2--Pandas基础

1. 学习目标

1. Pandas的数据结构
2. Pandas的操作工具

2. Pandas 概况

pandas的功能：使数据分析工作变得简单．

Pandas的特点:

1. 基于NumPy构建
2. 始建于2008年
3. 按轴自动数据对齐
4. 集成了时间序列功能
5. 包含：能处理时间序列数据，也能处理非时间序列数据的数据结构
6. 灵活处理缺失数据

3. Pandas的导入

```
from pandas import Series, DataFrame
import pandas as pd
```

说明: 因为Series 和DataFrame用得很多，所以把它们引入本地命名空间中，更方便．

4. Pandas的数据结构

Pandas有两个最主要的数据结构: Series和DataFrame.

4.1 Series的创建

Series由一组数据以及一组与之相关的数据标签(索引)组成. 这里的数据，是NumPy数据类型． 建立Series的最简单方法就是：将数组放入Series()中．

```
>>> from pandas import Series, DataFrame
>>> import pandas
>>> obj = Series([1,1,2,3,5])
>>> obj
0    1
1    1
2    2
3    3
4    5
dtype: int64
```

Series的表现形式有两种：**字符串表现形式**，和**数组表示形式**．字符串以两列展开：左边一列是索引，右边一列是值．　用Series的**values属性**获取其数组表现形式，　用**index属性**来获取索引对象．举例:

```
dtype: int64
>>> obj.values
array([1, 1, 2, 3, 5])
>>> obj.index
RangeIndex(start=0, stop=5, step=1)
```

第二种方法：我们可以为数组中的每一个元素自建索引．

```
>>> obj = Series([1,1,2,3,5],index=['a','b','c','d','f'])
>>> obj
a    1
b    1
c    2
d    3
f    5
dtype: int64
>>> obj.index
Index(['a', 'b', 'c', 'd', 'f'], dtype='object')
```

我们可以做NumPy数组运算．包括：用布尔型索引进行过滤；标量乘法；应用数学函数等等．

```
>>> obj.index
Index(['a', 'b', 'c', 'd', 'f'], dtype='object')
>>> obj[obj<2]
a    1
b    1
dtype: int64
>>> obj **2
a     1
b     1
c     4
d     9
f    25
```

```
dtype: int64
>>> np.sin(obj)
a    0.841471
b    0.841471
c    0.909297
d    0.141120
f   -0.958924
dtype: float64
>>> obj*3
a     3
b     3
c     6
d     9
f    15
dtype: int64
```

可以把Series看成定长的有序字典．这就是说：

1. 可以把Series用在原本需要字典的函数中．
2. 可以**通过字典来创建Series**.

```
>>> data = {"Chengdu": 3000, "Shanghai": 3500, "Chongqing": 2000}
>>> obj2 = Series(data)
>>> obj2
Chengdu      3000
Chongqing    2000
Shanghai     3500
dtype: int64
>>> 'Chengdu' in obj2
True
>>> 'Lasha' in obj2
False
>>> cities = ['Guangzhou','Fuzhou','Shanghai','Chongqing']
>>> obj3 = Series(data, index = cities)
>>> obj3
Guangzhou       NaN
Fuzhou          NaN
Shanghai     3500.0
Chongqing    2000.0
dtype: float64
```

上例为城市**总产值**的一个字典data． 用来生成了一个Series. 字典的键就变成Series的索引．

接下来，我们用data建Series,同时用cities列表来做索引. data中与cities索引匹配的值只有两个: 'Shanghai', 'Chongqing'. Guangzhou和'Fuzhou'对应的data值找不到，所以你看到NaN（Not a Number,在Pandas中表示数据缺失）.

