时间处理基础

Pandas提供了四种类型的生成日期时间的对象：日期时间、时间增量、时间跨度、日期偏移量

（1） 日期时间(Date Times)：具有时区支持的特定日期和时间。与Python标准库中的datetime.datetime类似。如2020年12月6日13点37分50秒；

（2） 时间增量(Time Deltas)：绝对持续时间，用于在指定时间点基础上增加指定的增量，如在某年月日的基础上增加2天、增加2个月、减少4小时等，最后产生一个新的时间点；

（3） 时间跨度(Time Span)：由时间点及其相关周期定义的时间跨度，如连续产生一年四个季度的时间序列；

（4） 日期偏移(Date Offsets)：以日历计算的相对持续时间，表示时间间隔，两个时间点之间的长度，如日、周、月、季度、年。

概念	标量类	数组类	Pandas数据类型	主要建立方法
日期时间(Date Times)	Timesstamp时间戳	DatetimeIndex时间索引	datetime64[ns]、datetime64[ns,tz]	to_datetime()、date_range()
时间增量(Time Deltas)	Timedelta时间增量	TimedeltaIndex时间增量索引	timedelta[ns]	to_timedelta()、timedelta_range()
时间跨度(Time Span)	Period时间周期	PeriodIndex周期索引	period[freq]	Period()、period_range()
日期偏移(Date Offsets)	DateOffset	None	None	DateOffset()

一般情况下，时间序列主要是 `Series` 或 `DataFrame` 的时间型索引，可以用时间元素进行操控。

```
import pandas as pd

import numpy as np

import time

import datetime
```

一、时间序列类型

1.时间戳

python datetime的替代品

时间戳相当于python的Datetime，在大多数情况下可以与之互换。

该类型用于组成DatetimeIndex的条目，以及pandas中其他面向时间序列的数据结构。

例子:

- 转换类似日期时间的字符串

```
pd.Timestamp('2022-01-01 12')
```

```
Timestamp('2022-01-01 12:00:00')
```

```
pd.Timestamp('2021-12-15 12')
```

```
Timestamp('2021-12-15 12:00:00')
```

```
pd.Timestamp('01-01-2022 12')
```

```
Timestamp('2022-01-01 12:00:00')
```

```
pd.Timestamp('2022-01')
```

- unit参数为s:

  这将转换以秒为单位表示Unix历元的浮点值

  1970年1月1日这个时间正是Unix系统的起始时间

```
pd.Timestamp(time.time(), unit="s")
```

```
Timestamp('2022-04-18 09:43:34.341360092')
```

```
pd.Timestamp(time.time())
```

```
Timestamp('1970-01-01 00:00:01.650275101')
```

- year, month, day,hour, minute, second, microsecond单独设置时间

```
pd.Timestamp(2022, 1, 1, 12)
```

```
# 提供年月日
pd.Timestamp(year=2022, month=1,day=1)
```

```
Timestamp('2022-01-01 00:00:00')
```

```
pd.Timestamp(year=2022, month=1, day=1, hour=12)
```

```
总结:

Timestamp

- 将字符串或者unix时间转化为 pandas对应的时间戳
```

2.时间间隔--实现datetime加减

表示持续时间，即两个日期或时间之间的差异。

相当于python的datetime.timedelta，在大多数情况下可以与之互换

```
ts = pd.Timestamp('2022-01-01 12')
ts + pd.Timedelta(-1, "d")
```

```
Timestamp('2021-12-31 12:00:00')
```

```
#时间间隔
td = pd.Timedelta(days=5, minutes=50, seconds=20)
td
```

```
Timedelta('5 days 00:50:20')
```

```
ts - td
```

```
Timestamp('2021-12-27 11:09:40')
```

```
td.total_seconds()
```

```
import datetime
# 取得当前时间
now = datetime.datetime.now()
print(now)

#计算当前时间往后100天的日期
#dt = now + pd.Timedelta(days=100)
dt = now + datetime.timedelta(days=100)
print(dt,type(dt))
#只显示年月日
dt.strftime('%Y-%m-%d')
```

```
2022-04-18 17:56:55.717635
2022-07-27 17:56:55.717635 <class 'datetime.datetime'>
```

```
'2022-07-27'
```

```
dt = pd.Timestamp('2022-01-11')
#dt + pd.Timedelta(days=100)
dt + datetime.timedelta(days=100)
```

```
Timestamp('2022-04-21 00:00:00')
```

3.时间转化

`to_datetime` 转换时间戳

你可能会想到，我们经常要和文本数据（字符串）打交道，能否快速将文本数据转为时间戳呢？

```
1    答案是可以的，通过 to_datetime 能快速将字符串转换为时间戳。当传递一个Series时，它会返回一个Series（具有相同的索引），而类似列表的则转换为DatetimeIndex。
```

`to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, format=None, unit=None, infer_datetime_format=False, origin='unix')`

函数用户将数组、序列或dict的对象转换为datetime对象

- `arg` 要转换为日期时间的对象
- `errors` :错误处理

  If 'raise',将引发异常.

