1. 层级索引

> 下面创建一个Series， 在输入索引Index时，输入了由两个子list组成的list，第一个子list是外层索引，第二个
> list是内层索引。

示例代码：

```
import pandas as pd
import numpy as np

ser_obj = pd.Series(np.random.randn(12),index=[
                ['a', 'a', 'a', 'b', 'b', 'b', 'c', 'c', 'c', 'd', 'd', 'd'],
                [0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2]
            ])
print(ser_obj)
```

运行结果：

```
a  0   -0.884133
   1   -0.567754
   2    0.593369
b  0   -1.017891
   1   -0.862712
   2   -0.276842
c  0    0.271346
   1    1.527237
   2    0.214306
d  0    0.241426
   1    1.550397
   2   -1.221101
dtype: float64
```

1.1 MultiIndex索引对象

- 打印这个Series的索引类型，显示是MultiIndex
- 直接将索引打印出来，可以看到有lavels,和labels两个信息。lavels表示两个层级中分别有那些标签，labels是每个位置分别是什么标签。

示例代码：

```
print(type(ser_obj.index))
print(ser_obj.index)
```

运行结果：

```
<class 'pandas.indexes.multi.MultiIndex'>
MultiIndex(levels=[['a', 'b', 'c', 'd'], [0, 1, 2]],
           labels=[[0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3], [0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2]])
```

选取子集

1. 根据索引获取数据。因为现在有两层索引，当通过外层索引获取数据的时候，可以直接利用外层索引的标签来获取。
2. 当要通过内层索引获取数据的时候，在list中传入两个元素，前者是表示要选取的外层索引，后者表示要选取的内层索引。

- **外层选取：**

> ser_obj['outer_label']

示例代码：

```
# 外层选取
print(ser_obj['c'])
```

运行结果：

```
0   -1.362096
1    1.558091
2   -0.452313
dtype: float64
```

- **内层选取：**

> ser_obj[:, 'inner_label']

示例代码：

```
# 内层选取
print(ser_obj[:, 2])
```

运行结果：

```
a    0.826662
b    0.015426
c   -0.452313
d   -0.051063
dtype: float64
```

常用于分组操作、透视表的生成等

1.2 交换分层顺序

swaplevel()

> .swaplevel( )交换内层与外层索引。示例代码：

```
print(ser_obj.swaplevel())
```

运行结果：

```
0  a    0.099174
1  a   -0.310414
2  a   -0.558047
0  b    1.742445
1  b    1.152924
2  b   -0.725332
0  c   -0.150638
1  c    0.251660
2  c    0.063387
0  d    1.080605
1  d    0.567547
2  d   -0.154148
dtype: float64
```

1.3 交换并排序分层

sortlevel()

> .sortlevel( )先对外层索引进行排序，再对内层索引进行排序，默认是升序。示例代码：

```
# 交换并排序分层
print(ser_obj.swaplevel().sortlevel())
```

运行结果：

```
0  a    0.099174
   b    1.742445
   c   -0.150638
   d    1.080605
1  a   -0.310414
   b    1.152924
   c    0.251660
   d    0.567547
2  a   -0.558047
   b   -0.725332
   c    0.063387
   d   -0.154148
dtype: float64
```

2. 统计计算/描述

示例代码：

```
import numpy as np
import pandas as pd

df_obj = pd.DataFrame(np.random.randn(5,4), columns = ['a', 'b', 'c', 'd'])
print(df_obj)
```

运行结果：

```
          a         b         c         d
0  1.469682  1.948965  1.373124 -0.564129
1 -1.466670 -0.494591  0.467787 -2.007771
2  1.368750  0.532142  0.487862 -1.130825
3 -0.758540 -0.479684  1.239135  1.073077
4 -0.007470  0.997034  2.669219  0.742070
```

2.1 常用的统计计算

> sum, mean, max, min…

- axis=0 按列统计，axis=1按行统计
- skipna 排除缺失值， 默认为True

示例代码：

```
import numpy as np
import pandas as pd

df_obj = pd.DataFrame(np.random.randn(5, 4), columns=['a', 'b', 'c', 'd'])

print(df_obj.sum())

print(df_obj.max())

print(df_obj.min(axis=1,  skipna=False))
```