检查数据是否缺失的函数:

Pandas方法：

pd.isnull()

pd.notnull()

实例方法:

obj.isnull()

obj.notnull()

```
>>> obj3
Guangzhou       NaN
Fuzhou          NaN
Shanghai     3500.0
Chongqing    2000.0
dtype: float64
>>> pd.isnull(obj3)
Guangzhou     True
Fuzhou        True
Shanghai     False
Chongqing    False
dtype: bool
>>> obj3.isnull()
Guangzhou     True
Fuzhou        True
Shanghai     False
Chongqing    False
dtype: bool
```

4.2 Series的重要功能：自动对齐不同索引的数据

```
>>> obj2 = Series(data)
>>> obj2
Chengdu      3000
Chongqing    2000
Shanghai     3500
dtype: int64
>>> data3 = {"Chengdu": 3000, "Shanghai": 3500, "Xiamen":2800, "Taipei":2900}
>>> obj3 = Series(data3)
>>> obj3
Chengdu     3000
Shanghai    3500
Taipei      2900
Xiamen      2800
dtype: int64
>>> obj2 + obj3
Chengdu      6000.0
Chongqing       NaN
Shanghai     7000.0
Taipei          NaN
Xiamen          NaN
```

```
dtype: float64
```

name属性

Series本身的name属性．　Series的索引的name属性．

```
>>> obj2.name = 'income'
>>> obj2.index.name = 'city'
>>> obj2
city
Chengdu      3000
Chongqing    2000
Shanghai     3500
Name: income, dtype: int64
```

Series的索引可以通过赋值就地修改:

```
>>> obj4
Chengdu      6000.0
Chongqing       NaN
Shanghai     7000.0
Taipei          NaN
Xiamen          NaN
dtype: float64
>>> obj4.index = ['CTD','CQ','SH','TP','SM']
>>> obj4
CTD    6000.0
CQ        NaN
SH     7000.0
TP        NaN
SM        NaN
dtype: float64
```

4.3 DataFrame介绍

DataFrame由一组有序的列组成，每列可以是不同的值类型(数值，布尔型，字符串).

DataFrame中的行与列是平行的．

DataFrame中的数据是以一个或多个二维块存放的．

4.4 创建DF的方法

直接方法是：传入一个由等长列表组成的字典．

或者传入一个由NumPy数组组成的字典．

```
# 字符串型值
>>> data2 = {'city': ['Chengdu','Chongqing','Fuzhou','Xiamen'],'income':
['3000','2800','2600','4000']  }
>>> data2
{'city': ['Chengdu', 'Chongqing', 'Fuzhou', 'Xiamen'], 'income': ['3000', '2800',
'2600', '4000']}
>>> frame = DataFrame(data2)
>>> frame
        city income
0    Chengdu   3000
1  Chongqing   2800
2     Fuzhou   2600
3     Xiamen   4000
>>> data3 = {'city': ['Xian','Chongqing','Fuzhou','Xiamen'],'income':
['3000','2800','2600','4000']  }>>> frame2 = DataFrame(data3)
>>> frame + frame2
                 city    income
0         ChengduXian  30003000
1  ChongqingChongqing  28002800
2        FuzhouFuzhou  26002600
3        XiamenXiamen  40004000

# 数值型值
>>> data3 = {'city': ['Xian','Chongqing','Fuzhou','Xiamen'],'income':
[3000,2800,2600,4000]  }
>>> data2 = {'city': ['Chengdu','Chongqing','Fuzhou','Xiamen'],'income':
[3200,2800,2600,4000]  }>>> frame3 = DataFrame(data3)
>>> frame2 = DataFrame(data2)
>>> frame2+frame3
                 city  income
0         ChengduXian    6200
1  ChongqingChongqing    5600
2        FuzhouFuzhou    5200
3        XiamenXiamen    8000

# 可以通过  设定colums列表的值  来指定列之顺序----对各列做排序
>>> DataFrame(data2, columns = ['income','city'])
   income       city
0    3200    Chengdu
1    2800  Chongqing
2    2600     Fuzhou
3    4000     Xiamen

# 如果传入的列，在数据中找不到，就会显示NA值
>>> DataFrame(data2, columns = ['income','city', 'pop'])
   income       city  pop
0    3200    Chengdu  NaN
1    2800  Chongqing  NaN
2    2600     Fuzhou  NaN
3    4000     Xiamen  NaN
```

