[Pandas 合并(merge)](https://geek-docs.com/pandas/pandas-data-process/pandas-merger-merge.html)

**Pandas 合并(merge)**，对于合并操作，熟悉SQL的同学可以将其理解为JOIN操作，它使用一个或多个键把多行数据结合在一起。

跟关系数据库打交道的同学通常使用SQL的JOIN查询，用几个表共用的引用值(键)从不同的表获取数据。以这些键为基础，我们能够获取列表形式的新数据，这些数据是对几个表中的数据进行组合得到的。Pandas库中这类操作叫做合并，执行合并操作的函数为merge()。

阅读本章内容前，可以先学习[Pandas基础教程](https://geek-docs.com/pandas/pandas-tutorials/pandas-tutorial.html)及[Pandas数据读写](https://geek-docs.com/pandas/pandas-read-write/pandas-data-read-and-write.html)。

**使用merge()函数进行合并**

如下所示，首先定义两个DataFrame对象，然后对两个DataFrame对象应用merge()函数进行合并操作。

import pandas as pd

frame1 = pd.DataFrame({'id':['ball', 'pencil', 'pen', 'mug', 'ashtray'],

'price':['12.33', '11.44', '33.21', '12.23', '33.62']})

frame2 = pd.DataFrame({'id':['pencil', 'pencil', 'ball', 'pen'],

'color':['white', 'red', 'red', 'black']})

print(frame1)

print('------------')

print(frame2)

print('-----------')

print(pd.merge(frame1, frame2))

Python

Copy

输出结果如下:  
  
如上所示，返回的DataFrame对象由原来两个DataFrame对象中ID相同的行组成，出了id这一列，新DataFrame包含了属于两个DataFrame的其他列。

**on 选项指定基准列**

在上面例子中，没有为merge()指定基于哪一列进行合并，实际应用中，常常需要指定基于哪一列进行合并。具体做法是增加on选项，把列的名称作为用于合并的键赋值给它。如下所示：

import pandas as pd

frame1 = pd.DataFrame({'id':['ball', 'pencil', 'pen', 'mug', 'ashtray'],

'color':['white', 'red', 'red', 'black','green'],

'brand':['OMG', 'ABC', 'ABC', 'POD', 'POD']})

frame2 = pd.DataFrame({'id':['pencil', 'pencil', 'ball', 'pen'],

'brand':['OMG', 'POD', 'ABC', 'POD']})

print(frame1)

print('------------')

print(frame2)

print('-----------')

print(pd.merge(frame1, frame2))

Python

Copy

输出结果如下:  
  
如上所示，由于我们定义的两个DataFrame对象，一个对象的列名称在另一个对象中也存在，所以对它们执行合并操作将得到一个空DataFrame对象。

因此我们需要明确定义pandas合并操作需要遵循的标准，我们用on选项指定合并操作所依据的基准列，合并标准不同，合并结果也会不同，如下所示：

import pandas as pd

frame1 = pd.DataFrame({'id':['ball', 'pencil', 'pen', 'mug', 'ashtray'],

'color':['white', 'red', 'red', 'black','green'],

'brand':['OMG', 'ABC', 'ABC', 'POD', 'POD']})

frame2 = pd.DataFrame({'id':['pencil', 'pencil', 'ball', 'pen'],

'brand':['OMG', 'POD', 'ABC', 'POD']})

print(pd.merge(frame1, frame2,on='id'))

print("------------")

print(pd.merge(frame1, frame2,on='brand'))

Python

Copy

输出结果如下:  


问题如影随形，假如两个DataFrame基准列的名称不一致，又该如何进行合并呢？为此，我们可以使用left\_on和right\_on选项指定第一个和第二个DataFrame的基准列，如下所示：

import pandas as pd

frame1 = pd.DataFrame({'id':['ball', 'pencil', 'pen', 'mug', 'ashtray'],

'color':['white', 'red', 'red', 'black','green'],

'brand':['OMG', 'ABC', 'ABC', 'POD', 'POD']})

frame2 = pd.DataFrame({'id':['pencil', 'pencil', 'ball', 'pen'],

'brand':['OMG', 'POD', 'ABC', 'POD']})

frame2.columns = ['brand', 'sid']

print(frame1)

print('------------')

print(frame2)

print('------------')

print(pd.merge(frame1,frame2,left\_on='id',right\_on='sid'))

Python

Copy

输出结果如下:  


**左连接，右连接，外连接**

如上所示，merge()函数默认执行的是**内连接操作**，上述结果执行的是**交叉操作**，其他还支持**左连接**，**右连接**和**外连接**，**外连接**把所有的键整合在一起，其效果相当于**左连接**和**右连接**的效果之和，连接类型用how选项指定。如下所示：

import pandas as pd

frame1 = pd.DataFrame({'id':['ball', 'pencil', 'pen', 'mug', 'ashtray'],

'color':['white', 'red', 'red', 'black','green'],

'brand':['OMG', 'ABC', 'ABC', 'POD', 'POD']})

frame2 = pd.DataFrame({'id':['pencil', 'pencil', 'ball', 'pen'],

'brand':['OMG', 'POD', 'ABC', 'POD']})

frame2.columns = ['brand', 'id']

print(frame1)

print('------------')

print(frame2)

print('------------')

print(pd.merge(frame1,frame2,on='id'))

