数据分析 6 数据规整：聚合、合并和重塑

在许多应用中，数据可能分散在许多文件或数据库中，存储的形式也不利于分析，应采用聚合、合并、重塑数据的方法进行处理。

层次化索引

层次化索引（hierarchical indexing）是pandas的一项重要功能，它使你能在一个轴上拥有多个（两个以上）索引级别。

```
In [9]: data = pd.Series(np.random.randn(9),
   ...:                   index=[['a', 'a', 'a', 'b', 'b', 'c', 'c', 'd', 'd'],
   ...:                          [1, 2, 3, 1, 3, 1, 2, 2, 3]])

In [10]: data
Out[10]:
a  1   -0.204708
   2    0.478943
   3   -0.519439
b  1   -0.555730
   3    1.965781
c  1    1.393406
   2    0.092908
d  2    0.281746
   3    0.769023
dtype: float64
```

```
In [12]: data['b']
Out[12]:
1   -0.555730
3    1.965781
dtype: float64

In [13]: data['b':'c']
Out[13]:
b  1   -0.555730
   3    1.965781
```

```
c  1    1.393406
   2    0.092908
dtype: float64

In [14]: data.loc[['b', 'd']]
Out[14]:
b  1   -0.555730
   3    1.965781
d  2    0.281746
   3    0.769023
dtype: float64
```

“内层”中进行选取

```
In [15]: data.loc[:, 2]
Out[15]:
a    0.478943
c    0.092908
d    0.281746
dtype: float64
```

```
In [16]: data.unstack()
Out[16]:
          1         2         3
a -0.204708  0.478943 -0.519439
b -0.555730       NaN  1.965781
c  1.393406  0.092908       NaN
d       NaN  0.281746  0.769023
```

unstack的逆运算是stack

```
In [17]: data.unstack().stack()
Out[17]:
a  1   -0.204708
   2    0.478943
```

```
   3   -0.519439
b  1   -0.555730
   3    1.965781
c  1    1.393406
   2    0.092908
d  2    0.281746
   3    0.769023
dtype: float64
```

对于一个DataFrame，每条轴都可以有分层索引

```
In [18]: frame = pd.DataFrame(np.arange(12).reshape((4, 3)),
   ....:                      index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],
   ....:                      columns=[['Ohio', 'Ohio', 'Colorado'],
   ....:                               ['Green', 'Red', 'Green']])

In [19]: frame
Out[19]:
     Ohio     Colorado
    Green Red    Green
a 1     0   1        2
  2     3   4        5
b 1     6   7        8
  2     9  10       11
```

```
In [20]: frame.index.names = ['key1', 'key2']

In [21]: frame.columns.names = ['state', 'color']

In [22]: frame
Out[22]:
state      Ohio     Colorado
color     Green Red    Green
key1 key2
a    1        0   1        2
     2        3   4        5
```

```
b    1        6   7        8
     2        9  10       11
```

有了部分列索引，因此可以轻松选取列分组

```
In [23]: frame['Ohio']
Out[23]:
color      Green  Red
key1 key2
a    1         0    1
     2         3    4
b    1         6    7
     2         9   10
```

重排与分级排序

调整某条轴上各级别的顺序

```
In [24]: frame.swaplevel('key1', 'key2')
Out[24]:
state      Ohio     Colorado
color     Green Red    Green
key2 key1
1    a        0   1        2
2    a        3   4        5
1    b        6   7        8
2    b        9  10       11
```

而sort_index则根据单个级别中的值对数据进行排序。交换级别时，常常也会用到sort_index，这样最终结果就是按照指定顺序进行字母排序了

```
In [25]: frame.sort_index(level=1)
Out[25]:
state      Ohio     Colorado
color     Green Red    Green
key1 key2
```

```
a     1         0   1         2
b     1         6   7         8
a     2         3   4         5
b     2         9  10        11

In [26]: frame.swaplevel(0, 1).sort_index(level=0)
Out[26]:
state      Ohio     Colorado
color     Green Red    Green
key2 key1
1    a         0   1         2
     b         6   7         8
2    a         3   4         5
     b         9  10        11
```

根据级别汇总统计

对DataFrame和Series的描述和汇总统计都有一个level选项，它用于指定在某条轴上求和的级别。

```
In [27]: frame.sum(level='key2')
Out[27]:
state  Ohio     Colorado
color Green Red    Green
key2
1         6   8       10
2        12  14       16

In [28]: frame.sum(level='color', axis=1)
Out[28]:
color      Green  Red
key1 key2
a    1         2    1
     2         8    4
b    1        14    7
     2        20   10
```