可以从DataFrame中提取一列(一个Series). 方式如下：

```
>>> DataFrame(data2, columns = ['income','city'])['income']
0    3200
1    2800
2    2600
3    4000
Name: income, dtype: int64
>>> frame2['income']
0    3200
1    2800
2    2600
3    4000
Name: income, dtype: int64

>>> frame2.ix[0]
city      Chengdu
income       3200
Name: 0, dtype: object
>>> frame2.ix[:]
        city  income
0    Chengdu    3200
1  Chongqing    2800
2     Fuzhou    2600
3     Xiamen    4000
>>> frame2.ix[:,1]
0    3200
1    2800
2    2600
3    4000
Name: income, dtype: int64
```

可以为DataFrame的列赋值：

```
>>> frame4 = DataFrame(data2, columns = ['income','city', 'pop'])
>>> frame4
   income       city  pop
0    3200    Chengdu  NaN
1    2800  Chongqing  NaN
2    2600     Fuzhou  NaN
3    4000     Xiamen  NaN
>>> frame4['pop'] = 2000.
>>> frame4
   income       city     pop
0    3200    Chengdu  2000.0
1    2800  Chongqing  2000.0
2    2600     Fuzhou  2000.0
3    4000     Xiamen  2000.0
```

将嵌套字典传入pd.DataFrame(),是什么效果？

```
>>> gdp = {'Chengdu':{'2000':30,'2005':45,'2010':69},'Chongqing':
```

```
{'2000':25,'2005':35,'2010':60} ,'Fuzhou': {'2000':30,'2005':42,'2010':60},'Xiamen':
{'2000':39,'2005':43,'2010':55} }
>>> frame5 = DataFrame(gdp)
>>> frame5
      Chengdu  Chongqing  Fuzhou  Xiamen
2000       30         25      30      39
2005       45         35      42      43
2010       69         60      60      55
>>> frame5.T
           2000  2005  2010
Chengdu      30    45    69
Chongqing    25    35    60
Fuzhou       30    42    60
Xiamen       39    43    55
```

外层字典的键作为列，内层字典的键作为行．

注意：你可以将DataFrame转置！

DataFrame的values属性，以二维ndarray形式返回DF中的数据．

```
>>> frame5.values
array([[30, 25, 30, 39],
       [45, 35, 42, 43],
       [69, 60, 60, 55]])
```

如果各列的数据类型不同，DataFrame的values属性会返回能兼容所有列的数据类型．

```
>>> frame4
   income       city     pop
0    3200    Chengdu  2000.0
1    2800  Chongqing  2000.0
2    2600     Fuzhou  2000.0
3    4000     Xiamen  2000.0
>>> frame4.values
array([[3200, 'Chengdu', 2000.0],
       [2800, 'Chongqing', 2000.0],
       [2600, 'Fuzhou', 2000.0],
       [4000, 'Xiamen', 2000.0]], dtype=object)
```

DataFrame的Index和columns有name属性．　我们可以去设置．这些信息会被显示出来．

```
>>> frame4.index.name = 'info'; frame4.columns.name = 'No.'
>>> frame4.index.name
'info'
>>> frame4
No.   income       city     pop
info
0       3200    Chengdu  2000.0
1       2800  Chongqing  2000.0
2       2600     Fuzhou  2000.0
3       4000     Xiamen  2000.0
```

在DataFrame中，可以用info()查看数据类型和统计．

```
>>> frame5.info()
<class 'pandas.core.frame.DataFrame'>
Index: 3 entries, 2000 to 2010
Data columns (total 4 columns):
Chengdu      3 non-null int64
Chongqing    3 non-null int64
Fuzhou       3 non-null int64
Xiamen       3 non-null int64
dtypes: int64(4)
memory usage: 120.0+ bytes
```

对于pandas来说，成千上万行，几百兆的数据，不是问题．

与NumPy一样，DataFrame的数据类型变更用astype()函数．

```
>>> frame5.Chongqing.astype('str')
2000    25
2005    35
2010    60
Name: Chongqing, dtype: object
```