  If 'coerce', 无效的转换,使用NaT.

  If 'ignore', 无效的转换,将使用输入的数据.
- `dayfirst` :转换时指定日期分析顺序
- `utc` :控制与时区相关的解析、本地化和转换(忽略)
- `format` : 用于分析时间的strftime，例如“%d/%m/%Y”,自定义格式
- `unit` : D,s,ms 将时间戳转datetime
- `origin` : 定义参考日期。数值将被解析为自该参考日期起的单位数

1). 处理各种输入格式

```
1  从一个数据帧的多个列中组装日期时间。
2
3  这些键可以是常见的缩写，
4
5  如['year'、'month'、'day'、'minute'、'second'、'ms'、'us'、'ns']) ，
6
7  也可以是相同的复数形式
```

```
df = pd.DataFrame({'year': [2015, 2016],'month': [2, 3],'day': [4, 5]})
df
```

```
pd.to_datetime(df)
```

2). 将字符串转datetime

```
pd.to_datetime(['11-12-2021'])
```

```
pd.to_datetime(["2005/11/23", "2010.12.31"])
```

3). 除了可以将文本数据转为时间戳外，还可以将 unix 时间转为时间戳。

```
pd.to_datetime([1349720105, 1349806505, 1349892905], unit="s")
```

```
pd.to_datetime([1349720105100, 1349720105200, 1349720105300], unit="ms")
```

4). 自动识别异常

- 210605

```
pd.to_datetime('210605')
```

```
pd.to_datetime('210605',yearfirst=True)
```

5). 配合unit参数,使用非unix时间

```
# origin参考起始时间
pd.to_datetime([1, 2, 3], unit='D', origin=pd.Timestamp('2020-01-11'))
```

6). 不可转换日期/时间

```
    如果日期不符合时间戳限制，则传递errors='ignore'将返回原始输入，而不是引发任何异常。

    除了将非日期（或不可解析的日期）强制传递给NaT之外，传递errors='coerce'还会将越界日期强制传递给NaT
```

- errors`:错误处理

  If 'raise',将引发异常.

  If 'coerce', 无效的转换,使用NaT.

  If 'ignore', 无效的转换,将使用输入的数据.

```
pd.to_datetime('120211204')
```

```
pd.to_datetime('120211204',errors="ignore")
```

```
pd.to_datetime('120211204',errors="coerce")
```

```
pd.to_datetime(pd.Series(["Jul 31, 2018", "2018-05-10", None]))
```

```

```

对于float arg，可能会发生精确舍入。要防止意外行为，请使用固定宽度的精确类型。

4.生成时间戳范围

有时候，我们可能想要生成某个范围内的时间戳。例如，我想要生成 "2018-6-26" 这一天之后的8天时间戳

我们可以使用 `date_range` 和 `bdate_range` 来完成时间戳范围的生成。

date_range() 返回固定频率的DatetimeIndex

```
date_range(start=None, end=None, periods=None, freq=None, tz=None, normalize=False, name=None, closed=None, **kwargs)
```

返回等距时间点的范围（其中任意两个相邻点之间的差值由给定频率指定），以便它们都满足开始<[=]x<[=]end，其中第一个和最后一个分别为。，该范围内的第一个和最后一个时间点位于freq的边界（如果以频率字符串的形式给出）或对freq有效

- start :生成日期的左边界
- end : 生成日期的左边界
- periods: 要生成的周期数
- freq : 频率, default 'D' ,频率字符串可以有倍数, '5H'
- tz: 时区用于返回本地化日期时间索引的时区名称，例如“Asia/Hong_Kong”。默认情况下，生成的DatetimeIndex是时区初始索引。
- normalize: 默认 False, 在生成日期范围之前，将开始/结束日期标准化
- name: 默认 None 设置返回DatetimeIndex name
- closed: 控制是否包括屏幕上的“开始”和“结束”边界默认设置包括两端的边界点 {None, ‘left’, ‘right’}
  (closed 1.4.0被启用) inclusive:包括边界{"both", "neither", "left", "right"}, default "both"

