2012-03-09 09:30:00+00:00', '2012-03-10 09:30:00+00:00',
               '2012-03-11 09:30:00+00:00', '2012-03-12 09:30:00+00:00',
```

```
               '2012-03-13 09:30:00+00:00', '2012-03-14 09:30:00+00:00',
               '2012-03-15 09:30:00+00:00', '2012-03-16 09:30:00+00:00',
               '2012-03-17 09:30:00+00:00', '2012-03-18
09:30:00+00:00'],
              dtype='datetime64[ns, UTC]', freq='D')
```

[26]
```
#从单纯到本地化的转换是通过tz_localize方法处理的：
ts_utc=ts.tz_localize('UTC')
ts_utc
```

```
2012-03-09 09:30:00+00:00    0.525350
2012-03-10 09:30:00+00:00    0.498384
2012-03-11 09:30:00+00:00   -1.603312
2012-03-12 09:30:00+00:00    0.368390
2012-03-13 09:30:00+00:00    1.245520
2012-03-14 09:30:00+00:00    2.336125
Freq: D, dtype: float64
```

[27]
```
ts_utc.index
```

```
DatetimeIndex(['2012-03-09 09:30:00+00:00', '2012-03-10 09:30:00+00:00',
               '2012-03-11 09:30:00+00:00', '2012-03-12 09:30:00+00:00',
               '2012-03-13 09:30:00+00:00', '2012-03-14
09:30:00+00:00'],
              dtype='datetime64[ns, UTC]', freq='D')
```

[28]
```
#一旦时间序列被本地化到某个特定时区，就可以用tz_convert将其转换到别的时区了：
ts_utc.tz_convert('US/Eastern')
```

```
2012-03-09 04:30:00-05:00    0.525350
2012-03-10 04:30:00-05:00    0.498384
2012-03-11 05:30:00-04:00   -1.603312
2012-03-12 05:30:00-04:00    0.368390
2012-03-13 05:30:00-04:00    1.245520
2012-03-14 05:30:00-04:00    2.336125
Freq: D, dtype: float64
```

[29]
```
#对于上面这种时间序列（它跨越了美国东部时区的夏令时转变期），我们可以将其本地化
到EST，然后转换为UTC或柏林时间：
ts_eastern=ts.tz_localize('US/Eastern')
ts_eastern.tz_convert('UTC')
```

```
2012-03-09 14:30:00+00:00    0.525350
2012-03-10 14:30:00+00:00    0.498384
2012-03-11 13:30:00+00:00   -1.603312
2012-03-12 13:30:00+00:00    0.368390
2012-03-13 13:30:00+00:00    1.245520
2012-03-14 13:30:00+00:00    2.336125
Freq: D, dtype: float64
```

[30]
```
#tz_localize和tz_convert也是DatetimeIndex的实例方法
ts.index.tz_localize('Asia/Shanghai')
```

```
DatetimeIndex(['2012-03-09 09:30:00+08:00', '2012-03-10 09:30:00+08:00',
               '2012-03-11 09:30:00+08:00', '2012-03-12 09:30:00+08:00',
               '2012-03-13 09:30:00+08:00', '2012-03-14
09:30:00+08:00'],
              dtype='datetime64[ns, Asia/Shanghai]', freq='D')
```

四、时期period及其算术运算

时期（period）表示的是时间区间，比如数日、数月、数季、数年等。 Period类所表示的就是这种数据类型，其构造函数需要用到一个字符串或整数和频率。

[31]
```
p=pd.Period(2007,freq='A-DEC')#这个Period对象表示的是从2007年1月1日到2007年12月31日之间的整段时间。
p
```

```
Period('2007', 'A-DEC')
```

[32]
```
#对Period对象加上或减去一个整数即可达到根据其频率进行位移的效果：
p+5,p-2
```

```
(Period('2012', 'A-DEC'), Period('2005', 'A-DEC'))
```

[33]
```
#period_range函数可用于创建规则的时期范围：
rng=pd.period_range('1/1/2000','6/30/2000',freq='M')
rng
```

```
PeriodIndex(['2000-01', '2000-02', '2000-03', '2000-04', '2000-05',
'2000-06'], dtype='period[M]', freq='M')
```

[34]
```
#PeriodIndex类保存了一组Period，它可以在任何pandas数据结构中被用作轴索引：
pd.Series(np.random.randn(6),index=rng)
```

```
2000-01   -0.165660
2000-02   -0.146480
2000-03    0.287034
2000-04    0.767443
2000-05   -0.084857
2000-06    1.513086
Freq: M, dtype: float64
```

[35]
```
#PeriodIndex类的构造函数还允许直接使用一组字符串
values=['200103','200202','200301']
index=pd.PeriodIndex(values,freq='Q-DEC')
index
```