4.5 索引对象

特征：

1. 索引对象不可修改．
2. 每一个索引都有一些方法和属性

Index的方法和属性

方法	作用
append	连接另一个Index对象，产生一个新的Index
diff	计算差集，得到一个Index
intersection	计算交集
union	计算并集

5 Series和DataFrame基本功能

5.1 重新索引

pandas对象的一个重要方法： reindex, 作用是创建一个适应新索引的新对象． 例：

```
>>> obj
a    1
b    1
c    2
d    3
f    5
dtype: int64
>>> obj.reindex(['f','c','b','a','d'])
f    5
c    2
b    1
a    1
d    3
dtype: int64
>>> obj.reindex(['f','c','b','a','forth'])
f        5.0
c        2.0
b        1.0
a        1.0
forth    NaN
dtype: float64
```

```
>>> obj5 = Series(['blue','yellow', 'red'], index = [0,2,4])
>>> obj5
0      blue
2    yellow
4       red
dtype: object
>>> obj5.reindex(range(6), method='ffill')
0      blue
1      blue
2    yellow

3    yellow
```

```
4       red
5       red
dtype: object
>>> obj5.reindex(range(6), method='bfill')
0      blue
1    yellow
2    yellow
3       red
4       red
5       NaN
dtype: object
```

总结:

1. 调用该Series的reindex将根据新索引进行**重排**．如果某个索引值不存在，则引入缺失值(NA).
2. reindex函数可用的参数：　index , mothod (插值的方式：　bfill, ffill), fill_value (填充 / 替代值)

利用ix的标签索引功能，完成**重新索引**．

```
>>> obj.ix[['a','b','d','f','h']]
     0
a  1.0
b  1.0
d  3.0
f  5.0
h  NaN
>>> obj.ix[['a','b','d','f','h'],[0]]
     0
a  1.0
b  1.0
d  3.0
f  5.0
h  NaN
```

ix[[],[]]：

ix[]中的第一个[]填行之索引名称．此时，你可以任意设置其顺序．

5.2 丢弃某个轴上的项

目的：　生成删除了指定值的新对象

方法：　obj.drop(列表)

```
>>> import numpy as np
>>> obj = Series(np.arange(5), index = ['a','b','c','d','e'])

>>> obj
```

```
a    0
b    1
c    2
d    3
e    4
dtype: int64
>>> obj.drop(['c','d'])
a    0
b    1
e    4
dtype: int64

>>> obj.drop('c')
a    0
b    1
d    3
e    4
dtype: int64
```

对于DataFrame,我们也可以删除任意轴上的索引值 .

```
>>> data = DataFrame(np.arange(16).reshape((4,4)), index = ['CD','CQ', 'GZ', 'XM'],
columns = ['One', 'Two', 'Three', 'Four'] )
>>> data
    One  Two  Three  Four
CD    0    1      2     3
CQ    4    5      6     7
GZ    8    9     10    11
XM   12   13     14    15
>>> data.drop(['GZ','XM'])
    One  Two  Three  Four
CD    0    1      2     3
CQ    4    5      6     7
>>> data.drop('One')
Traceback (most recent call last):......
ValueError: labels ['One'] not contained in axis
>>> data.drop('One', axis=1)
    Two  Three  Four
CD    1      2     3
CQ    5      6     7
GZ    9     10    11
XM   13     14    15
>>> data.drop(['One','Two'], axis=1)
    Three  Four
CD      2     3
CQ      6     7
GZ     10    11
XM     14    15
```

5.3 索引与过滤

Series的索引与NumPy数组类似．但是差别在于：　Series的索引不只是整数．

```
>>> obj
a    0
b    1
c    2
d    3
e    4
dtype: int64
>>> obj['e']
4
>>> obj[2:4]
c    2
d    3
dtype: int64
>>> obj[[1,4]]
b    1
e    4
dtype: int64
>>> obj[[1,4]][obj< 3]
b    1
dtype: int64
```

利用标签的切片运算与普通Python切片运算不同之处：前者是一个封闭的区间．

```
>>> obj['b':'e']
b    1
c    2
d    3
e    4
dtype: int64
```