1). 指定值

- 默认是包含开始和结束时间, 默认频率使用的D(天)

```
pd.date_range(start='1/1/2021', end='1/08/2021')
```

```
pd.date_range(start='2010', end='2011')
```

2). 指定开始日期,设置期间数

```
pd.date_range(start='1/1/2018', periods=8)
```

- 指定开始、结束和期间；频率自动生成（线性间隔）

```
pd.date_range(start='2018-04-24', end='2018-04-27', periods=3)
```

```
pd.date_range(start='2018-04-24', end='2018-04-27', periods=4)
```

3). 频率 freq

freq	描述
Y	年
M	月
D	日(默认)
T(MIN)	分钟
S	秒
L	毫秒
U	微秒
A-DEC	每年指定月份的最后一个日历日
W-MON	指定每月的哪个星期开始
WOM_2MON	指定每个月的第几个星期(这里是第二个星期)
Q-DEC(Q-月)	指定月为季度末，每个季度末最后一月的最后一个日历日
B,(M,Q,A),S	分别代表了工作日，（以月为频率，以季度为频率，以年为频率），最接近月初的那一天
B	工作日

- 月缩写：JAN/FEB/MAR/APR/MAY/JUN/JUL/AUG/SEP/OCT/NOV/DEC
- 星期几缩写：MON/TUE/WED/THU/FRI/SAT/SUN
- 所以Q-月只有三种情况：1-4-7-10,2-5-8-11,3-6-9-12

```
pd.date_range('2022/1/1','2022/2/1', freq = 'W-MON')
# W-MON: 从指定星期几开始算起，每周
# 星期几缩写: MON/TUE/WED/THU/FRI/SAT/SUN
```

```
pd.date_range('2021/1/1','2021/5/1', freq = 'WOM-2MON')
# WOM-2MON: 每月的第几个星期几开始算，这里是每月第二个星期一
```

```
# 默认freq = 'D': 每日历日
pd.date_range('2022/1/1','2022/1/4')
```

```
pd.date_range('2022/1/1','2022/1/5', freq = 'B')  # B: 每工作日
```

```
pd.date_range('2022/1/1','2022/1/2', freq = 'H')  # H: 每小时
```

```
pd.date_range('2022/1/1 12:00','2022/1/1 12:10', freq = 'T')  # T/MIN: 每分
```

```
print(pd.date_range('2022/1/1 12:00:00','2022/1/1 12:00:10', freq = 'S'))  # S: 每秒
```

```
print(pd.date_range('2022/1/1 12:00:00','2022/1/1 12:00:10', freq = 'L'))  # L: 每毫秒（千分之一秒）
```

```
pd.date_range('2022/1/1 12:00:00','2022/1/1 12:00:10', freq = 'U')
 # U: 每微秒（百万分之一秒）
```

```
# M: 每月最后一个日历日
pd.date_range('2017','2018', freq = 'M')
```

```
# Q-月: 指定月为季度末，每个季度末最后一月的最后一个日历日
print(pd.date_range('2017','2020', freq = 'Q-DEC'))
```

```
DatetimeIndex(['2017-03-31', '2017-06-30', '2017-09-30', '2017-12-31',
               '2018-03-31', '2018-06-30', '2018-09-30', '2018-12-31',
               '2019-03-31', '2019-06-30', '2019-09-30', '2019-12-31'],
              dtype='datetime64[ns]', freq='Q-DEC')
```

```
# A-月: 每年指定月份的最后一个日历日
print(pd.date_range('2017','2020', freq = 'A-DEC'))
```

```
DatetimeIndex(['2017-12-31', '2018-12-31', '2019-12-31'],
dtype='datetime64[ns]', freq='A-DEC')
```

```
# B,(M,Q,A),S   分别代表了工作日，（以月为频率，以季度为频率，以年为频率），最接近月初的那一天
print(pd.date_range('2017','2018', freq = 'BM'))
print(pd.date_range('2017','2020', freq = 'BQ-DEC'))
print(pd.date_range('2017','2020', freq = 'BA-DEC'))
print('------')
# BM：每月最后一个工作日
# BQ-月：指定月为季度末，每个季度末最后一月的最后一个工作日
# BA-月：每年指定月份的最后一个工作日
```

```
print(pd.date_range('2017','2018', freq = 'MS'))
print(pd.date_range('2017','2020', freq = 'QS-DEC'))
print(pd.date_range('2017','2020', freq = 'AS-DEC'))
print('------')
# M：每月第一个日历日
# Q-月：指定月为季度末，每个季度末最后一月的第一个日历日
# A-月：每年指定月份的第一个日历日
```

```
print(pd.date_range('2017','2018', freq = 'BMS'))
print(pd.date_range('2017','2020', freq = 'BQS-DEC'))
print(pd.date_range('2017','2020', freq = 'BAS-DEC'))
print('------')
# BM：每月第一个工作日
# BQ-月：指定月为季度末，每个季度末最后一月的第一个工作日
# BA-月：每年指定月份的第一个工作日
```

```
# pd.date_range()-日期范围：复合频率

print(pd.date_range('2017/1/1','2017/2/1', freq = '7D'))  # 7天
print(pd.date_range('2017/1/1','2017/1/2', freq = '2h30min'))  # 2小时30分钟
print(pd.date_range('2017','2018', freq = '2M'))  # 2月，每月最后一个日历日
```

```
pd.date_range(start='1/1/2018', periods=5, freq='3M')
```