运行结果：

```
a   -0.093774
b    1.485695
c   -2.413556
d    1.441479
dtype: float64

a    0.715802
```

```
b    2.806553
c    1.007157
d    2.592477
dtype: float64

0   -2.067502
1   -0.599844
2   -0.986101
3   -1.670302
4   -0.384834
dtype: float64
```

2.2 常用统计描述

describe 产生多个统计数据

示例代码：

```
import numpy as np
import pandas as pd

df_obj = pd.DataFrame(np.random.randn(5, 4), columns=['a', 'b', 'c', 'd'])

print(df_obj.describe())
```

运行结果：

```
              a         b         c         d
count  5.000000  5.000000  5.000000  5.000000
mean  -0.540213  0.006083  0.823476  0.527913
std    0.646222  0.980733  1.194791  1.072275
min   -1.660017 -1.353596 -0.185150 -0.845403
25%   -0.516731 -0.382940 -0.095912 -0.204725
50%   -0.253750 -0.073449  0.150285  0.554409
75%   -0.189627  0.628880  2.049298  1.485285
max   -0.080941  1.211522  2.198858  1.649998
```

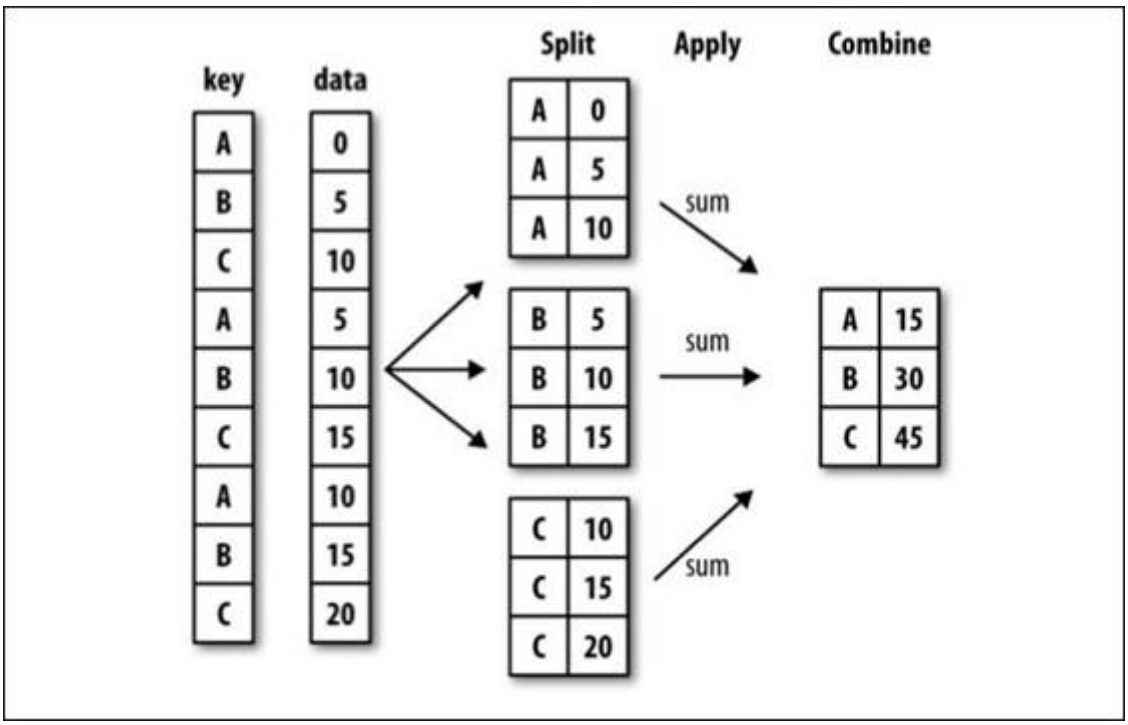
2.3 常用的统计描述方法

方法	说明
count	非NA值的数量
describe	针对Series或各DataFrame列计算汇总统计
min、max	计算最小值和最大值
argmin、argmax	计算能够获取到最小值和最大值的索引位置（整数）
idxmin、idxmax	计算能够获取到最小值和最大值的索引值
quantile	计算样本的分位数（0到1）
sum	值的总和
mean	值的平均数
median	值的算术中位数（50%分位数）
mad	根据平均值计算平均绝对离差
var	样本值的方差
std	样本值的标准差