```
PeriodIndex(['2003Q1', '2002Q1', '2001Q1'], dtype='period[Q-DEC]',
freq='Q-DEC')
```

五、日期的频率以及移动

pandas中的时间序列一般被认为是不规则的，也就是说，它们没有固定的频率。对于大部分应用程序而言，这是无所谓的。但是，它常常需要以某种相对固定的频率进行分析，比如每日、每月、每15分钟等（这样自然会在时间序列中引入缺失值）。幸运的是，pandas有一整套标准时间序列频率以及用于重采样、频率推断、生成固定频率日期范围的工具。我们主要介绍频率和偏移量以及时移。

频率和偏移量

Pandas时间序列工具的基础是频率或日期偏移的概念。
pandas中的频率是由一个基础频率（base frequency）和一个乘数组成的。基础频率通常以一个字符串别名表示，比如“M”表示每月，“H”表示每小时。对于每个基础频率，都有一个被称为日期偏移量（date offset）的对象与之对应。

有些频率所描述的时间点并不是均匀分隔的。例如，“M”（日历月末）和“BM”（每月最后一个工作日）就取决于每月的天数，对于后者，还要考虑月末是不是周末。由于没有更好的术语，我将这些称为锚点偏移量（anchored offset）。
时间序列的基础频率：

D	日历日
W	每周
M	月底
Q	季度末
A	年底
H	小时数
T	分分钟
S	秒数
L	米利斯秒
U	微秒
N	纳秒
B	营业日
BM	营业月结束
BQ	营业季度末
BA	营业年度结束
BH	营业时间

每个月、每个季度和每年的频率都是在指定期间结束时标记的。通过添加S它们的后缀将在开头标记：

MS	月开始
QS	季度启动
AS	年初
BMS	营业月开始
BQS	营业季度开始
BAS	营业年度开始

此外，还可以通过添加三个字母的月份代码作为后缀来更改用于标记任何季度或年度代码的月份：
Q-JAN，BQ-FEB，QS-MAR，BQS-APR等

同样，可以通过添加三个字母的工作日代码来修改每周频率的分割点：
W-SUN，W-MON，W-TUE，W-WED等

最重要的是，它们可以与数字相结合来指定其他频率。

[36]
```
#例如，在频率为2小时30分钟的情况下，我们可以将该小时(H)和分钟(T)守则如下：
pd.timedelta_range(0, periods=9, freq="2H30T")
```

```
TimedeltaIndex(['00:00:00', '02:30:00', '05:00:00', '07:30:00',
'10:00:00',
                '12:30:00', '15:00:00', '17:30:00', '20:00:00'],
               dtype='timedelta64[ns]', freq='150T')
```

[37]
```
#WOM（Week Of Month）日期是一种非常实用的频率类
#例如2019年每月第三个星期五
```

```
pd.date_range(start='1/1/2019',periods=12,freq='WOM-3FRI')
```

```
DatetimeIndex(['2019-01-18', '2019-02-15', '2019-03-15', '2019-04-19',
               '2019-05-17', '2019-06-21', '2019-07-19', '2019-08-16',
               '2019-09-20', '2019-10-18', '2019-11-15', '2019-12-20'],
              dtype='datetime64[ns]', freq='WOM-3FRI')
```

时移 Shift

常见的特定于时间序列的操作是数据在时间上的转移。
移动（shifting）指的是沿着时间轴将数据前移或后移。
pandas有两种密切相关的计算方法：shift()和tshift()。Series和Dataframe都有一个shift()和tshift()方法。
简而言之，他们之间的区别是shift() 移动数据，tshift()移动索引..在这两种情况下，移位都是以频率的倍数指定的。
df.shift(periods=1, freq=None, axis=0)
period：表示移动的幅度
freq： 可选参数，默认值为None，只适用于时间序列，如果这个参数存在，那么会按照参数值移动时间索引，而数据值没有发生变化。
axis： 轴向。

[38]
```
#shift
ts =
pd.Series(np.random.randn(4),index=pd.date_range('1/1/2000',perio
ds=4,freq='m'))
print(ts,'\n')
print("正向移动两位数据:\n",ts.shift(2),'\n')
print("反向移动两位数据:\n",ts.shift(-2),'\n')
#shift通常用于计算一个时间序列或多个时间序列（如DataFrame的列）中的百分比变化。
print("相比上月百分比的变化:\n",ts/ts.shift(1),'\n')
#由于单纯的移位操作不会修改索引，所以部分数据会被丢弃。如果频率已知，则可以将其传给shift以便实现对时间戳进行位移而不是对数据进行简单位移
print("输入频率实现对时间戳位移:\n",ts.shift(2,freq='M'),'\n')
```

```
2000-01-31   -1.365959
2000-02-29    1.445598
2000-03-31   -0.953769
2000-04-30    1.323299
Freq: M, dtype: float64

正向移动两位数据:
 2000-01-31         NaN
2000-02-29         NaN
```

```
2000-03-31   -1.365959
2000-04-30    1.445598
Freq: M, dtype: float64

反向移动两位数据:
 2000-01-31   -0.953769
2000-02-29    1.323299
2000-03-31         NaN
2000-04-30         NaN
Freq: M, dtype: float64

相比上月百分比的变化:
 2000-01-31         NaN
2000-02-29   -1.058303
2000-03-31   -0.659775
2000-04-30   -1.387443
Freq: M, dtype: float64