使用DataFrame的列进行索引

将DataFrame的一个或多个列当做行索引来用，或者可能希望将行索引变成DataFrame的列

```
In [29]: frame = pd.DataFrame({'a': range(7), 'b': range(7, 0, -1),
   ....:                       'c': ['one', 'one', 'one', 'two', 'two',
   ....:                             'two', 'two'],
   ....:                       'd': [0, 1, 2, 0, 1, 2, 3]})

In [30]: frame
Out[30]:
   a  b    c  d
0  0  7  one  0
1  1  6  one  1
2  2  5  one  2
3  3  4  two  0
4  4  3  two  1
5  5  2  two  2
6  6  1  two  3
```

```
In [31]: frame2 = frame.set_index(['c', 'd'])

In [32]: frame2
Out[32]:
       a  b
c   d
one 0  0  7
    1  1  6
    2  2  5
two 0  3  4
    1  4  3
    2  5  2
    3  6  1
```

默认情况下，那些列会从DataFrame中移除，但也可以将其保留下来

```
In [33]: frame.set_index(['c', 'd'], drop=False)
Out[33]:
        a  b    c  d
c    d
one 0   0  7  one  0
    1   1  6  one  1
    2   2  5  one  2
two 0   3  4  two  0
    1   4  3  two  1
    2   5  2  two  2
    3   6  1  two  3
```

reset_index的功能跟set_index刚好相反，层次化索引的级别会被转移到列里面

```
In [34]: frame2.reset_index()
Out[34]:
c  d   a  b
0  one  0  0  7
1  one  1  1  6
2  one  2  2  5
3  two  0  3  4
4  two  1  4  3
5  two  2  5  2
6  two  3  6  1
```

合并数据集

pandas对象中的数据可以通过一些方式进行合并

- pandas.merge可根据一个或多个键将不同DataFrame中的行连接起来。SQL或其他关系型数据库的用户对此应该会比较熟悉，因为它实现的就是数据库的join操作。
- pandas.concat可以沿着一条轴将多个对象堆叠到一起。
- 实例方法combine_first可以将重复数据拼接在一起，用一个对象中的值填充另一个对象中的缺失值

数据库风格的DataFrame合并

数据集的合并（merge）或连接（join）运算是通过一个或多个键将行连接起来的

```
In [35]: df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
   ....:                      'data1': range(7)})

In [36]: df2 = pd.DataFrame({'key': ['a', 'b', 'd'],
   ....:                      'data2': range(3)})

In [37]: df1
Out[37]:
   data1 key
0      0   b
1      1   b
2      2   a
3      3   c
4      4   a
5      5   a
6      6   b

In [38]: df2
Out[38]:
   data2 key
0      0   a
1      1   b
2      2   d
```

这是一种多对一的合并

```
In [39]: pd.merge(df1, df2)
Out[39]:
   data1 key  data2
0      0   b      1
1      1   b      1
2      6   b      1
3      2   a      0
4      4   a      0
```

```
5      5   a      0
```

没有指明要用哪个列进行连接。如果没有指定，merge就会将重叠列的列名当做键。最好明确指定一下

```
In [40]: pd.merge(df1, df2, on='key')
Out[40]:
   data1 key  data2
0      0   b      1
1      1   b      1
2      6   b      1
3      2   a      0
4      4   a      0
5      5   a      0
```

如果两个对象的列名不同，也可以分别进行指定

```
In [41]: df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
   ....:                      'data1': range(7)})

In [42]: df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'],
   ....:                      'data2': range(3)})

In [43]: pd.merge(df3, df4, left_on='lkey', right_on='rkey')
Out[43]:
   data1 lkey  data2 rkey
0      0    b      1    b
1      1    b      1    b
2      6    b      1    b
3      2    a      0    a
4      4    a      0    a
5      5    a      0    a
```

结果里面c和d以及与之相关的数据消失了。默认情况下，merge做的是"内连接"；结果中的键是交集。其他方式还有"left"、"right"以及"outer"。外连接求取的是键的并集，组合了左连接和右连接的效果

```
In [44]: pd.merge(df1, df2, how='outer')
Out[44]:
   data1 key  data2
0    0.0   b    1.0
1    1.0   b    1.0
2    6.0   b    1.0
3    2.0   a    0.0
4    4.0   a    0.0
5    5.0   a    0.0
6    3.0   c    NaN
7    NaN   d    2.0
```