注意：　结果包括标签e所在的行．

赋值也是类似的道理．

```
>>> obj['b':'e'] = 100
>>> obj
a      0
b    100
c    100
d    100
e    100
dtype: int64
>>> obj['b':'e'] = 0
>>> obj
a    0
b    0
c    0
```

```
d    0
e    0
dtype: int64
>>> obj[1:4] = 100
>>> obj
a      0
b    100
c    100
d    100
e      0
dtype: int64
```

对DataFrame做索引： 获取一列或者多列 .

```
>>> data['Four']
CD     3
CQ     7
GZ    11
XM    15
Name: Four, dtype: int64
>>> data[['Four','One']]
    Four  One
CD     3    0
CQ     7    4
GZ    11    8
XM    15   12
>>> data[['One','Four']]
    One  Four
CD    0     3
CQ    4     7
GZ    8    11
XM   12    15
```

选取行的方法：

```
>>> data[1:3]
    One  Two  Three  Four
CQ    4    5      6     7
GZ    8    9     10    11
>>> data[1:]
    One  Two  Three  Four
CQ    4    5      6     7
GZ    8    9     10    11
XM   12   13     14    15
>>> data[:4]
    One  Two  Three  Four
CD    0    1      2     3
CQ    4    5      6     7
GZ    8    9     10    11
XM   12   13     14    15
```

```
>>> data[:3]
    One  Two  Three  Four
CD    0    1      2     3
CQ    4    5      6     7
GZ    8    9     10    11
>>> data['Four'] > 10
CD    False
CQ    False
GZ     True
XM     True
Name: Four, dtype: bool
>>> data[data['Four'] > 10]
    One  Two  Three  Four
GZ    8    9     10    11
XM   12   13     14    15
```

我们还可以通过**布尔型DataFrame**来做索引：

```
>>> data >10
      One    Two  Three   Four
CD  False  False  False  False
CQ  False  False  False  False
GZ  False  False  False   True
XM   True   True   True   True
>>> data < 10
      One    Two  Three   Four
CD   True   True   True   True
CQ   True   True   True   True
GZ   True   True  False  False
XM  False  False  False  False
>>> data[data<10] = 0
>>> data
    One  Two  Three  Four
CD    0    0      0     0
CQ    0    0      0     0
GZ    0    0     10    11
XM   12   13     14    15
```

问： 想在**行**上也做**标签索引**，我们该怎么做？

答案： **专门的索引字段ix**.

```
>>> data['CD',['Three','Four']]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/huang/anaconda3/lib/python3.6/site-packages/pandas/core/frame.py", line
2139, in __getitem__
    return self._getitem_column(key)
  File "/home/huang/anaconda3/lib/python3.6/site-packages/pandas/core/frame.py", line
2146, in _getitem_column
    return self._get_item_cache(key)
  File "/home/huang/anaconda3/lib/python3.6/site-packages/pandas/core/generic.py", line
```

```
1840, in _get_item_cache
    res = cache.get(item)
TypeError: unhashable type: 'list'
>>> data.ix['CD',['Three','Four']]
Three    2
Four     3
Name: CD, dtype: int64
>>> data.ix[['CD','XM'],['Three','Four']]
    Three  Four
CD      2     3
XM     14    15
```

5.4 算术运算与数据对齐

Series: 两个 S eries相加返回一个新的Series, 在不重叠的索引处引入缺失值 . 索引是原来两个索引的并集 . (数据对齐)