输入频率实现对时间戳位移:
 2000-03-31   -1.365959
2000-04-30    1.445598
2000-05-31   -0.953769
2000-06-30    1.323299
Freq: M, dtype: float64
```

[39]
```
#pandas的日期偏移量还可以用在datetime或Timesstamp上
from pandas.tseries.offsets import Day,MonthEnd
now = datetime(2019,8,19)
now+3*Day(),now+MonthEnd(),now+MonthEnd(2)#如果加的是锚点偏移量（比如MonthEnd），第一次增量会将原日期滚动到符合频率规则的下个日期
```

```
(Timestamp('2019-08-22 00:00:00'),
 Timestamp('2019-08-31 00:00:00'),
 Timestamp('2019-09-30 00:00:00'))
```

[40]
```
#通过锚点偏移量的rollforward和rollback方法，可显式将日期向前或向后滚动
offset=MonthEnd()
now,offset.rollforward(now),offset.rollback(now)
```

```
(datetime.datetime(2019, 8, 19, 0, 0),
 Timestamp('2019-08-31 00:00:00'),
 Timestamp('2019-07-31 00:00:00'))
```

六、重采样及频率转换

涌采样（resampling）指的是将时间序列从一个频率转换到另一个频率的处理过程。将高频率数据聚合到低频率称为降采样（downsampling），而将低频率数据转换到高频率则称为升采样（upsampling）。本部分我们主要介绍降采样和升采样。

resample方法

pandas对象都有一个resample方法，它是重采样以及各种频率转换工作的主力函数。
resample(freq,how='mean',axis=0,fill_method=None,closed='right',label='right',loffset=None,limit=None,convention=None)
参数说明
freq　　表示重采样频率的字符串或DateOffset，例如“M’、‘5min'或Second（15）
how='mean'　　用于产生聚合值的函数名或数组函数。默认为‘mean'。其他常用的值有：‘first'、last'、‘median'、‘ohlc'、‘max'、'min'
axis=0　　重采样的轴，默认为axis=0
fill_method=None　　升采样时如何插值，比如“ffill或“bfill'。默认不插值
closed='right'　　在降采样中，各时间段的哪一端是闭合（即包含）的，‘right'或left'。默认为right'
label='right'　　在降采样中，如何设置聚合值的标签，right'或left'（面元的右边界或左边界）。
loffset=None　　面元标签的时间校正值，比如-1s'/Second（-1）用于将聚合标签调早1秒
limit=None　　在前向或后向填充时，允许填充的最大时期数kind=None聚合到时期或时间戳，默认聚合到时间序列的索引类型
convention=None　　当重采样时期时，将低频率转换到高频率所采用的约定（“start'或end’）。默认为end'

[41]
```
dates = pd.DatetimeIndex(['1/2/2000','1/5/2000','1/7/2000','1/8/2000','1/10/2000','1/12/2000'])
ts = pd.Series(np.random.randn(6),index=dates)
ts
```

```
2000-01-02    0.235419
2000-01-05    2.024163
2000-01-07   -0.374881
2000-01-08   -1.159212
2000-01-10    0.232465
2000-01-12    0.636269
dtype: float64
```

[42]
```
#利用resample构造完整序列
a=ts.resample('D').asfreq()
a
```

```
2000-01-02    0.235419
2000-01-03         NaN
2000-01-04         NaN
2000-01-05    2.024163
2000-01-06         NaN
2000-01-07   -0.374881
2000-01-08   -1.159212
2000-01-09         NaN
2000-01-10    0.232465
2000-01-11         NaN
2000-01-12    0.636269
Freq: D, dtype: float64
```

金融领域中有一种无所不在的时间序列聚合方式，即计算各面元的四个值：第一个值（open，开盘）、最后一个值（close，收盘）、最大值（high，最高）以及最小值（low，最低）。传入how='ohlc'即可得到一个含有这四种聚合值的DataFrame。整个过程很高效，只需一次扫描即可计算出结果：

[43]
```
#OHLC重采样，统计开盘，最高，最低以及收盘
ts.resample('5min',how='ohlc')
```