选项	说明
inner	使用两个表都有的键
left	使用左表中所有的键
right	使用右表中所有的键
outer	使用两个表中所有的键

多对多的合并

```
In [45]: df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
   ....:                     'data1': range(6)})

In [46]: df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'],
   ....:                     'data2': range(5)})

In [47]: df1
Out[47]:
   data1 key
```

```
0       0   b
1       1   b
2       2   a
3       3   c
4       4   a
5       5   b

In [48]: df2
Out[48]:
   data2 key
0      0   a
1      1   b
2      2   a
3      3   b
4      4   d

In [49]: pd.merge(df1, df2, on='key', how='left')
Out[49]:
    data1 key  data2
0       0   b    1.0
1       0   b    3.0
2       1   b    1.0
3       1   b    3.0
4       2   a    0.0
5       2   a    2.0
6       3   c    NaN
7       4   a    0.0
8       4   a    2.0
9       5   b    1.0
10      5   b    3.0
```

多对多连接，由于左边的DataFrame有3个"b"行，右边的有2个，所以最终结果中就有6个"b"行

```
In [50]: pd.merge(df1, df2, how='inner')
Out[50]:
   data1 key  data2
0      0   b      1
```

```
1       0   b       3
2       1   b       1
3       1   b       3
4       5   b       1
5       5   b       3
6       2   a       0
7       2   a       2
8       4   a       0
9       4   a       2
```

根据多个键进行合并

```
In [51]: left = pd.DataFrame({'key1': ['foo', 'foo', 'bar'],
   ....:                      'key2': ['one', 'two', 'one'],
   ....:                      'lval': [1, 2, 3]})

In [52]: right = pd.DataFrame({'key1': ['foo', 'foo', 'bar', 'bar'],
   ....:                       'key2': ['one', 'one', 'one', 'two'],
   ....:                       'rval': [4, 5, 6, 7]})

In [53]: pd.merge(left, right, on=['key1', 'key2'], how='outer')
Out[53]:
  key1 key2  lval  rval
0  foo  one   1.0   4.0
1  foo  one   1.0   5.0
2  foo  two   2.0   NaN
3  bar  one   3.0   6.0
4  bar  two   NaN   7.0
```

重复列名的处理

```
In [54]: pd.merge(left, right, on='key1')
Out[54]:
  key1 key2_x  lval key2_y  rval
0  foo    one     1    one     4
1  foo    one     1    one     5
2  foo    two     2    one     4
```

```
3   foo    two     2    one     5
4   bar    one     3    one     6
5   bar    one     3    two     7

In [55]: pd.merge(left, right, on='key1', suffixes=('_left', '_right'))
Out[55]:
  key1 key2_left  lval key2_right  rval
0   foo       one     1        one     4
1   foo       one     1        one     5
2   foo       two     2        one     4
3   foo       two     2        one     5
4   bar       one     3        one     6
5   bar       one     3        two     7
```

索引上的合并

连接键位于其索引中。在这种情况下，你可以传入left_index=True或right_index=True（或两个都传）以说明索引应该被用作连接键

```

In [56]: left1 = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'],
   ....:                        'value': range(6)})

In [57]: right1 = pd.DataFrame({'group_val': [3.5, 7]}, index=['a', 'b'])

In [58]: left1
Out[58]:

  key  value
0   a      0
1   b      1
2   a      2
3   a      3
4   b      4
5   c      5

In [59]: right1
Out[59]:
   group_val
```

```
a        3.5
b        7.0

In [60]: pd.merge(left1, right1, left_on='key', right_index=True)
Out[60]:
  key  value  group_val
0   a      0        3.5
2   a      2        3.5
3   a      3        3.5
1   b      1        7.0
4   b      4        7.0
```