3.分组

- 对数据集进行分组，然后对每组进行统计分析
- SQL能够对数据进行过滤，分组聚合
- pandas能利用groupby进行更加复杂的分组运算
- 分组运算过程：split->apply->combine

1. 拆分：进行分组的根据
2. 应用：每个分组运行的计算规则
3. 合并：把每个分组的计算结果合并起来



```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
                     'two', 'two', 'one','three'],
            'data1': np.random.randn(8),
            'data2': np.random.randn(8)}
df_obj = pd.DataFrame(dict_obj)

print(df_obj)
```

运行结果：

```
  key1   key2     data1     data2
0    a    one  0.061886  1.996054
1    b    one -0.490538 -1.019215
2    a    two -2.588103  1.924369
3    b  three  0.700773  0.779460
4    a    two -1.029944  1.613716
5    b    two -0.427249 -2.083439
6    a    one -0.984490 -0.796844
7    a  three -0.318081 -1.320501
```

3.1 GroupBy对象

3.1.1 分组操作

> groupby()进行分组，GroupBy对象没有进行实际运算，只是包含分组的中间数据

> 按列名分组：obj.groupby(‘label’)

示例代码：

```
# dataframe根据key1进行分组
print(type(df_obj.groupby('key1')))

# dataframe的 data1 列根据 key1 进行分组
print(type(df_obj['data1'].groupby(df_obj['key1'])))
```

运行结果：

```
<class 'pandas.core.groupby.DataFrameGroupBy'>
<class 'pandas.core.groupby.SeriesGroupBy'>
```

3.1.2 分组运算

对GroupBy对象进行分组运算/多重分组运算，如mean()

非数值数据不进行分组运算

示例代码：

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
                     'two', 'two', 'one', 'three'],
            'data1': np.random.randn(8),
            'data2': np.random.randn(8)}
df_obj = pd.DataFrame(dict_obj)

# 分组运算
grouped1 = df_obj.groupby('key1')
print(grouped1.mean())

grouped2 = df_obj['data1'].groupby(df_obj['key1'])
print(grouped2.mean())
```

运行结果：

```
         data1     data2
key1
a     0.437389 -0.230101
b     0.014657  0.802114
key1
a    0.437389
b    0.014657
Name: data1, dtype: float64
```

- size() 返回每个分组的元素个

数示例代码：

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
                     'two', 'two', 'one', 'three'],
            'data1': np.random.randn(8),
            'data2': np.random.randn(8)}
df_obj = pd.DataFrame(dict_obj)

# 分组运算
```

```
grouped1 = df_obj.groupby('key1')
grouped2 = df_obj['data1'].groupby(df_obj['key1'])

# size
print(grouped1.size())
print(grouped2.size())
```

运行结果：

```
key1
a    5
b    3
dtype: int64

key1
a    5
b    3
dtype: int64
```

3.1.3 按自定义的key分组

> obj.groupby(self_def_key)

> 自定义的key可为列表或多层列表

> obj.groupby([‘label1’, ‘label2’])->多层dataframe

示例代码：

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
                     'two', 'two', 'one','three'],
            'data1': np.random.randn(8),
            'data2': np.random.randn(8)}
df_obj = pd.DataFrame(dict_obj)