```
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning:
how in .resample() is deprecated
the new syntax is .resample(...).ohlc()
```

	open	high	low	close
2000-01-02 00:00:00	0.235419	0.235419	0.235419	0.235419
2000-01-02 00:05:00	NaN	NaN	NaN	NaN
2000-01-02 00:10:00	NaN	NaN	NaN	NaN
2000-01-02 00:15:00	NaN	NaN	NaN	NaN
2000-01-02 00:20:00	NaN	NaN	NaN	NaN
2000-01-02 00:25:00	NaN	NaN	NaN	NaN

	open	high	low	close
2000-01-02 00:30:00	NaN	NaN	NaN	NaN

2000-01-02 00:35:00	NaN	NaN	NaN	NaN
2000-01-02 00:40:00	NaN	NaN	NaN	NaN
2000-01-02 00:45:00	NaN	NaN	NaN	NaN
2000-01-02 00:50:00	NaN	NaN	NaN	NaN
2000-01-02 00:55:00	NaN	NaN	NaN	NaN
2000-01-02 01:00:00	NaN	NaN	NaN	NaN
2000-01-02 01:05:00	NaN	NaN	NaN	NaN
2000-01-02 01:10:00	NaN	NaN	NaN	NaN
2000-01-02 01:15:00	NaN	NaN	NaN	NaN
2000-01-02 01:20:00	NaN	NaN	NaN	NaN
2000-01-02 01:25:00	NaN	NaN	NaN	NaN
2000-01-02 01:30:00	NaN	NaN	NaN	NaN
2000-01-02 01:35:00	NaN	NaN	NaN	NaN
2000-01-02 01:40:00	NaN	NaN	NaN	NaN
2000-01-02 01:45:00	NaN	NaN	NaN	NaN
2000-01-02 01:50:00	NaN	NaN	NaN	NaN
2000-01-02 01:55:00	NaN	NaN	NaN	NaN
2000-01-02 02:00:00	NaN	NaN	NaN	NaN
2000-01-02 02:05:00	NaN	NaN	NaN	NaN
2000-01-02 02:10:00	NaN	NaN	NaN	NaN
2000-01-02 02:15:00	NaN	NaN	NaN	NaN
2000-01-02 02:20:00	NaN	NaN	NaN	NaN
2000-01-02 02:25:00	NaN	NaN	NaN	NaN
...	...	...	...	...
2000-01-11 21:35:00	NaN	NaN	NaN	NaN
2000-01-11 21:40:00	NaN	NaN	NaN	NaN

	open	high	low	close
2000-01-11 21:45:00	NaN	NaN	NaN	NaN

2000-01-11 21:45:00	NaN	NaN	NaN	NaN
2000-01-11 21:50:00	NaN	NaN	NaN	NaN
2000-01-11 21:55:00	NaN	NaN	NaN	NaN
2000-01-11 22:00:00	NaN	NaN	NaN	NaN
2000-01-11 22:05:00	NaN	NaN	NaN	NaN
2000-01-11 22:10:00	NaN	NaN	NaN	NaN
2000-01-11 22:15:00	NaN	NaN	NaN	NaN
2000-01-11 22:20:00	NaN	NaN	NaN	NaN
2000-01-11 22:25:00	NaN	NaN	NaN	NaN
2000-01-11 22:30:00	NaN	NaN	NaN	NaN
2000-01-11 22:35:00	NaN	NaN	NaN	NaN
2000-01-11 22:40:00	NaN	NaN	NaN	NaN
2000-01-11 22:45:00	NaN	NaN	NaN	NaN
2000-01-11 22:50:00	NaN	NaN	NaN	NaN
2000-01-11 22:55:00	NaN	NaN	NaN	NaN
2000-01-11 23:00:00	NaN	NaN	NaN	NaN
2000-01-11 23:05:00	NaN	NaN	NaN	NaN
2000-01-11 23:10:00	NaN	NaN	NaN	NaN
2000-01-11 23:15:00	NaN	NaN	NaN	NaN
2000-01-11 23:20:00	NaN	NaN	NaN	NaN
2000-01-11 23:25:00	NaN	NaN	NaN	NaN
2000-01-11 23:30:00	NaN	NaN	NaN	NaN
2000-01-11 23:35:00	NaN	NaN	NaN	NaN
2000-01-11 23:40:00	NaN	NaN	NaN	NaN
2000-01-11 23:45:00	NaN	NaN	NaN	NaN
2000-01-11 23:50:00	NaN	NaN	NaN	NaN
2000-01-11 23:55:00	NaN	NaN	NaN	NaN

	open	high	low	close
2000-01-12 00:00:00	0.636269	0.636269	0.636269	0.636269

2000-01-12 00:00:00	0.030209	0.030209	0.030209	0.030209

2881 rows × 4 columns

降采样

将数据聚合到规整的低频率是一件非常普通的时间序列处理任务。待聚合的数据不必拥有固定的频率，期望的频率会自动定义聚合的面元边界，这些面元用于将时间序列拆分为多个片段。例如，要转换到月度频率（M'或‘BM），数据需要被划分到多个单月时间段中。各时间段都是半开放的。一个数据点只能属于一个时间段，所有时间段的并集必须能组成整个时间帧。在用resample对数据进行降采样时，需要考虑两样东西：
·各区间哪边是闭合的。
·如何标记各个聚合面元，用区间的开头还是末尾。

[44]
```
rng = pd.date_range('1/1/2019',periods=12,freq='T')
ts = pd.Series(np.arange(12),index=rng)
ts
```

```
2019-01-01 00:00:00     0
2019-01-01 00:01:00     1
2019-01-01 00:02:00     2
2019-01-01 00:03:00     3
2019-01-01 00:04:00     4
2019-01-01 00:05:00     5
2019-01-01 00:06:00     6
2019-01-01 00:07:00     7
2019-01-01 00:08:00     8
2019-01-01 00:09:00     9
2019-01-01 00:10:00    10
2019-01-01 00:11:00    11
Freq: T, dtype: int32
```

[45]
```
#通过求和的方式将这些数据聚合到“5分钟”中
ts.resample('5min',how='sum')
```