Python

Copy

执行结果如下:  
  
分别执行**外连接**，**左连接**，**右连接**，如下所示：

import pandas as pd

frame1 = pd.DataFrame({'id':['ball', 'pencil', 'pen', 'mug', 'ashtray'],

'color':['white', 'red', 'red', 'black','green'],

'brand':['OMG', 'ABC', 'ABC', 'POD', 'POD']})

frame2 = pd.DataFrame({'id':['pencil', 'pencil', 'ball', 'pen'],

'brand':['OMG', 'POD', 'ABC', 'POD']})

frame2.columns = ['brand', 'id']

print(pd.merge(frame1,frame2,how='outer'))

print('------------')

print(pd.merge(frame1,frame2,how='left'))

print('------------')

print(pd.merge(frame1,frame2,how='right'))

Python

Copy

输出结果如下:

brand color id

0 OMG white ball

1 ABC red pencil

2 ABC red pen

3 POD black mug

4 POD green ashtray

5 OMG NaN pencil

6 POD NaN pencil

7 ABC NaN ball

8 POD NaN pen

------------

brand color id

0 OMG white ball

1 ABC red pencil

2 ABC red pen

3 POD black mug

4 POD green ashtray

------------

brand color id

0 OMG NaN pencil

1 POD NaN pencil

2 ABC NaN ball

3 POD NaN pen

Shell

Copy

要合并多个键，则把多个键赋值为on选项，如下示：

import pandas as pd

frame1 = pd.DataFrame({'id':['ball', 'pencil', 'pen', 'mug', 'ashtray'],

'color':['white', 'red', 'red', 'black','green'],

'brand':['OMG', 'ABC', 'ABC', 'POD', 'POD']})

frame2 = pd.DataFrame({'id':['pencil', 'pencil', 'ball', 'pen'],

'brand':['OMG', 'POD', 'ABC', 'POD']})

frame2.columns = ['brand', 'id']

print(pd.merge(frame1,frame2,on=['id','brand'],how='outer'))

Python

Copy

输出结果如下:  


**根据索引进行合并**

有的时候，合并操作不是用DataFrame的列，而是用索引作为键。把left\_index和right\_index选项的值置为True，就可将其作为合并DataFrame的基准。如下所示：

import pandas as pd

frame1 = pd.DataFrame({'id':['ball', 'pencil', 'pen', 'mug', 'ashtray'],

'color':['white', 'red', 'red', 'black','green'],

'brand':['OMG', 'ABC', 'ABC', 'POD', 'POD']})

frame2 = pd.DataFrame({'id':['pencil', 'pencil', 'ball', 'pen'],

'brand':['OMG', 'POD', 'ABC', 'POD']})

frame2.columns = ['brand', 'id']

print(pd.merge(frame1,frame2,right\_index=True,left\_index=True))

Python

Copy

输出结果如下:  


DataFrame对象的join()函数更适合于根据索引进行合并，我们可以用它合并多个**索引相同列不同**的DataFrame对象。如上所示，因为frame1的列名称和frame2的列名称有重合，直接调用frame1.join(frame2)会给出错误信息，这里要重命名frame2的列。如下所示:

import pandas as pd

frame1 = pd.DataFrame({'id':['ball', 'pencil', 'pen', 'mug', 'ashtray'],

'color':['white', 'red', 'red', 'black','green'],

'brand':['OMG', 'ABC', 'ABC', 'POD', 'POD']})

frame2 = pd.DataFrame({'id':['pencil', 'pencil', 'ball', 'pen'],

'brand':['OMG', 'POD', 'ABC', 'POD']})

frame2.columns = ['brand2', 'id2']

print(frame1.join(frame2))

Python

Copy

输出结果如下:  


如上所示，合并操作是以索引而不是列为基准，合并后得到的DataFrame对象包含了只存在于frame1的索引4，整合了frame2，索引为4的各元素使用NaN填充。

# [Pandas 拼接(concat)](https://geek-docs.com/pandas/pandas-data-process/pandas-splicing-concat.html)

**Pandas 拼接(concat)**，前面介绍了[Pandas合并](https://geek-docs.com/pandas/pandas-data-process/pandas-merger-merge.html)，Pandas另一种数据整合操作叫做拼接(concatenation)，Pandas的convat()函数实现了索引拼接的功能。

阅读本章内容前，可以先学习[Pandas基础教程](https://geek-docs.com/pandas/pandas-tutorials/pandas-tutorial.html)及[Pandas数据读写](https://geek-docs.com/pandas/pandas-read-write/pandas-data-read-and-write.html)。

## concatenate() 函数

[Numpy](https://geek-docs.com/numpy)的 concatenate()函数就是用于数组的拼接操作。如下所示:

import numpy as np

array1 = np.array([[0,1,2],[3,4,5],[6,7,8]])

array2 = array1 + 6

print(array2)

print("------------")

print(np.concatenate([array1, array2],axis=0))

print("------------")

print(np.concatenate([array1, array2],axis=1))