# 按自定义key分组，列表
self_def_key = [0, 1, 2, 3, 3, 4, 5, 7]
print(df_obj.groupby(self_def_key).size())

# 按自定义key分组，多层列表
print(df_obj.groupby([df_obj['key1'],   df_obj['key2']]).size())

# 按多个列多层分组
grouped2 = df_obj.groupby(['key1', 'key2'])
print(grouped2.size())

# 多层分组按key的顺序进行
```

```
grouped3 = df_obj.groupby(['key2', 'key1'])
print(grouped3.mean())
#   unstack可以将多层索引的结果转换成单层的dataframe
print(grouped3.mean().unstack())
```

运行结果：

```
0    1
1    1
2    1
3    2
4    1
5    1
7    1
dtype: int64

key1  key2
a     one      2
      three    1
      two      2
b     one      1
      three    1
      two      1
dtype: int64

key1  key2
a     one      2
      three    1
      two      2
b     one      1
      three    1
      two      1
dtype: int64

               data1     data2
key2  key1
one   a    -1.100637  0.168577
      b     0.839764 -1.048685
three a    -0.641627  1.967077
      b    -0.312337  1.239482
two   a    -0.553330  0.330943
      b     0.024266  0.064230

          data1               data2
key1          a         b         a         b
key2
one   -1.100637  0.839764  0.168577 -1.048685
three -0.641627 -0.312337  1.967077  1.239482
two   -0.553330  0.024266  0.330943  0.064230
```

3.2 GroupBy对象迭代操作

每次迭代返回一个元组 (group_name, group_data)，可用于分组数据的具体运算

3.2.1 单层分组

示例代码：

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
                     'two', 'two', 'one','three'],
            'data1': np.random.randn(8),
            'data2': np.random.randn(8)}
df_obj = pd.DataFrame(dict_obj)

grouped1 = df_obj.groupby('key1')

# 单层分组，根据key1
for group_name, group_data in grouped1:
    print(group_name)
    print(group_data)
```

运行结果：

```
a
  key1   key2     data1     data2
0    a    one  0.327682  1.137163
2    a    two -2.018393  1.156390
4    a    two  1.790860  0.062181
6    a    one  0.393871 -1.661719
7    a  three  0.452544 -0.503565

b
  key1   key2     data1     data2
1    b    one -2.731446 -0.434587
3    b  three  0.026817 -0.190618
5    b    two -1.156609  0.265790
```

3.2.2 多层分组

示例代码：

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
```

```
                    'two', 'two', 'one', 'three'],
            'data1': np.random.randn(8),
            'data2': np.random.randn(8)}
df_obj = pd.DataFrame(dict_obj)

# 按自定义key分组，列表
self_def_key = [0, 1, 2, 3, 3, 4, 5, 7]

# 按多个列多层分组
grouped = df_obj.groupby(['key1', 'key2'])

# 多层分组，根据key1 和 key2
for group_name, group_data in grouped:
    print(group_name)
    print(group_data)
```

运行结果：

```
('a', 'one')
  key1 key2     data1     data2
0    a  one  0.117471  1.044322
6    a  one -0.945911  0.881467
('a', 'three')
  key1   key2     data1     data2
7    a  three  0.826353  1.531075
('a', 'two')
  key1 key2     data1     data2
2    a  two -0.702421 -0.115524
4    a  two -0.838747  0.299189
('b', 'one')
  key1 key2     data1     data2
1    b  one  1.198171 -0.486353
('b', 'three')
  key1   key2     data1     data2
3    b  three -0.158719 -0.164132
('b', 'two')
  key1 key2     data1     data2
5    b  two  0.295161  0.189368
```

3.3 GroupBy对象转换列表或字典

示例代码：

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two', 'three',
                     'two', 'two', 'one', 'three'],
            'data1': np.random.randn(8),
```

```
            'data2': np.random.randn(8)}
df_obj = pd.DataFrame(dict_obj)

grouped = df_obj.groupby('key1')

# GroupBy对象转换list
print(list(grouped))

# GroupBy对象转换dict
print(dict(list(grouped)))
```