```
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning:
how in .resample() is deprecated
the new syntax is .resample(...).sum()
```

```
2019-01-01 00:00:00    10
2019-01-01 00:05:00    35
2019-01-01 00:10:00    21
Freq: 5T, dtype: int32
```

[46]
```
#传入的频率将会以“5分钟”的增量定义面元边界。
#默认情况下，面元的右边界是包含的，因此00：00到00：05的区间中是包含00：05的
1。
#选择包含右边界，不包含左边界
ts.resample('5min',how='sum',closed='right')
```

```
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: FutureWarning:
how in .resample() is deprecated
the new syntax is .resample(...).sum()
  after removing the cwd from sys.path.
```

```
2018-12-31 23:55:00     0
2019-01-01 00:00:00    15
2019-01-01 00:05:00    40
2019-01-01 00:10:00    11
Freq: 5T, dtype: int32
```

[47]
```
#选择左边界，右边界作为标签
ts.resample('5min',how='sum',closed='left',label='right')
```

```
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning:
how in .resample() is deprecated
the new syntax is .resample(...).sum()
```

```
2019-01-01 00:05:00    10
2019-01-01 00:10:00    35
2019-01-01 00:15:00    21
Freq: 5T, dtype: int32
```

[48]
```
#如果你希望对结果索引进行位移，可以使用loffser
ts.resample('5min',how='sum',loffset='-1s')
```

```
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning:
how in .resample() is deprecated
the new syntax is .resample(...).sum()
```

```
2018-12-31 23:59:59    10
2019-01-01 00:04:59    35
2019-01-01 00:09:59    21
Freq: 5T, dtype: int32
```

另一种降采样的办法是使用pandas的groupby功能。例如，你打算根据月份或星期几进行分组，只需传入一个能够访问时间序列的索引上的这些字段的函数即可：

[49]
```
#通过groupby功能实现降采样
rng = pd.date_range('1/1/2000',periods=100,freq='D')
ts = pd.Series(np.arange(100),index=rng)
ts.groupby(lambda x:x.month).mean()
```

```
1    15
2    45
3    75
4    95
dtype: int32
```

[50]
```
ts.groupby(lambda x:x.weekday).mean()
```

```
0    47.5
1    48.5
2    49.5
3    50.5
4    51.5
5    49.0
6    50.0
dtype: float64
```

升采样和插值

[51]
```
frame = pd.DataFrame(np.random.randn(2,4),
                     index =
pd.date_range('1/1/2000',periods=2,freq='W-Wed'),
                    columns=['Colorado','Texas','New
York','Ohio'])
frame
```

	Colorado	Texas	New York	Ohio
2000-01-05	-0.116549	-0.586241	0.535678	0.908759

	Colorado	Texas	New York	Ohio
2000-01-12	0.900270	1.486212	-0.025083	0.473045

[52]
```
#将其重采样到日频率，默认引入缺失值
#新版本需要加上asfreq()获取值
#m默认会引入缺失值
df_daily = frame.resample('D').asfreq()
df_daily
```

	Colorado	Texas	New York	Ohio
2000-01-05	-0.116549	-0.586241	0.535678	0.908759
2000-01-06	NaN	NaN	NaN	NaN
2000-01-07	NaN	NaN	NaN	NaN
2000-01-08	NaN	NaN	NaN	NaN
2000-01-09	NaN	NaN	NaN	NaN
2000-01-10	NaN	NaN	NaN	NaN
2000-01-11	NaN	NaN	NaN	NaN
2000-01-12	0.900270	1.486212	-0.025083	0.473045

[53]
```
#也可以使用前面的周型值进行填充
frame.resample('D',fill_method='ffill').asfreq('D')
```

```
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning:
fill_method is deprecated to .resample()
the new syntax is .resample(...).ffill()
```

	Colorado	Texas	New York	Ohio
2000-01-05	-0.116549	-0.586241	0.535678	0.908759
2000-01-06	-0.116549	-0.586241	0.535678	0.908759
2000-01-07	-0.116549	-0.586241	0.535678	0.908759
2000-01-08	-0.116549	-0.586241	0.535678	0.908759
2000-01-09	-0.116549	-0.586241	0.535678	0.908759
2000-01-10	-0.116549	-0.586241	0.535678	0.908759
2000-01-11	-0.116549	-0.586241	0.535678	0.908759
2000-01-12	0.900270	1.486212	-0.025083	0.473045

[54]
```
#限制填充期数
```