层次化索引的数据，索引的合并默认是多键合并

```
In [62]: lefth = pd.DataFrame({'key1': ['Ohio', 'Ohio', 'Ohio',
   ....:                                'Nevada', 'Nevada'],
   ....:                       'key2': [2000, 2001, 2002, 2001, 2002],
   ....:                       'data': np.arange(5.)})

In [63]: righth = pd.DataFrame(np.arange(12).reshape((6, 2)),
   ....:                       index=[['Nevada', 'Nevada', 'Ohio', 'Ohio',
   ....:                               'Ohio', 'Ohio'],
   ....:                              [2001, 2000, 2000, 2000, 2001, 2002]],
   ....:                       columns=['event1', 'event2'])

In [64]: lefth
Out[64]:
   data    key1  key2
0   0.0    Ohio  2000
1   1.0    Ohio  2001
2   2.0    Ohio  2002
3   3.0  Nevada  2001
4   4.0  Nevada  2002

In [65]: righth
Out[65]:
             event1  event2
Nevada 2001       0       1
```

```
        2000       2       3
Ohio    2000       4       5
        2000       6       7
        2001       8       9
        2002      10      11
```

必须以列表的形式指明用作合并键的多个列（注意用how='outer'对重复索引值的处理）

```
In [66]: pd.merge(lefth, righth, left_on=['key1', 'key2'], right_index=True)
Out[66]:
   data    key1  key2  event1  event2
0   0.0    Ohio  2000       4       5
0   0.0    Ohio  2000       6       7
1   1.0    Ohio  2001       8       9
2   2.0    Ohio  2002      10      11
3   3.0  Nevada  2001       0       1

In [67]: pd.merge(lefth, righth, left_on=['key1', 'key2'],
   ....:          right_index=True, how='outer')
Out[67]:
   data    key1  key2  event1  event2
0   0.0    Ohio  2000     4.0     5.0
0   0.0    Ohio  2000     6.0     7.0
1   1.0    Ohio  2001     8.0     9.0
2   2.0    Ohio  2002    10.0    11.0
3   3.0  Nevada  2001     0.0     1.0
4   4.0  Nevada  2002     NaN     NaN
4   NaN  Nevada  2000     2.0     3.0
```

同时使用合并双方的索引

```
In [68]: left2 = pd.DataFrame([[1., 2.], [3., 4.], [5., 6.]],
   ....:                      index=['a', 'c', 'e'],
   ....:                      columns=['Ohio', 'Nevada'])

In [69]: right2 = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [13, 14]],
   ....:                       index=['b', 'c', 'd', 'e'],
```

```
   ....:                          columns=['Missouri', 'Alabama'])

In [70]: left2
Out[70]:
   Ohio  Nevada
a   1.0     2.0
c   3.0     4.0
e   5.0     6.0

In [71]: right2
Out[71]:
   Missouri  Alabama
b       7.0      8.0
c       9.0     10.0
d      11.0     12.0
e      13.0     14.0

In [72]: pd.merge(left2, right2, how='outer', left_index=True,
right_index=True)
Out[72]:
   Ohio  Nevada  Missouri  Alabama
a   1.0     2.0       NaN      NaN
b   NaN     NaN       7.0      8.0
c   3.0     4.0       9.0     10.0
d   NaN     NaN      11.0     12.0
e   5.0     6.0      13.0     14.0
```

join实例方法，能实现按索引合并

```
In [73]: left2.join(right2, how='outer')
Out[73]:
   Ohio  Nevada  Missouri  Alabama
a   1.0     2.0       NaN      NaN
b   NaN     NaN       7.0      8.0
c   3.0     4.0       9.0     10.0
d   NaN     NaN      11.0     12.0
e   5.0     6.0      13.0     14.0
```

```
In [74]: left1.join(right1, on='key')
Out[74]:
  key  value  group_val
0   a      0        3.5
1   b      1        7.0
2   a      2        3.5
3   a      3        3.5
4   b      4        7.0
5   c      5        NaN
```

向join传入一组DataFrame

```
In [75]: another = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [16., 17.]],
   ....:                         index=['a', 'c', 'e', 'f'],
   ....:                         columns=['New York',
'Oregon'])

In [76]: another
Out[76]:
   New York  Oregon
a       7.0     8.0
c       9.0    10.0
e      11.0    12.0
f      16.0    17.0

In [77]: left2.join([right2, another])
Out[77]:
   Ohio  Nevada  Missouri  Alabama  New York  Oregon
a   1.0     2.0       NaN      NaN       7.0     8.0
c   3.0     4.0       9.0     10.0       9.0    10.0
e   5.0     6.0      13.0     14.0      11.0    12.0

In [78]: left2.join([right2, another], how='outer')
Out[78]:
   Ohio  Nevada  Missouri  Alabama  New York  Oregon
a   1.0     2.0       NaN      NaN       7.0     8.0
```

```
b    NaN    NaN     7.0     8.0    NaN    NaN
c    3.0    4.0     9.0    10.0    9.0   10.0
d    NaN    NaN    11.0    12.0    NaN    NaN
e    5.0    6.0    13.0    14.0   11.0   12.0
f    NaN    NaN     NaN     NaN   16.0   17.0
```