4). `closed`：边界设置

```
    控制是否包括屏幕上的“开始”和“结束”边界默认设置包括两端的边界点{None, ‘left’, ‘right’}
```

```
pd.date_range(start='2022-01-01', end='2022-01-04', closed=None)
```

```
pd.date_range(start='2022-01-01', end='2022-01-04')
```

如果需要排除"end",则使用“closed='left'”

```
pd.date_range(start='2022-01-01', end='2022-01-04',closed="left")
```

如果需要排除"start",则使用“closed='right'”

```
pd.date_range(start='2022-01-01', end='2022-01-04',closed="right")
```

5) `normalize`: 在生成日期范围之前，将开始/结束日期标准化

```
pd.date_range(start = '1/1/2021 15:30', periods =10, name =
'mypd')
```

```
pd.date_range(start = '1/1/2021 15:30', periods =10, name =
'mypd',normalize=True)
```

```
pd.date_range(start = '1/1/2021 15:30', periods=10, name='mypd',
freq="H")
```

```
pd.date_range(start = '1/1/2021 15:30', periods=10,
name='mypd',normalize=True, freq="H")
```

```

```

bdate_range(start=None, end=None, periods=None, freq='B')

返回固定频率的DatetimeIndex，默认频率为工作日

```
pd.bdate_range(start='2022-04-01', end='2022-05-01')
```

```
DatetimeIndex(['2022-04-01', '2022-04-04', '2022-04-05', '2022-04-
06',
               '2022-04-07', '2022-04-08', '2022-04-11', '2022-04-
12',
               '2022-04-13', '2022-04-14', '2022-04-15', '2022-04-
18',
               '2022-04-19', '2022-04-20', '2022-04-21', '2022-04-
22',
               '2022-04-25', '2022-04-26', '2022-04-27', '2022-04-
28',
               '2022-04-29'],
              dtype='datetime64[ns]', freq='B')
```

```

```

```

```

5.时期频率转换

`asfreq(freq, method=None, how=None, normalize=False, fill_value=None)`

将时间序列转换为指定的频率

- 如果此数据帧的索引是PeriodIndex，则新索引是使用PeriodIndex转换原始索引的结果
- 否则，新指数将相当于pd。date_range（start，end，freq=freq），其中start和end分别是原始索引中的第一个和最后一个条目
- 与新索引中未出现在原始索引中的任何时间步对应的值将为空（NaN），除非提供了填充此类未知值的方法

```
# asfreq: 时期频率转换

ts = pd.Series(np.random.rand(4),
               index = pd.date_range('20170101','20170104'))
print(ts)
print(ts.asfreq('4H', method = 'ffill'))
# 改变频率，这里是D改为4H
# method: 插值模式，None不插值，ffill用之前值填充，bfill用之后值填充
```

```
2017-01-01    0.936773
2017-01-02    0.021578
2017-01-03    0.622016
2017-01-04    0.102583
Freq: D, dtype: float64
2017-01-01 00:00:00    0.936773
2017-01-01 04:00:00    0.021578
2017-01-01 08:00:00    0.021578
2017-01-01 12:00:00    0.021578
2017-01-01 16:00:00    0.021578
2017-01-01 20:00:00    0.021578
2017-01-02 00:00:00    0.021578
2017-01-02 04:00:00    0.622016
2017-01-02 08:00:00    0.622016
2017-01-02 12:00:00    0.622016
2017-01-02 16:00:00    0.622016
2017-01-02 20:00:00    0.622016
2017-01-03 00:00:00    0.622016
2017-01-03 04:00:00    0.102583
```

```
2017-01-03 08:00:00    0.102583
2017-01-03 12:00:00    0.102583
2017-01-03 16:00:00    0.102583
2017-01-03 20:00:00    0.102583
2017-01-04 00:00:00    0.102583
Freq: 4H, dtype: float64
```