运行结果：

```
[('a',   key1   key2     data1     data2
0    a    one -0.441549 -1.077987
2    a    two -1.485748  0.926357
4    a    two -0.354483  0.504437
6    a    one  1.254582 -1.060020
7    a  three  1.334259 -1.090224), ('b',   key1   key2     data1     data2
1    b    one  0.454144  0.476292
3    b  three  0.840347 -0.438205
5    b    two -0.262609 -0.265114)]

{'a':   key1   key2     data1     data2
0    a    one -0.441549 -1.077987
2    a    two -1.485748  0.926357
4    a    two -0.354483  0.504437
6    a    one  1.254582 -1.060020
7    a  three  1.334259 -1.090224, 'b':   key1   key2     data1     data2
1    b    one  0.454144  0.476292
3    b  three  0.840347 -0.438205
5    b    two -0.262609 -0.265114}
```

3.3.1 按列、数据类型分组

示例代码：

```
# 按列分组
print(df_obj.dtypes)

# 按数据类型分组
print(df_obj.groupby(df_obj.dtypes, axis=1).size())
print(df_obj.groupby(df_obj.dtypes,  axis=1).sum())
```

运行结果：

```
key1      object
key2      object
data1    float64
data2    float64
```

```
dtype: object

float64    2
object     2
dtype: int64

    float64  object
0 -2.157264    aone
1  2.760672    bone
2  0.686927    atwo
3  0.081491  bthree
4 -0.245499    atwo
5 -0.415241    btwo
6  0.144359    aone
7 -0.698190  athree
```

3.3.2 其他分组方法

示例代码：

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
                     'two', 'two', 'one','three'],
            'data1': np.random.randn(8),
            'data2': np.random.randn(8)}
df_obj = pd.DataFrame(dict_obj)

df_obj2 = pd.DataFrame(np.random.randint(1, 10, (5, 5)),
                       columns=['a', 'b', 'c', 'd','e'],
                       index=['A', 'B', 'C', 'D','E'])

df_obj2.ix[1, 1:4] = np.NaN

print(df_obj2)
```

运行结果：

```
   a    b    c    d  e
A  3  1.0  4.0  2.0  3
B  1  NaN  NaN  NaN  6
C  9  8.0  5.0  4.0  6
D  2  4.0  1.0  9.0  7
E  5  7.0  9.0  7.0  4
```

3.3.3 通过字典分组

示例代码：

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
                     'two', 'two', 'one','three'],
            'data1': np.random.randn(8),
            'data2': np.random.randn(8)}
df_obj = pd.DataFrame(dict_obj)

df_obj2 = pd.DataFrame(np.random.randint(1, 10, (5, 5)),
                       columns=['a', 'b', 'c', 'd','e'],
                       index=['A', 'B', 'C', 'D','E'])

# 通过字典分组
mapping_dict = {'a': 'Python', 'b': 'Python', 'c': 'Java', 'd': 'C', 'e': 'Java'}
print(df_obj2.groupby(mapping_dict, axis=1).size())
print(df_obj2.groupby(mapping_dict, axis=1).count()) # 非NaN的个数
print(df_obj2.groupby(mapping_dict,  axis=1).sum())
```

运行结果：

```
C         1
Java      2
Python    2
dtype: int64

   C  Java  Python
A  1     2       2
B  1     2       2
C  1     2       2
D  1     2       2
E  1     2       2

   C  Java  Python
A  7    17      11
B  1    16      13
C  6     9      12
D  8     7      11
E  2    14       7
```

3.3.4 函数参数为行索引或列索引

示例代码：

```
import pandas as pd
import numpy as np
```

```
dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two', 'three',
                     'two', 'two', 'one', 'three'],
            'data1': np.random.randn(8),
            'data2': np.random.randn(8)}

# 通过函数分组
df_obj = pd.DataFrame(np.random.randint(1, 10, (5, 5)),
                      columns=['a', 'b', 'c', 'd', 'e'],
                      index=['AA', 'BBB', 'CC', 'D', 'EE'])

def group_key(idx):
    """
        idx 为列索引或行索引
    """
    # return idx
    return len(idx)

print(df_obj.groupby(group_key).size())
```