轴向连接

数据合并运算也被称作连接（concatenation）、绑定（binding）或堆叠（stacking）

```
In [79]: arr = np.arange(12).reshape((3, 4))


In [80]: arr
Out[80]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])


In [81]: np.concatenate([arr, arr], axis=1)
Out[81]:
array([[ 0,  1,  2,  3,  0,  1,  2,  3],
       [ 4,  5,  6,  7,  4,  5,  6,  7],
       [ 8,  9, 10, 11,  8,  9, 10, 11]])
```

pandas的concat函数合并操作

```
In [82]: s1 = pd.Series([0, 1], index=['a', 'b'])


In [83]: s2 = pd.Series([2, 3, 4], index=['c', 'd', 'e'])


In [84]: s3 = pd.Series([5, 6], index=['f', 'g'])
```

调用concat可以将值和索引粘合在一起

```
In [85]: pd.concat([s1, s2, s3])
Out[85]:
```

```
a    0
b    1
c    2
d    3
e    4
f    5
g    6
dtype: int64
```

传入axis=1，则结果就会变成一个DataFrame（axis=1是列）

```
In [86]: pd.concat([s1, s2, s3], axis=1)
Out[86]:
     0    1    2
a  0.0  NaN  NaN
b  1.0  NaN  NaN
c  NaN  2.0  NaN
d  NaN  3.0  NaN
e  NaN  4.0  NaN
f  NaN  NaN  5.0
g  NaN  NaN  6.0
```

```
In [87]: s4 = pd.concat([s1, s3])

In [88]: s4
Out[88]:
a    0
b    1
f    5
g    6
dtype: int64

In [89]: pd.concat([s1, s4], axis=1)
Out[89]:
     0  1
a  0.0  0
```

```
b  1.0  1
f  NaN  5
g  NaN  6

In [90]: pd.concat([s1, s4], axis=1, join='inner')
Out[90]:
   0  1
a  0  0
b  1  1
```

```
In [91]: pd.concat([s1, s4], axis=1, join_axes=[['a', 'c', 'b', 'e']])
Out[91]:
     0    1
a  0.0  0.0
c  NaN  NaN
b  1.0  1.0
e  NaN  NaN
```

参与连接的片段在结果中区分不开。假设你想要在连接轴上创建一个层次化索引。使用keys参数即可达到这个目的

```
In [92]: result = pd.concat([s1, s1, s3], keys=['one','two', 'three'])

In [93]: result
Out[93]:
one    a    0
       b    1
two    a    0
       b    1
three  f    5
       g    6
dtype: int64

In [94]: result.unstack()
Out[94]:
         a    b    f    g
```

```
one    0.0  1.0  NaN  NaN
two    0.0  1.0  NaN  NaN
three  NaN  NaN  5.0  6.0
```

如果沿着axis=1对Series进行合并，则keys就会成为DataFrame的列头

```
In [95]: pd.concat([s1, s2, s3], axis=1, keys=['one','two', 'three'])
Out[95]:
   one  two  three
a  0.0  NaN    NaN
b  1.0  NaN    NaN
c  NaN  2.0    NaN
d  NaN  3.0    NaN
e  NaN  4.0    NaN
f  NaN  NaN    5.0
g  NaN  NaN    6.0
```

```
In [96]: df1 = pd.DataFrame(np.arange(6).reshape(3, 2), index=['a', 'b', 'c'],
   ....:                    columns=['one', 'two'])

In [97]: df2 = pd.DataFrame(5 + np.arange(4).reshape(2, 2), index=['a', 'c'],
   ....:                    columns=['three', 'four'])

In [98]: df1
Out[98]:
   one  two
a    0    1
b    2    3
c    4    5

In [99]: df2
Out[99]:
   three  four
a      5     6
c      7     8
```

```
In [100]: pd.concat([df1, df2], axis=1, keys=['level1', 'level2'])
Out[100]:
  level1     level2
     one two  three four
a      0   1    5.0  6.0
b      2   3    NaN  NaN
c      4   5    7.0  8.0
```

```
In [101]: pd.concat({'level1': df1, 'level2': df2}, axis=1)

Out[101]:
  level1     level2
     one two  three four
a      0   1    5.0  6.0
b      2   3    NaN  NaN
c      4   5    7.0  8.0
```