```
frame.resample('D',fill_method='ffill',limit=2).asfreq('D')
```

```
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning:
fill_method is deprecated to .resample()
the new syntax is .resample(...).ffill(limit=2)
```

	Colorado	Texas	New York	Ohio
2000-01-05	-0.116549	-0.586241	0.535678	0.908759
2000-01-06	-0.116549	-0.586241	0.535678	0.908759
2000-01-07	-0.116549	-0.586241	0.535678	0.908759
2000-01-08	NaN	NaN	NaN	NaN
2000-01-09	NaN	NaN	NaN	NaN
2000-01-10	NaN	NaN	NaN	NaN
2000-01-11	NaN	NaN	NaN	NaN
2000-01-12	0.900270	1.486212	-0.025083	0.473045

通过时期进行重采样

由于时期指的是时间区间，所以升采样和降采样的规则就比较严格：
·在降采样中，目标频率必须是源频率的子时期（subperiod）。
·在升采样中，目标频率必须是源频率的超时期（superperiod）。
如果不满足这些条件，就会引发异常。这主要影响的是按季、年、周计算的频率。例如，由Q-MAR定义的时间区间只能升采样为A-MAR、A-JUN、A-SEP、A-DEC等：

[55]
```
frame = pd.DataFrame(np.random.randn(24,4),
                     index = pd.period_range('1-2000','12-
2001',freq='M'),
                    columns=['Colorado','Texas','New
York','Ohio'])
frame[:5]
```

	Colorado	Texas	New York	Ohio
2000-01	-1.360795	0.829555	-0.596584	0.791686
2000-02	-1.346094	0.120454	-1.179666	-1.021255
2000-03	-1.557079	0.815604	-1.154167	-1.200424

2000-04	-1.727358	0.022920	0.633838	0.590345
2000-05	-1.978317	0.397222	-0.384799	-0.978572

[56]
```
#降采样
annual_frame = frame.resample('A-DEC',how='mean')
annual_frame
```

```
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning:
how in .resample() is deprecated
the new syntax is .resample(...).mean()
```

	Colorado	Texas	New York	Ohio
2000	-0.690458	-0.215923	-0.161279	0.237725
2001	-0.036384	-0.045400	-0.221312	0.090525

[57]
```
#升采样稍微麻烦，因为需要决定在新的频率各区间的哪端用于放置原来的值
#convention参数默认为start，可设置为end
#重新把年的汇总转为为季度型：Q-DEC，季度型，每年以12月结束
annual_frame.resample('Q-DEC',fill_method='ffill')
```

```
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: FutureWarning:
fill_method is deprecated to .resample()
the new syntax is .resample(...).ffill()
  after removing the cwd from sys.path.
```

	Colorado	Texas	New York	Ohio
2000Q1	-0.690458	-0.215923	-0.161279	0.237725
2000Q2	-0.690458	-0.215923	-0.161279	0.237725
2000Q3	-0.690458	-0.215923	-0.161279	0.237725
2000Q4	-0.690458	-0.215923	-0.161279	0.237725
2001Q1	-0.036384	-0.045400	-0.221312	0.090525
2001Q2	-0.036384	-0.045400	-0.221312	0.090525
2001Q3	-0.036384	-0.045400	-0.221312	0.090525

	Colorado	Texas	New York	Ohio
2001Q4	-0.036384	-0.045400	-0.221312	0.090525

[58]
```
annual_frame.resample('Q-
DEC',fill_method='ffill',convention='end')
```

```
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWarning:
fill_method is deprecated to .resample()
the new syntax is .resample(...).ffill()
  """Entry point for launching an IPython kernel.
```

	Colorado	Texas	New York	Ohio
2000Q4	-0.690458	-0.215923	-0.161279	0.237725
2001Q1	-0.690458	-0.215923	-0.161279	0.237725
2001Q2	-0.690458	-0.215923	-0.161279	0.237725
2001Q3	-0.690458	-0.215923	-0.161279	0.237725
2001Q4	-0.036384	-0.045400	-0.221312	0.090525

七、移动窗口函数

在移动窗口（可以带有指数衰减权数）上计算的各种统计函数也是一类常见于时间序列的数组变换。我将它们称为移动窗口函数（moving window function），其中还包括那些窗口不定长的函数（如指数加权移动平均）。跟其他统计函数一样，移动窗口函数也会自动排除缺失值。

```
DataFrame.rolling(window, min_periods=None, center=False,
win_type=None, on=None, axis=0, closed=None)
window       时间窗大小
min_periods 每个窗口最少包含的观测数量
center       窗口的标签设置为居中
win_type     窗口的类型
on           可选参数，指定要计算的列名
axis         轴向
closed       定义区间的开闭
```

[59]
```
index=pd.date_range('20190116','20190130')
data=[4,8,6,5,9,1,4,5,2,4,6,7,9,13,6]
ser_data=pd.Series(data,index=index)
ser_data
```

```
2019-01-16     4
2019-01-17     8
```

```
2019-01-18     6
2019-01-19     5
2019-01-20     9
2019-01-21     1
2019-01-22     4
2019-01-23     5
2019-01-24     2
2019-01-25     4
2019-01-26     6
2019-01-27     7
2019-01-28     9
2019-01-29    13
2019-01-30     6
Freq: D, dtype: int64
```

[60]
```
ser_data.rolling(3).mean()
```