运行结果：

```
1    1
2    3
3    1
dtype: int64
```

3.3.5 通过索引级别分组

示例代码：

```
import pandas as pd
import numpy as np

# 通过索引级别分组
columns = pd.MultiIndex.from_arrays([['Python', 'Java', 'Python', 'Java', 'Python'],
                                     ['A', 'A', 'B', 'C', 'B']], names=['language', 'index'])
df_obj4 = pd.DataFrame(np.random.randint(1, 10, (5, 5)), columns=columns)
print(df_obj4)

# 根据language进行分组
print(df_obj4.groupby(level='language',   axis=1).sum())
# 根据index进行分组
print(df_obj4.groupby(level='index',  axis=1).sum())
```

运行结果：

```
language Python Java Python Java Python
```

```
index         A    A      B    C      B
0             8    9      5    2      2
1             6    6      2    9      6
2             8    6      3    8      3
3             1    8      1    5      2
4             8    2      5    7      9

language   Java   Python
0            11       15
1            15       14
2            14       14
3            13        4
4             9       22

index    A    B  C
0       17    7  2
1       12    8  9
2       14    6  8
3        9    3  5
4       10   14  7
```

4. 聚合

数据处理的最后一步为数据聚合，通常指的是转换数据，使每一个数组生成一个 单一的数

- 值数组产生标量的过程，如sum()、mean()、count()等
- 常用于对分组后的数据进行计算

示例代码：

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
                     'two', 'two', 'one', 'three'],
            'data1': np.random.randint(1, 10,8),
            'data2': np.random.randint(1, 10,8)}
df_obj = pd.DataFrame(dict_obj)
print(df_obj)
```

运行结果：

```
  key1   key2  data1  data2
0    a    one      4      2
1    b    one      8      3
2    a    two      1      5
3    b  three      1      2
4    a    two      3      5
5    b    two      8      1
6    a    one      2      6
7    a  three      8      8
```

4.1 内置的聚合函数

sum(), mean(), max(), min(), count(), size(), describe()

示例代码：

```
print(df_obj.groupby('key1').sum())
print(df_obj.groupby('key1').max())
print(df_obj.groupby('key1').min())
print(df_obj.groupby('key1').mean())
print(df_obj.groupby('key1').size())
print(df_obj.groupby('key1').count())
print(df_obj.groupby('key1').describe())
```

运行结果：

```
      data1  data2
key1
a        25     30
b        15     11

     key2  data1  data2
key1
a     two      9      8
b     two      9      7

     key2  data1  data2
key1
a     one      2      2
b     one      1      1

      data1     data2
key1
a       5.0  6.000000
b       5.0  3.666667
key1
a    5
b    3
dtype: int64
```

```
      key2  data1  data2
key1
a        5      5      5
b        3      3      3

     data1                                ...      data2
     count mean       std  min  25%  50% ...        std  min  25%  50%  75%  max
key1                                      ...
a      5.0  5.0  2.915476  2.0  3.0  4.0 ...   2.345208  2.0  6.0  7.0  7.0  8.0
b      3.0  5.0  4.000000  1.0  3.0  5.0 ...   3.055050  1.0  2.0  3.0  5.0  7.0

[2 rows x 16 columns]
```

4.2 自定义函数传入agg方法

grouped. agg(func)

func的参数为groupby索引对应的记录

示例代码：

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
                     'two', 'two', 'one','three'],
            'data1': np.random.randint(1, 10,8),
            'data2': np.random.randint(1, 10,8)}

df_obj = pd.DataFrame(dict_obj)
df_obj5 =pd.DataFrame(dict_obj)


# 自定义聚合函数
def peak_range(df):
    """
        返回数值范围
    """
    # print type(df) #参数为索引所对应的记录
    return df.max() - df.min()


print(df_obj5.groupby('key1').agg(peak_range))
print(df_obj.groupby('key1').agg(lambda df: df.max() - df.min()))
```