6. shift()-时间频率进行移位

`shift(periods=1, freq=None, axis=0, fill_value=None)`

按所需的时段数和可选的时间频率进行移位索引。

如果未传递freq，则在不重新调整数据的情况下移动索引。如果传递了freq（在这种情况下，索引必须是date或datetime，否则将引发NotImplementedError），只要在索引中设置了freq或推断的_freq属性，就可以推断freq

- periods: 要转换的时段数。可以是正面的，也可以是负面的
- freq:如果指定了freq，则索引值会移位，但数据不会重新对齐。也就是说，如果要在移动时扩展索引并保留原始数据
- axis:{0 or 'index', 1 or 'columns', None} 转换方向.
- fill_value: 填充值

```
df = pd.DataFrame(np.random.rand(16).reshape((4,4)),
                  index = pd.date_range('20210101','20210104'),
                  columns=list('ABCD'))
df
```

```
# 正数：数值后移（滞后），模式为行
df.shift(periods=2)
```

```
# 正数：数值后移（滞后） 设置为列
df.shift(periods=1, axis="columns")
```

```
# 正数：数值后移，NaN填充为0
df.shift(periods=3, fill_value=0)
```

```
# 当设置freq时 ，对时间索引移动
df.shift(periods=3, freq="D")
```

```
# 计算变化百分比，这里计算：该时间戳与上一个时间戳相比，变化百分比
per = ts/ts.shift(1) - 1
print(per)
```

```

```

7. Pandas时期： Period()

```
p = pd.Period('2017')
p
```

```
Period('2017', 'A-DEC')
```

```
p = pd.Period('2017', freq = 'M')
print(p, type(p))
```

```
2017-01 <class 'pandas._libs.tslibs.period.Period'>
```

- 通过加减整数可以实现对Period的移动

```
print(p + 1)
print(p - 2)
```

```
2017-02
2016-11
```

- 如果两个Period对象拥有相同频率，则它们的差就是它们之间的单位数量

```
pd.Period('2018', freq='M') - p
```

```
<12 * MonthEnds>
```

period_range函数可用于创建规则的时期范围

```
rng = pd.period_range('2021-1-1', '2021-6-1', freq='M')
rng
```

```
PeriodIndex(['2021-01', '2021-02', '2021-03', '2021-04', '2021-
05', '2021-06'], dtype='period[M]', freq='M')
```

```
pd.Series(np.random.rand(6), index=rng)
```

```
2021-01    0.051439
2021-02    0.411414
2021-03    0.442789
2021-04    0.556011
2021-05    0.488353
2021-06    0.027299
Freq: M, dtype: float64
```

PeriodIndex类的构造函数允许直接使用一组字符串表示一段时期

```
values = ['200103', '200104', '200105']
# 必须指定 freq
index = pd.PeriodIndex(values, freq='M')
index
```

```
PeriodIndex(['2001-03', '2001-04', '2001-05'], dtype='period[M]',
freq='M')
```

时期的频率转换 asfreq

```
# A-月：每年指定月份的最后一个日历日
p = pd.Period('2021', freq='A-DEC')
p
```

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

```
p.asfreq('M')
```

```
Period('2021-12', 'M')
```

```
p.asfreq('M', how="start") #  # 也可写 how = 's'
```

```
Period('2021-01', 'M')
```

```
p.asfreq('M', how="end") # 也可写 how = 'e'
```

```
Period('2021-12', 'M')
```

```
p.asfreq('D')
```

```
Period('2021-12-31', 'D')
```