DataFrame: 同时对行和列做对齐操作 .

```
>>> s = Series([3.0, 5.9, 8.9, 12],index=['a','b','c','d'])
>>> s2 = Series([-3.0, 4, 9, -2, 10],index=['a','c','d','e','f'])
>>> s
a     3.0
b     5.9
c     8.9
d    12.0
dtype: float64
>>> s2
a    -3.0
c     4.0
d     9.0
e    -2.0
f    10.0
dtype: float64
>>> s + s2
a     0.0
b     NaN
c    12.9
d    21.0
e     NaN
f     NaN
dtype: float64
>>> data1 = DataFrame(np.arange(16).reshape(4,4),index=['CD','CQ','GZ','XM'],
columns=list('abcd'))
>>> data2 = DataFrame(np.arange(16).reshape(4,4),index=['CD','CQ','SZ','XM'],
columns=list('abcf'))
>>> data1
     a   b   c   d
CD   0   1   2   3
CQ   4   5   6   7
GZ   8   9  10  11
```

```
XM  12  13  14  15
>>> data2
     a   b   c   f
CD   0   1   2   3
CQ   4   5   6   7
SZ   8   9  10  11
XM  12  13  14  15
>>> data1 + data2
       a     b     c   d   f
CD   0.0   2.0   4.0 NaN NaN
CQ   8.0  10.0  12.0 NaN NaN
GZ   NaN   NaN   NaN NaN NaN
SZ   NaN   NaN   NaN NaN NaN
XM  24.0  26.0  28.0 NaN NaN
```

```
>>> data1.add(data2, fill_value = 0)
       a     b     c     d     f
CD   0.0   2.0   4.0   3.0   3.0
CQ   8.0  10.0  12.0   7.0   7.0
GZ   8.0   9.0  10.0  11.0   NaN
SZ   8.0   9.0  10.0   NaN  11.0
XM  24.0  26.0  28.0  15.0  15.0
```

df里面的函数add()是执行比较通用的相加运算．

其他运算函数：

add (+)

sub (-)

div (/)

mul (*)

可以把DataFrame 与Series相加:

1. 默认情况下，Series的索引，　对应的是DataFrame的列标签．可以直接用加减符号运算(+,-).
2. 如过你想要Series的索引对应DataFrame的行标签(匹配DataFrame的行，且在列上广播)，必须用算术运算方法.

```
>>> data1
     a   b   c   d
CD   0   1   2   3
CQ   4   5   6   7
GZ   8   9  10  11
XM  12  13  14  15
>>> s
```

```
a     3.0
b     5.9
c     8.9
d    12.0
dtype: float64
>>> data1 + s
       a     b     c     d
CD   3.0   6.9  10.9  15.0
CQ   7.0  10.9  14.9  19.0
GZ  11.0  14.9  18.9  23.0
XM  15.0  18.9  22.9  27.0
>>> data.T
       CD  CQ  GZ  XM
One     0   4   8  12
Two     1   5   9  13
Three   2   6  10  14
Four    3   7  11  15
>>> data.T + s
       CD  CQ  GZ  XM   a   b   c   d
One   NaN NaN NaN NaN NaN NaN NaN NaN
Two   NaN NaN NaN NaN NaN NaN NaN NaN
Three NaN NaN NaN NaN NaN NaN NaN NaN
Four  NaN NaN NaN NaN NaN NaN NaN NaN
```