运行结果：

```
      data1  data2
key1
a         8      7
b         7      4

      data1  data2
key1
a         8      7
b         7      4
```

4.3 应用多个聚合函数

同时应用多个函数进行聚合操作，使用函数列表

示例代码：

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
                     'two', 'two', 'one','three'],
            'data1': np.random.randint(1, 10,8),
            'data2': np.random.randint(1, 10,8)}

df_obj =pd.DataFrame(dict_obj)


# 自定义聚合函数
def peak_range(df):
    """
        返回数值范围
    """
    # print type(df) #参数为索引所对应的记录
    return df.max() - df.min()


# 同时应用多个聚合函数
print(df_obj.groupby('key1').agg(['mean', 'std', 'count', peak_range])) #默认列名为函数名

print(df_obj.groupby('key1').agg(['mean', 'std', 'count', ('range', peak_range)])) # 通过元组提供新的列名
```

运行结果：

```
         data1                     ...         data2
          mean       std count    ...           std count peak_range
key1                               ...
a     5.000000  2.449490     5    ...      2.280351     5          6
b     7.333333  2.081666     3    ...      2.516611     3          5

[2 rows x 8 columns]

         data1                           data2
          mean       std count range      mean       std count range
key1
a     5.000000  2.449490     5     6  4.200000  2.280351     5     6
b     7.333333  2.081666     3     4  3.333333  2.516611     3     5
```

4.4 对不同列作用不同聚合函数

示例代码：

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
                     'two', 'two', 'one','three'],
            'data1': np.random.randint(1, 10,8),
            'data2': np.random.randint(1, 10,8)}

df_obj =pd.DataFrame(dict_obj)

# 每列作用不同的聚合函数
dict_mapping = {'data1':'mean',
                'data2': 'sum'}
print(df_obj.groupby('key1').agg(dict_mapping))

dict_mapping = {'data1': ['mean','max'],
                'data2': 'sum'}
print(df_obj.groupby('key1').agg(dict_mapping))
```

运行结果：

```
         data1  data2
key1
a      6.800000     19
b      5.333333     14
         data1     data2

          mean max   sum
key1
a     6.800000   9    19
b     5.333333   8    14
```

4.5 常用的内置聚合函数

函数名	说明
count	分组中非NA值的数量
sum	非NA值的和
mean	非NA值的平均值
median	非NA值的算术中位数
std、var	无偏（分母为n - 1）标准差和方差
min、max	非NA值的最小值和最大值
prod	非NA值的积
first、last	第一个和最后一个非NA值

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
                     'two', 'two', 'one', 'three'],
            'data1': np.random.randint(1, 10,8),
            'data2': np.random.randint(1, 10,8)}
df_obj = pd.DataFrame(dict_obj)
print(df_obj)

#    按key1分组后，计算data1，data2的统计信息并附加到原始表格中，并添加表头前缀
k1_sum = df_obj.groupby('key1').sum().add_prefix('sum_')
print(k1_sum)
```

运行结果：

```
  key1   key2  data1  data2
0    a    one      1      2
1    b    one      2      5
2    a    two      7      8
3    b  three      6      2
4    a    two      7      7
5    b    two      7      8
6    a    one      2      9
7    a  three      6      1

      sum_data1  sum_data2
key1
a            23         27
b            15         15
```

4.6 merge

使用merge的外连接，比较复杂

示例代码：

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
                     'two', 'two', 'one','three'],
            'data1': np.random.randint(1, 10,8),
            'data2': np.random.randint(1, 10,8)}
df_obj =pd.DataFrame(dict_obj)

#    按key1分组后，计算data1，data2的统计信息并附加到原始表格中，并添加表头前缀
k1_sum =df_obj.groupby('key1').sum().add_prefix('sum_')

# 方法1，使用merge
k1_sum_merge = pd.merge(df_obj, k1_sum, left_on='key1', right_index=True)
print(k1_sum_merge)
```