```
2019-01-16         NaN
2019-01-17         NaN
2019-01-18    6.000000
2019-01-19    6.333333
2019-01-20    6.666667
2019-01-21    5.000000
2019-01-22    4.666667
2019-01-23    3.333333
2019-01-24    3.666667
2019-01-25    3.666667
2019-01-26    4.000000
2019-01-27    5.666667
2019-01-28    7.333333
2019-01-29    9.666667
2019-01-30    9.333333
Freq: D, dtype: float64
```

当窗口开始滑动时，第一个时间点和第二个时间点的时间为空，这是因为这里窗口长度为3，他们前面的数都不够3，所以到2019-01-18时，他的数据就是2019-01-16到2019-01-18三天的均值。
那么，在计算2019-01-16序列的窗口数据时，虽然不够窗口长度3，但是至少有当天的数据，那么能否就用当天的数据代表窗口数据呢？答案是肯定的，这里我们可以通过min_periods参数控制，表示窗口最少包含的观测值，小于这个值的窗口长度显示为空，等于和大于时有值，如下所示：

[61]
```
ser_data.rolling(3,min_periods=1).mean()#表示窗口最少包含的观测值为1
```

```
2019-01-16    4.000000
2019-01-17    6.000000
2019-01-18    6.000000
```

```
2019-01-19    6.333333
2019-01-20    6.666667
2019-01-21    5.000000
2019-01-22    4.666667
2019-01-23    3.333333
2019-01-24    3.666667
2019-01-25    3.666667
2019-01-26    4.000000
2019-01-27    5.666667
2019-01-28    7.333333
2019-01-29    9.666667
2019-01-30    9.333333
Freq: D, dtype: float64
```

除此之外，还有一些其它的rolling函数，均是以rolling为前缀的函数
rolling_count返回各窗口非NA观测值的数量
rolling_sum移动窗口的和
rolling_mean移动窗口的平均值
rolling_median移动窗口的中位数
rolling_var、rolling_std移动窗口的方差和标准差。分母为n-1
rolling_skew、olling_kurt移动窗口的偏度（三阶矩）和峰度（四阶矩）
rolling_min、rolling_max移动窗口的最小值和最大值
rolling_quantile移动窗口指定百分位数/样本分位数位置的值
rolling_cor、rolling_cov移动窗口的相关系数和协方差
rolling_apply对移动窗口应用普通数组函数

[62]
```
print("各窗口非NA观测值的数量:",pd.rolling_count(ser_data,window =
3))
print("移动窗口的和:",pd.rolling_sum(ser_data,window = 3))
print("移动窗口的平均值:",pd.rolling_mean(ser_data,window = 3))
print("移动窗口的中位数:",pd.rolling_median(ser_data,window = 3))
print("移动窗口的方差:",pd.rolling_var(ser_data,window = 3))
print("移动窗口的标准差:",pd.rolling_std(ser_data,window = 3))
```

```
各窗口非NA观测值的数量: 2019-01-16    1.0
2019-01-17    2.0
2019-01-18    3.0
2019-01-19    3.0
2019-01-20    3.0
2019-01-21    3.0
2019-01-22    3.0
2019-01-23    3.0
2019-01-24    3.0
2019-01-25    3.0
2019-01-26    3.0
2019-01-27    3.0
2019-01-28    3.0
2019-01-29    3.0
2019-01-30    3.0
```

```
Freq: D, dtype: float64
移动窗口的和: 2019-01-16     NaN
2019-01-17     NaN
2019-01-18    18.0
2019-01-19    19.0
2019-01-20    20.0
2019-01-21    15.0
2019-01-22    14.0
2019-01-23    10.0
2019-01-24    11.0
2019-01-25    11.0
2019-01-26    12.0
2019-01-27    17.0
2019-01-28    22.0
2019-01-29    29.0
2019-01-30    28.0
Freq: D, dtype: float64
移动窗口的平均值: 2019-01-16         NaN
2019-01-17         NaN
2019-01-18    6.000000
2019-01-19    6.333333
2019-01-20    6.666667
2019-01-21    5.000000
2019-01-22    4.666667
2019-01-23    3.333333
2019-01-24    3.666667
2019-01-25    3.666667
2019-01-26    4.000000
2019-01-27    5.666667
2019-01-28    7.333333
2019-01-29    9.666667
2019-01-30    9.333333
Freq: D, dtype: float64
移动窗口的中位数: 2019-01-16    NaN
2019-01-17    NaN
2019-01-18    6.0
2019-01-19    6.0
2019-01-20    6.0
2019-01-21    5.0
2019-01-22    4.0
2019-01-23    4.0
2019-01-24    4.0
2019-01-25    4.0
2019-01-26    4.0
2019-01-27    6.0
2019-01-28    7.0
2019-01-29    9.0
2019-01-30    9.0
Freq: D, dtype: float64
移动窗口的方差: 2019-01-16          NaN
2019-01-17          NaN
2019-01-18     4.000000
```