用names参数命名创建的轴级别

```
In [102]: pd.concat([df1, df2], axis=1, keys=['level1', 'level2'],
   .....:            names=['upper', 'lower'])
Out[102]:
upper level1     level2
lower    one two  three four
a          0   1    5.0  6.0
b          2   3    NaN  NaN
c          4   5    7.0  8.0
```

DataFrame的行索引不包含任何相关数据, 传入ignore_index=True

```
In [103]: df1 = pd.DataFrame(np.random.randn(3, 4), columns=['a', 'b', 'c',
'd'])

In [104]: df2 = pd.DataFrame(np.random.randn(2, 3), columns=['b', 'd', 'a'])
```

```
In [105]: df1
Out[105]:
          a         b         c         d
0  1.246435  1.007189 -1.296221  0.274992
1  0.228913  1.352917  0.886429 -2.001637
2 -0.371843  1.669025 -0.438570 -0.539741

In [106]: df2
Out[106]:
          b         d         a
0  0.476985  3.248944 -1.021228
1 -0.577087  0.124121  0.302614
```

```
In [107]: pd.concat([df1, df2], ignore_index=True)
Out[107]:
          a         b         c         d
0  1.246435  1.007189 -1.296221  0.274992
1  0.228913  1.352917  0.886429 -2.001637
2 -0.371843  1.669025 -0.438570 -0.539741
3 -1.021228  0.476985       NaN  3.248944
4  0.302614 -0.577087       NaN  0.124121
```

合并重叠数据

索引全部或部分重叠的两个数据集

```
In [108]: a = pd.Series([np.nan, 2.5, np.nan, 3.5, 4.5, np.nan],
   .....:               index=['f', 'e', 'd', 'c', 'b', 'a'])

In [109]: b = pd.Series(np.arange(len(a), dtype=np.float64),
   .....:               index=['f', 'e', 'd', 'c', 'b', 'a'])

In [110]: b[-1] = np.nan

In [111]: a
Out[111]:
```

```
f    NaN
e    2.5
d    NaN
c    3.5
b    4.5
a    NaN
dtype: float64

In [112]: b
Out[112]:
f    0.0
e    1.0
d    2.0
c    3.0
b    4.0
a    NaN
dtype: float64

In [113]: np.where(pd.isnull(a), b, a)
Out[113]: array([ 0. ,  2.5,  2. ,  3.5,  4.5,  nan])
```

此语句实现一样的功能

```
In [114]: b[:-2].combine_first(a[2:])
Out[114]:
a    NaN
b    4.5
c    3.0
d    2.0
e    1.0
f    0.0
dtype: float64
```

对于DataFrame，combine_first自然也会在列上做同样的事情，因此你可以将其看做：用传递对象中的数据为调用对象的缺失数据“打补丁”

```
In [115]: df1 = pd.DataFrame({'a': [1., np.nan, 5., np.nan],
```

```
   .....:                     'b': [np.nan, 2., np.nan, 6.],
   .....:                     'c': range(2, 18, 4)})

In [116]: df2 = pd.DataFrame({'a': [5., 4., np.nan, 3., 7.],
   .....:                     'b': [np.nan, 3., 4., 6., 8.]})

In [117]: df1
Out[117]:
     a    b   c
0  1.0  NaN   2
1  NaN  2.0   6
2  5.0  NaN  10
3  NaN  6.0  14

In [118]: df2
Out[118]:
     a    b
0  5.0  NaN
1  4.0  3.0
2  NaN  4.0
3  3.0  6.0
4  7.0  8.0

In [119]: df1.combine_first(df2)
Out[119]:
     a    b     c
0  1.0  NaN   2.0
1  4.0  2.0   6.0
2  5.0  4.0  10.0
3  3.0  6.0  14.0
4  7.0  8.0   NaN
```