对于PeriodIndex或TimeSeries的频率转换方式相同

```
rng = pd.period_range('2006', '2009', freq='A-DEC')
rng
```

```
PeriodIndex(['2006', '2007', '2008', '2009'], dtype='period[A-DEC]', freq='A-DEC')
```

```
ts = pd.Series(np.random.rand(len(rng)), rng)
ts
```

```
2006    0.562403
2007    0.323997
2008    0.673822
2009    0.750930
Freq: A-DEC, dtype: float64
```

```
ts.asfreq('M', how='s')
```

```
2006-01    0.562403
2007-01    0.323997
2008-01    0.673822
2009-01    0.750930
Freq: M, dtype: float64
```

```
ts2 = pd.Series(np.random.rand(len(rng)), index = rng.asfreq('D',
how = 'start'))
ts2
```

```
2006-01-01    0.115021
2007-01-01    0.636835
2008-01-01    0.565186
2009-01-01    0.704160
Freq: D, dtype: float64
```

时间戳与时期之间的转换：pd.to_period()、pd.to_timestamp()

```
rng = pd.date_range('2017/1/1', periods = 10, freq = 'M')
prng = pd.period_range('2017','2018', freq = 'M')
print(rng)
print(prng)
```

```
DatetimeIndex(['2017-01-31', '2017-02-28', '2017-03-31', '2017-04-30',
               '2017-05-31', '2017-06-30', '2017-07-31', '2017-08-31',
               '2017-09-30', '2017-10-31'],
              dtype='datetime64[ns]', freq='M')
PeriodIndex(['2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06',
             '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12',
             '2018-01'],
            dtype='period[M]', freq='M')
```

```
ts1 = pd.Series(np.random.rand(len(rng)), index = rng)
ts1
```

```
2017-01-31    0.579400
2017-02-28    0.387962
2017-03-31    0.933857
2017-04-30    0.348257
2017-05-31    0.592015
2017-06-30    0.596254
2017-07-31    0.442468
2017-08-31    0.628151
2017-09-30    0.236808
2017-10-31    0.989255
Freq: M, dtype: float64
```

```
# 每月最后一日，转化为每月
ts1.to_period().head()
```

```
2017-01    0.579400
2017-02    0.387962
2017-03    0.933857
2017-04    0.348257
2017-05    0.592015
Freq: M, dtype: float64
```

```
ts2 = pd.Series(np.random.rand(len(prng)), index = prng)
print(ts2.head())
print(ts2.to_timestamp().head())
```

```
2017-01    0.743320
2017-02    0.612895
2017-03    0.656805
2017-04    0.438479
2017-05    0.144916
Freq: M, dtype: float64
2017-01-01    0.743320
2017-02-01    0.612895
2017-03-01    0.656805
2017-04-01    0.438479
2017-05-01    0.144916
Freq: MS, dtype: float64
```

时间序列 - 重采样resample

Pandas中的resample，重新采样，是对原样本重新处理的一个方法，是一个对常规时间序列数据重新采样和频率转换的便捷的方法。

重新取样时间序列数据。

方便的时间序列的频率转换和重采样方法。

对象必须具有类似datetime的索引(DatetimeIndex、PeriodIndex或TimedeltaIndex)，或将类似datetime的值传递给on或level关键字。