5.5 排序和排名

sort_index(): 用于对行索引或列索引做排序 .

```
>>> obj = Series(range(5), index=['z','b','a','c','f'])
>>> obj
z    0
b    1
a    2
c    3
f    4
dtype: int64
>>> obj.sort_index()
a    2
b    1
c    3
f    4
z    0
dtype: int64
>>> data
    One  Two  Three  Four
CD    0    1      2     3
CQ    4    5      6     7
GZ    8    9     10    11
XM   12   13     14    15
>>> data.sort_index()
    One  Two  Three  Four
CD    0    1      2     3
CQ    4    5      6     7
```

```
GZ    8    9     10    11
XM   12   13     14    15
>>> data.sort_index(axis=1)
    Four  One  Three  Two
CD     3    0      2    1
CQ     7    4      6    5
GZ    11    8     10    9
XM    15   12     14   13
>>> data.sort_index(axis=1, ascending=False)
    Two  Three  One  Four
CD    1      2    0     3
CQ    5      6    4     7
GZ    9     10    8    11
XM   13     14   12    15
>>> obj.sort_index()
a    2
b    1
c    3
f    4
z    0
dtype: int64
>>> obj2 = obj.sort_index()
>>> obj2
a    2
b    1
c    3
f    4
z    0
dtype: int64
>>> obj2.order()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/huang/anaconda3/lib/python3.6/site-packages/pandas/core/generic.py", line
3614, in __getattr__
    return object.__getattribute__(self, name)
AttributeError: 'Series' object has no attribute 'order'
>>> obj2
a    2
b    1
c    3
f    4
z    0
dtype: int64
>>> obj = Series([4, 3, 6, 1])
>>> obj.order()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/huang/anaconda3/lib/python3.6/site-packages/pandas/core/generic.py", line
3614, in __getattr__
    return object.__getattribute__(self, name)
AttributeError: 'Series' object has no attribute 'order'
>>> data1
     a   b   c   d
```

```
CD   0   1   2   3
CQ   4   5   6   7
GZ   8   9  10  11
XM  12  13  14  15
>>> data
    One  Two  Three  Four
CD    0    1      2     3
CQ    4    5      6     7
GZ    8    9     10    11
XM   12   13     14    15
>>> data.sort_index(by='Four')
__main__:1: FutureWarning: by argument to sort_index is deprecated, please use
.sort_values(by=...)
    One  Two  Three  Four
CD    0    1      2     3
CQ    4    5      6     7
GZ    8    9     10    11
XM   12   13     14    15
>>> data = DataFrame({'c': [5, 8,-1,3], 'e': [0, 0, 1, 1]})
>>> data
   c  e
0  5  0
1  8  0
2 -1  1
3  3  1
>>> data.sort_index(by='c')
   c  e
2 -1  1
3  3  1
0  5  0
1  8  0
>>> data.sort_index(by='e')
   c  e
0  5  0
1  8  0
2 -1  1
3  3  1
>>> data = DataFrame({'c': [5, 8,-1,3], 'e': [1, 0, 1, 1]})
>>> data
   c  e
0  5  1
1  8  0
2 -1  1
3  3  1
>>> data.sort_index(by='e')
   c  e
1  8  0
0  5  1
2 -1  1
3  3  1
>>> data.sort_index(by=['e','c'])
   c  e
1  8  0
```

```
2 -1  1
3  3  1
0  5  1
```

排名：（ranking）

排名会增设一个排名值（从 1 开始）.

使用obj.rank()方法．对于DataFrame,rank()的关键字参数axis默认为0. 但在需要时，你可以设置为1.

```
>>> data = DataFrame({'c':[4, 7.5, -3, 3], 'e':[0,1,1,0], 'f': [-2,3,10,-3]})
>>> data
     c  e   f
0  4.0  0  -2
1  7.5  1   3
2 -3.0  1  10
3  3.0  0  -3
>>> data.rank()
     c    e    f
0  3.0  1.5  2.0
1  4.0  3.5  3.0
2  1.0  3.5  4.0
3  2.0  1.5  1.0
>>> data.rank(axis=0)
     c    e    f
0  3.0  1.5  2.0
1  4.0  3.5  3.0
2  1.0  3.5  4.0
3  2.0  1.5  1.0
>>>
>>> data.rank(axis=1)
     c    e    f
0  3.0  2.0  1.0
1  3.0  1.0  2.0
2  1.0  2.0  3.0
3  3.0  2.0  1.0
>>> data
     c  e   f
0  4.0  0  -2
1  7.5  1   3
2 -3.0  1  10
3  3.0  0  -3
>>> data.rank()
     c    e    f
0  3.0  1.5  2.0
1  4.0  3.5  3.0
2  1.0  3.5  4.0
3  2.0  1.5  1.0
>>> data.rank(axis=0)
     c    e    f
0  3.0  1.5  2.0
1  4.0  3.5  3.0
2  1.0  3.5  4.0
```

```
3  2.0  1.5  1.0
>>> data.rank(axis=1)
     c    e    f
0  3.0  2.0  1.0
1  3.0  1.0  2.0
2  1.0  2.0  3.0
3  3.0  2.0  1.0
```

| --- | --- |
'average'	在相等的分组中，为各个值分配平均排名
' min '	使用整个的最小排名
' max '	……最大排名
' first '	按值在原始数据中出现的顺序排名

6. 函数汇总与计算统计

7. 处理缺失数据

8. 层次化索引