```
2019-01-19     2.333333
2019-01-20     4.333333
2019-01-21    16.000000
2019-01-22    16.333333
2019-01-23     4.333333
2019-01-24     2.333333
2019-01-25     2.333333
2019-01-26     4.000000
2019-01-27     2.333333
2019-01-28     2.333333
2019-01-29     9.333333
2019-01-30    12.333333
Freq: D, dtype: float64
移动窗口的标准差: 2019-01-16         NaN
2019-01-17         NaN
2019-01-18    2.000000
2019-01-19    1.527525
2019-01-20    2.081666
2019-01-21    4.000000
2019-01-22    4.041452
2019-01-23    2.081666
2019-01-24    1.527525
2019-01-25    1.527525
2019-01-26    2.000000
2019-01-27    1.527525
2019-01-28    1.527525
2019-01-29    3.055050
2019-01-30    3.511885
Freq: D, dtype: float64
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWarning:
pd.rolling_count is deprecated for Series and will be removed in a
future version, replace with
        Series.rolling(window=3).count()
  """Entry point for launching an IPython kernel.
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning:
pd.rolling_sum is deprecated for Series and will be removed in a future
version, replace with
        Series.rolling(window=3,center=False).sum()

D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: FutureWarning:
pd.rolling_mean is deprecated for Series and will be removed in a future
version, replace with
        Series.rolling(window=3,center=False).mean()
  This is separate from the ipykernel package so we can avoid doing
imports until
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: FutureWarning:
pd.rolling_median is deprecated for Series and will be removed in a
future version, replace with
        Series.rolling(window=3,center=False).median()
  after removing the cwd from sys.path.
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:5: FutureWarning:
pd.rolling_var is deprecated for Series and will be removed in a future
```

```
version, replace with
        Series.rolling(window=3,center=False).var()
  """
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: FutureWarning:
pd.rolling_std is deprecated for Series and will be removed in a future
version, replace with
        Series.rolling(window=3,center=False).std()
```

八、指数加权函数

另一种使用固定大小窗口及相等权数观测值的办法是，定义一个衰减因子常量，以便使近期的观测值拥有更大的权数。用数学术语来讲，如果ma1是时间t的移动平均结果，x是时间序列，结果中的各个值可用ma1=ama(t-1)+（a-1）x(t-1)进行计算，其中a为衰减因子。衰减因子的定义方式有很多，比较流行的是使用时间间隔（span），它可以使结果兼容于窗口大小等于时间间隔的简单移动窗口函数。由于指数加权统计会赋予近期的观测值更大的权数，因此相对于等权统计，它能“适应”更快的变化。

```
ewma(arg,com=None,span=None,halflife=None,min_periods=0,freq=None,adjus
t =True,how = None,ignore_na = False)指数加权移动平均
ewmvar(arg,com=None,span=None,halflife=None,min_periods=0,bias=False,fr
eq=None,how=None,ignore_na=False,adjust=True)指数加权移动方差
ewmstd(arg,com=None,span=None,halflife=None,min_periods=0,bias=False,ig
nore_na=False,adjust=True) 指数加权移动标准差
ewmcorr(arg1,arg2=None,com=None,span=None,halflife=None,min_periods=0,f
req=None,pairwise=None,how=None,ignore_na=False,adjust=True)指数加权移动
相关系数
ewmcov(arg1,arg2,com=None,span=None,halflife=None,min_periods=0,bias=Fa
lse,freq=None) 指数加权移动协方差
参数说明
arg，arg1，arg2：Series，DataFrame
com：质量中心。可选的
span：根据跨度指定衰减，可选
halflife：根据半衰期指定衰减，可选
min_periods：具有值所需的窗口中的最小观察数（否则结果为NA），默认值为0
bias：使用标准估计偏差校正，默认为False
how：用于下采样或重采样的方法，默认为mean
ignore_na：计算权重时忽略缺失值；指定True以重现0.15.0之前的行为，默认为False
freq：在计算统计数据之前要符合的频率，默认=无
adjust：除以初期的衰减调整因子，以解释相对权重的不平衡（将EWMA视为移动平均线），默认
为True
```

[63]
```
import statsmodels.api as sm

data_loader = sm.datasets.sunspots.load_pandas()
```

```
df = data_loader.data
df.head(15)
```

```
D:\Anaconda3\lib\site-packages\statsmodels\compat\pandas.py:56:
FutureWarning: The pandas.core.datetools module is deprecated and will
be removed in a future version. Please use the pandas.tseries module
instead.
  from pandas.core import datetools
```

	YEAR	SUNACTIVITY
0	1700.0	5.0
1	1701.0	11.0
2	1702.0	16.0
3	1703.0	23.0
4	1704.0	36.0
5	1705.0	58.0
6	1706.0	29.0
7	1707.0	20.0
8	1708.0	10.0
9	1709.0	8.0
10	1710.0	3.0
11	1711.0	0.0
12	1712.0	0.0
13	1713.0	2.0
14	1714.0	11.0

[64]
```
print("EMA:",df["SUNACTIVITY"].ewm(span=10,min_periods=10).mean()
.head(15))
```

```
EMA: 0          NaN
1           NaN
2           NaN
3           NaN
4           NaN
5           NaN
6           NaN
7           NaN
8           NaN
```

```
9     20.690866
10    17.076843
11    13.664921
12    10.982917
13     9.244962
14     9.580603
Name: SUNACTIVITY, dtype: float64
```