重塑和轴向旋转

用于重新排列表格型数据的基础运算。这些函数也称作重塑（reshape）或轴向旋转（pivot）运算

重塑层次化索引

- stack：将数据的列“旋转”为行
- unstack：将数据的行“旋转”为列

```
In [120]: data = pd.DataFrame(np.arange(6).reshape((2, 3)),
   .....:                      index=pd.Index(['Ohio','Colorado'],
name='state'),
   .....:                      columns=pd.Index(['one', 'two', 'three'],
   .....:                      name='number'))

In [121]: data
Out[121]:
number    one  two  three
state
Ohio        0    1      2
Colorado    3    4      5
```

对该数据使用stack方法即可将列转换为行，得到一个Series

```
In [122]: result = data.stack()

In [123]: result
Out[123]:
state     number
Ohio      one       0
          two       1
          three     2
Colorado  one       3
          two       4
          three     5
dtype: int64
```

对于一个层次化索引的Series，你可以用unstack将其重排为一个DataFrame：

```
In [124]: result.unstack()
Out[124]:
number    one  two  three
```

```
state
Ohio        0    1      2
Colorado    3    4      5
```

默认情况下，unstack操作的是最内层（stack也是如此）。传入分层级别的编号或名称即可对其它级别进行unstack操作

```
In [125]: result.unstack(0)
Out[125]:
state   Ohio  Colorado
number
one        0         3
two        1         4
three      2         5

In [126]: result.unstack('state')
Out[126]:
state   Ohio  Colorado
number
one        0         3
two        1         4
three      2         5
```

将“长格式”旋转为“宽格式”

多个时间序列数据通常是以所谓的“长格式”（long）或“堆叠格式”（stacked）存储在数据库和CSV中的。我们先加载一些示例数据，做一些时间序列规整和数据清洗

```
In [139]: data = pd.read_csv('examples/macrodata.csv')

In [140]: data.head()
Out[140]:
     year  quarter   realgdp  realcons  realinv  realgovt  realdpi    cpi  \
0  1959.0      1.0  2710.349    1707.4  286.898   470.045   1886.9  28.98
1  1959.0      2.0  2778.801    1733.7  310.859   481.301   1919.7  29.15
2  1959.0      3.0  2775.488    1751.8  289.226   491.260   1916.4  29.35
```

```
3  1959.0      4.0  2785.204    1753.7  299.356   484.052   1931.3  29.37
4  1960.0      1.0  2847.699    1770.5  331.722   462.199   1955.5  29.54
      m1  tbilrate  unemp      pop  infl  realint
0  139.7      2.82    5.8  177.146  0.00     0.00
1  141.7      3.08    5.1  177.830  2.34     0.74
2  140.5      3.82    5.3  178.657  2.74     1.09
3  140.0      4.33    5.6  179.386  0.27     4.06
4  139.6      3.50    5.2  180.007  2.31     1.19

In [141]: periods = pd.PeriodIndex(year=data.year, quarter=data.quarter,
   .....:                          name='date')

In [142]: columns = pd.Index(['realgdp', 'infl', 'unemp'], name='item')

In [143]: data = data.reindex(columns=columns)

In [144]: data.index = periods.to_timestamp('D', 'end')

In [145]: ldata = data.stack().reset_index().rename(columns={0: 'value'})
```

不同的item值分别形成一列，date列中的时间戳则用作索引

```
# 前两个传递的值分别用作行和列索引，最后一个可选值则是用于填充DataFrame的数据列
In [147]: pivoted = ldata.pivot('date', 'item', 'value')

In [148]: pivoted
Out[148]:
item        infl    realgdp  unemp
date
1959-03-31  0.00   2710.349    5.8
1959-06-30  2.34   2778.801    5.1
1959-09-30  2.74   2775.488    5.3
1959-12-31  0.27   2785.204    5.6
1960-03-31  2.31   2847.699    5.2
1960-06-30  0.14   2834.390    5.2
1960-09-30  2.70   2839.022    5.6
1960-12-31  1.21   2802.616    6.3
1961-03-31 -0.40   2819.264    6.8
```