`DataFrame.resample(rule, closed=None, label=None, level=None)`

- rule: 表示目标转换的偏移量字符串或对象
- closed:在降采样时，各时间段的哪一段是闭合的，‘right’或‘left’，默认‘right’
- label:在降采样时，如何设置聚合值的标签，例如，9：30-9：35会被标记成9：30还是9：35,默认9：35

```
rng = pd.date_range('20170101', periods = 12)
ts = pd.Series(np.arange(12), index = rng)
print(ts)
```

```
2017-01-01     0
2017-01-02     1
2017-01-03     2
2017-01-04     3
2017-01-05     4
2017-01-06     5
2017-01-07     6
2017-01-08     7
2017-01-09     8
2017-01-10     9
2017-01-11    10
2017-01-12    11
Freq: D, dtype: int32
```

```
# 将序列下采样到5天的数据箱中，并将放入数据箱的时间戳的值相加
ts.resample('5D').sum()
```

```
2017-01-01    10
2017-01-06    35
2017-01-11    21
Freq: 5D, dtype: int32
```

```
#  得到一个新的聚合后的Series，聚合方式为求和
print(ts.resample('5D').mean(),'→ 求平均值\n')
print(ts.resample('5D').max(),'→ 求最大值\n')
print(ts.resample('5D').min(),'→ 求最小值\n')
print(ts.resample('5D').median(),'→ 求中值\n')
print(ts.resample('5D').first(),'→ 返回第一个值\n')
print(ts.resample('5D').last(),'→ 返回最后一个值\n')
print(ts.resample('5D').ohlc(),'→ OHLC重采样\n')
# OHLC:金融领域的时间序列聚合方式 → open开盘、high最大值、low最小值、close收盘
```

```
2017-01-01     2.0
2017-01-06     7.0
2017-01-11    10.5
Freq: 5D, dtype: float64 → 求平均值

2017-01-01     4
2017-01-06     9
2017-01-11    11
```

```
Freq: 5D, dtype: int32 → 求最大值

2017-01-01     0
2017-01-06     5
2017-01-11    10
Freq: 5D, dtype: int32 → 求最小值

2017-01-01     2.0
2017-01-06     7.0
2017-01-11    10.5
Freq: 5D, dtype: float64 → 求中值

2017-01-01     0
2017-01-06     5
2017-01-11    10
Freq: 5D, dtype: int32 → 返回第一个值

2017-01-01     4
2017-01-06     9
2017-01-11    11
Freq: 5D, dtype: int32 → 返回最后一个值
```

```python
# 降采样
rng = pd.date_range('20170101', periods = 12)
ts = pd.Series(np.arange(1,13), index = rng)
print(ts)
```

```
2017-01-01     1
2017-01-02     2
2017-01-03     3
2017-01-04     4
2017-01-05     5
2017-01-06     6
2017-01-07     7
2017-01-08     8
2017-01-09     9
2017-01-10    10
2017-01-11    11
2017-01-12    12
Freq: D, dtype: int32
```

```python
ts.resample('5D').sum()  #'→ 默认\n'
```

```
2017-01-01    15
2017-01-06    40
2017-01-11    23
Freq: 5D, dtype: int32
```

closed：各时间段哪一端是闭合（即包含）的，默认 左闭右闭

- 详解：这里values为0-11，按照5D重采样 → [1,2,3,4,5],[6,7,8,9,10],[11,12]
- left指定间隔左边为结束 → [1,2,3,4,5],[6,7,8,9,10],[11,12]
- right指定间隔右边为结束 → [1],[2,3,4,5,6],[7,8,9,10,11],[12]

```
# 将系列降采样到5天的箱中，但关闭箱间隔的左侧
print(ts.resample('5D', closed = 'left').sum(), '→ left\n')
```

```
2017-01-01    15
2017-01-06    40
2017-01-11    23
Freq: 5D, dtype: int32 → left
```

```
# 将系列降采样到5天的箱中，但关闭箱间隔的右侧
print(ts.resample('5D', closed = 'right').sum(), '→ right\n')
```

```
2016-12-27     1
2017-01-01    20
2017-01-06    45
2017-01-11    12
Freq: 5D, dtype: int32 → right
```

```
print(ts.resample('5D', label = 'left').sum(),'→ leftlabel\n')
# label: 聚合值的index，默认为取左
# 值采样认为默认（这里closed默认）
```

```
print(ts.resample('5D', label = 'right').sum(),'→ rightlabel\n')
```

```
# 升采样及插值

rng = pd.date_range('2017/1/1 0:0:0', periods = 5, freq = 'H')
ts = pd.DataFrame(np.arange(15).reshape(5,3),
```

```
                  index = rng,
                  columns = ['a','b','c'])
print(ts)

print(ts.resample('15T').asfreq())
print(ts.resample('15T').ffill())
print(ts.resample('15T').bfill())
# 低频转高频，主要是如何插值
# .asfreq()：不做填充，返回Nan
# .ffill()：向上填充
# .bfill()：向下填充
```

```
                     a   b   c
2017-01-01 00:00:00  0   1   2
2017-01-01 01:00:00  3   4   5
2017-01-01 02:00:00  6   7   8
2017-01-01 03:00:00  9  10  11
2017-01-01 04:00:00 12  13  14
                       a     b     c
2017-01-01 00:00:00  0.0   1.0   2.0
2017-01-01 00:15:00  NaN   NaN   NaN
2017-01-01 00:30:00  NaN   NaN   NaN
2017-01-01 00:45:00  NaN   NaN   NaN
2017-01-01 01:00:00  3.0   4.0   5.0
2017-01-01 01:15:00  NaN   NaN   NaN
2017-01-01 01:30:00  NaN   NaN   NaN
2017-01-01 01:45:00  NaN   NaN   NaN
2017-01-01 02:00:00  6.0   7.0   8.0
2017-01-01 02:15:00  NaN   NaN   NaN
2017-01-01 02:30:00  NaN   NaN   NaN
2017-01-01 02:45:00  NaN   NaN   NaN
2017-01-01 03:00:00  9.0  10.0  11.0
2017-01-01 03:15:00  NaN   NaN   NaN
2017-01-01 03:30:00  NaN   NaN   NaN
2017-01-01 03:45:00  NaN   NaN   NaN
2017-01-01 04:00:00 12.0  13.0  14.0
                     a   b   c
2017-01-01 00:00:00  0   1   2
2017-01-01 00:15:00  0   1   2
2017-01-01 00:30:00  0   1   2
2017-01-01 00:45:00  0   1   2
2017-01-01 01:00:00  3   4   5
2017-01-01 01:15:00  3   4   5
2017-01-01 01:30:00  3   4   5
2017-01-01 01:45:00  3   4   5
```

```
2017-01-01 02:00:00   6   7   8
2017-01-01 02:15:00   6   7   8
2017-01-01 02:30:00   6   7   8
2017-01-01 02:45:00   6   7   8
2017-01-01 03:00:00   9  10  11
2017-01-01 03:15:00   9  10  11
2017-01-01 03:30:00   9  10  11
2017-01-01 03:45:00   9  10  11
2017-01-01 04:00:00  12  13  14
                      a   b   c
2017-01-01 00:00:00   0   1   2
2017-01-01 00:15:00   3   4   5
2017-01-01 00:30:00   3   4   5
2017-01-01 00:45:00   3   4   5
2017-01-01 01:00:00   3   4   5
2017-01-01 01:15:00   6   7   8
2017-01-01 01:30:00   6   7   8
2017-01-01 01:45:00   6   7   8
2017-01-01 02:00:00   6   7   8
2017-01-01 02:15:00   9  10  11
2017-01-01 02:30:00   9  10  11
2017-01-01 02:45:00   9  10  11
2017-01-01 03:00:00   9  10  11
2017-01-01 03:15:00  12  13  14
2017-01-01 03:30:00  12  13  14
2017-01-01 03:45:00  12  13  14
2017-01-01 04:00:00  12  13  14
```

```