运行结果：

```
  key1   key2  data1  data2  sum_data1  sum_data2
0    a    one      4      2         26         27
2    a    two      1      5         26         27
4    a    two      9      8         26         27
6    a    one      6      8         26         27
7    a  three      6      4         26         27
1    b    one      3      1         20         11
3    b  three      9      9         20         11
5    b    two      8      1         20         11
```

4.7 transform

transform的计算结果和原始数据的形状保持一

致，如：grouped.transform(np.sum)

示例代码：

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
                     'two', 'two', 'one','three'],
            'data1': np.random.randint(1, 10,8),
            'data2': np.random.randint(1, 10,8)}
df_obj =pd.DataFrame(dict_obj)

#    按key1分组后，计算data1，data2的统计信息并附加到原始表格中，并添加表头前缀
k1_sum =df_obj.groupby('key1').sum().add_prefix('sum_')

# 方法2，使用transform
k1_sum_tf =df_obj.groupby('key1').transform(np.sum).add_prefix('sum_')
df_obj[k1_sum_tf.columns] =k1_sum_tf
print(df_obj)
```

```
  key1   key2  data1  data2           sum_key2 sum_data1 sum_data2
0    a    one      6      8  onetwotwoonethree        27        27
1    b    one      1      6        onethreetwo        10        17
2    a    two      6      7  onetwotwoonethree        27        27
3    b  three      6      7        onethreetwo        10        17
4    a    two      5      6  onetwotwoonethree        27        27
5    b    two      3      4        onethreetwo        10        17
6    a    one      1      1  onetwotwoonethree        27        27
7    a  three      9      5  onetwotwoonethree        27        27
```

也可传入自定义函数，

示例代码：

```
import pandas as pd
import numpy as np

dict_obj = {'key1': ['a', 'b', 'a', 'b',
                     'a', 'b', 'a', 'a'],
            'key2': ['one', 'one', 'two','three',
                     'two', 'two', 'one','three'],
            'data1': np.random.randint(1, 10,8),
            'data2': np.random.randint(1, 10,8)}
df_obj =pd.DataFrame(dict_obj)

#   按key1分组后，计算data1，data2的统计信息并附加到原始表格中，并添加表头前缀
k1_sum =df_obj.groupby('key1').sum().add_prefix('sum_')

k1_sum_tf =df_obj.groupby('key1').transform(np.sum).add_prefix('sum_')
df_obj[k1_sum_tf.columns] =k1_sum_tf


# 自定义函数传入transform
def diff_mean(s):
    """
        返回数据与均值的差值
    """
    return s - s.mean()

print(df_obj.groupby('key1').transform(diff_mean))
```

运行结果：

```
      data1     data2  sum_data1  sum_data2
0  1.200000 -3.200000          0          0
1  3.333333  3.333333          0          0
2  1.200000  0.800000          0          0
3 -1.666667 -1.666667          0          0
4  1.200000 -2.200000          0          0
5 -1.666667 -1.666667          0          0
6 -1.800000  2.800000          0          0
7 -1.800000  1.800000          0          0
```

4.8 groupby.apply()

apply()函数更适用于执行更为一般的GRoupBy操作，但是他对参数有特定的要求：作为参数的函数必须生成一个标量（聚合），因为只有这样才能进行广播。

示例代码：

```
import pandas as pd

frame = pd.DataFrame({'color': ['white', 'black', 'white', ' white', 'black', 'black'],
                      'status': ['up', 'up', 'down', 'down', 'down', 'up'],
                      'value1': [12.33, 14.55, 22.34, 27.84, 23.40, 18.33],
                      'value2': [11.23, 31.80, 29.99, 31.18, 18.25, 22.44]})
print(frame)
print(frame.groupby(['color']).apply(lambda x: x.max()))
```

运行结果：

```
    color status  value1  value2
0   white     up   12.33   11.23
1   black     up   14.55   31.80
2   white   down   22.34   29.99
3   white   down   27.84   31.18
4   black   down   23.40   18.25
5   black     up   18.33   22.44

         color status  value1  value2
color
 white   white   down   27.84   31.18
black    black     up   23.40   31.80
white    white     up   22.34   29.99
```