如果忽略最后一个参数，得到的DataFrame就会带有层次化的列

```
1961-06-30  1.47   2872.005    7.0
...          ...        ...    ...
2007-06-30  2.75  13203.977    4.5
2007-09-30  3.45  13321.109    4.7
2007-12-31  6.38  13391.249    4.8
2008-03-31  2.82  13366.865    4.9
2008-06-30  8.53  13415.266    5.4
2008-09-30 -3.16  13324.600    6.0
2008-12-31 -8.79  13141.920    6.9
2009-03-31  0.94  12925.410    8.1
2009-06-30  3.37  12901.504    9.2
2009-09-30  3.56  12990.341    9.6
[203 rows x 3 columns]
```

```
In [149]: ldata['value2'] = np.random.randn(len(ldata))

In [150]: ldata[:10]
Out[150]:
        date     item     value    value2
0 1959-03-31  realgdp  2710.349  0.523772
1 1959-03-31     infl     0.000  0.000940
2 1959-03-31    unemp     5.800  1.343810
3 1959-06-30  realgdp  2778.801 -0.713544
4 1959-06-30     infl     2.340 -0.831154
5 1959-06-30    unemp     5.100 -2.370232
6 1959-09-30  realgdp  2775.488 -1.860761
7 1959-09-30     infl     2.740 -0.860757
8 1959-09-30    unemp     5.300  0.560145
9 1959-12-31  realgdp  2785.204 -1.265934
```

如果忽略最后一个参数，得到的DataFrame就会带有层次化的列

```
In [151]: pivoted = ldata.pivot('date', 'item')

In [152]: pivoted[:5]
```

```
Out[152]:
           value                        value2
item        infl   realgdp unemp      infl   realgdp     unemp
date
1959-03-31  0.00  2710.349   5.8  0.000940  0.523772  1.343810
1959-06-30  2.34  2778.801   5.1 -0.831154 -0.713544 -2.370232
1959-09-30  2.74  2775.488   5.3 -0.860757 -1.860761  0.560145
1959-12-31  0.27  2785.204   5.6  0.119827 -1.265934 -1.063512
1960-03-31  2.31  2847.699   5.2 -2.359419  0.332883 -0.199543

In [153]: pivoted['value'][:5]
Out[153]:
item        infl   realgdp  unemp
date
1959-03-31  0.00  2710.349    5.8
1959-06-30  2.34  2778.801    5.1
1959-09-30  2.74  2775.488    5.3
1959-12-31  0.27  2785.204    5.6
1960-03-31  2.31  2847.699    5.2
```

将“宽格式”旋转为“长格式”

```
In [157]: df = pd.DataFrame({'key': ['foo', 'bar', 'baz'],
   .....:                    'A': [1, 2, 3],
   .....:                    'B': [4, 5, 6],
   .....:                    'C': [7, 8, 9]})

In [158]: df
Out[158]:
   A  B  C  key
0  1  4  7  foo
1  2  5  8  bar
2  3  6  9  baz
```

当使用pandas.melt，我们必须指明哪些列是分组指标。下面使用key作为唯一的分组指标

```
In [159]: melted = pd.melt(df, ['key'])

In [160]: melted
Out[160]:
   key variable  value
0  foo        A      1
1  bar        A      2
2  baz        A      3
3  foo        B      4
4  bar        B      5
5  baz        B      6
6  foo        C      7
7  bar        C      8
8  baz        C      9
```

使用pivot，可以重塑回原来的样子

```
In [161]: reshaped = melted.pivot('key', 'variable', 'value')

In [162]: reshaped
Out[162]:
variable  A  B  C
key
bar       2  5  8
baz       3  6  9
foo       1  4  7
```

因为pivot的结果从列创建了一个索引，用作行标签，我们可以使用reset_index将数据移回列

```
In [163]: reshaped.reset_index()
Out[163]:
variable  key  A  B  C
0         bar  2  5  8
1         baz  3  6  9
2         foo  1  4  7
```

指定列的子集，作为值的列

```
In [164]: pd.melt(df, id_vars=['key'], value_vars=['A', 'B'])
Out[164]:
   key variable  value
0  foo        A      1
1  bar        A      2
2  baz        A      3
3  foo        B      4
4  bar        B      5
5  baz        B      6
```

pandas.melt也可以不用分组指标

```
In [165]: pd.melt(df, value_vars=['A', 'B', 'C'])
Out[165]:
  variable  value
0        A      1
1        A      2
2        A      3
3        B      4
4        B      5
5        B      6
6        C      7
7        C      8
8        C      9

In [166]: pd.melt(df, value_vars=['key', 'A', 'B'])
Out[166]:
  variable value
0      key   foo
1      key   bar
2      key   baz
3        A     1
4        A     2
5        A     3
6        B     4
7        B     5
```

8	B	6