```

```
2016-01     0
2016-02     1
2016-03     2
2016-04     3
2016-05     4
2016-06     5
2016-07     6
2016-08     7
2016-09     8
2016-10     9
2016-11    10
2016-12    11
2017-01    12
Freq: M, dtype: int32
```

```
---------------------------------------------------------------------
---------

IncompatibleFrequency                     Traceback (most recent
call last)

<ipython-input-85-b8f0934a7ca5> in <module>
      5 print(ts)
      6
----> 7 print(ts.resample('3M',kind='period').sum())  # 降采样
      8 print(ts.resample('15D').ffill())  # 升采样
```

```
D:\Anaconda3\lib\site-packages\pandas\core\resample.py in f(self,
_method, min_count, *args, **kwargs)
    924     def f(self, _method=method, min_count=0, *args,
**kwargs):
    925         nv.validate_resampler_func(_method, args, kwargs)
--> 926         return self._downsample(_method,
min_count=min_count)
    927
    928     f.__doc__ = getattr(GroupBy, method).__doc__
```

```
D:\Anaconda3\lib\site-packages\pandas\core\resample.py in
_downsample(self, how, **kwargs)
   1185
   1186         raise IncompatibleFrequency(
-> 1187             f"Frequency {ax.freq} cannot be resampled to
{self.freq}, "
   1188             "as they are not sub or super periods"
   1189         )
```

```
IncompatibleFrequency: Frequency <MonthEnd> cannot be resampled to
<3 * MonthEnds>, as they are not sub or super periods
```

作业

作业1：请输出以下时间序列

```
---时间序列1:-----
2022-01-01    0.123626
2022-01-02    0.932822
2022-01-03    0.251188
2022-01-04    0.079741
2022-01-05    0.328489
Freq: D, dtype: float64
---时间序列2:-----
2022-01-31 00:00:00    0.242894
2022-01-31 00:00:01    0.377202
2022-01-31 00:00:02    0.046502
2022-01-31 00:00:03    0.873964
Freq: S, dtype: float64
---时间序列3:-----
                       value1    value2    value3
2021-12-01 00:00:00  0.748819  0.368575  0.286854
2021-12-01 00:10:00  0.231864  0.828566  0.117412
2021-12-01 00:20:00  0.342888  0.609301  0.296714
2021-12-01 00:30:00  0.936361  0.448391  0.349759
```

```
1
```