Python

Copy

输出结果如下:  


## concat() 函数

Pandas库以及它的Series和DataFrame等数据结构实现了带编号的索引，Pandas的concat()函数实现了按**索引拼接**的功能。如下所示：

import pandas as pd

import numpy as np

ser1 = pd.Series(np.random.rand(4), index=[1,2,3,4])

ser2 = pd.Series(np.random.rand(4), index=[5,6,7,8])

print(ser1)

print('------------')

print(ser2)

print('------------')

print(pd.concat([ser1, ser2]))

Python

Copy

输出结果如下所示:

1 0.459076

2 0.102168

3 0.779139

4 0.691197

dtype: float64

------------

5 0.164705

6 0.948454

7 0.839946

8 0.835565

dtype: float64

------------

1 0.459076

2 0.102168

3 0.779139

4 0.691197

5 0.164705

6 0.948454

7 0.839946

8 0.835565

dtype: float64

Shell

Copy

concat()函数默认按照axis=0这条轴进行拼接数据，并返回Series对象。如果指定axis=1，返回结果将是DataFrame对象，如下所示：

import pandas as pd

import numpy as np

ser1 = pd.Series(np.random.rand(4), index=[1,2,3,4])

ser2 = pd.Series(np.random.rand(4), index=[5,6,7,8])

print(pd.concat([ser1, ser2], axis=1))

Python

Copy

输出结果如下:  


concat()函数默认为**外连接**操作，把join选项设置为inner，可以执行内连接操作，如下所示：

import pandas as pd

import numpy as np

ser1 = pd.Series(np.random.rand(4), index=[1,2,3,4])

ser2 = pd.Series(np.random.rand(4), index=[5,6,7,8])

df = pd.concat([ser1, ser2], axis=1)

print(df)

print('--------')

print(pd.concat([ser1, df], axis=1, join='inner'))

Python

Copy

输出结果如下：  


## 等级索引

假如我们想用于拼接的轴上创建**等级索引**，可以借助keys选项来完成，如下所示：

import pandas as pd

import numpy as np

ser1 = pd.Series(np.random.rand(4), index=[1,2,3,4])

ser2 = pd.Series(np.random.rand(4), index=[5,6,7,8])

print(pd.concat([ser1, ser2], keys=[1,2]))

Python

Copy

输出结果如下:  


按照axis拼接Series对象，指定的键变成拼接后得到的DataFrame对象**列的名称**，如下所示：

import pandas as pd

import numpy as np

ser1 = pd.Series(np.random.rand(4), index=[1,2,3,4])

ser2 = pd.Series(np.random.rand(4), index=[5,6,7,8])

print(pd.concat([ser1, ser2], axis=1,keys=[1,2]))

Python

Copy

输出结果如下:  


## DataFrame 对象的拼接

前面介绍的是Series对象的拼接，DataFrame 对象的拼接方法与之相同，如下所示：

import pandas as pd

import numpy as np

frame1 = pd.DataFrame(np.random.rand(9).reshape(3,3), index=[1,2,3],columns=['A','B','C'])

frame2 = pd.DataFrame(np.random.rand(9).reshape(3,3), index=[4,5,6],columns=['A','B','C'])

print(pd.concat([frame1,frame2]))

Python

Copy

输出结果如下:  


指定axis=1拼接数据，如下所示：

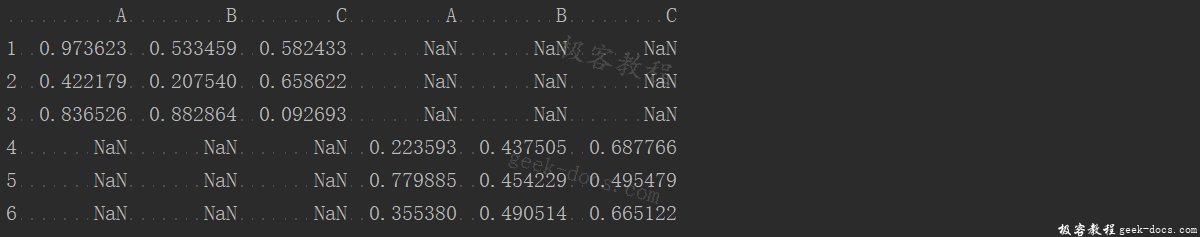
import pandas as pd

import numpy as np

frame1 = pd.DataFrame(np.random.rand(9).reshape(3,3), index=[1,2,3],columns=['A','B','C'])

frame2 = pd.DataFrame(np.random.rand(9).reshape(3,3), index=[4,5,6],columns=['A','B','C'])

print(pd.concat([frame1,frame2], axis=1))

输出结果如下:  


# [Pandas 删除数据](https://geek-docs.com/pandas/pandas-data-process/pandas-delete-data.html)

**Pandas 删除数据**，Pandas提供了一个用于删除操作的函数drop(),它返回不包含已删除索引及其元素的新对象。另外一种删除数据列的方式，就是对DataFrame对象应用del命令。通常情况，所有重复的行都需要从DataFrame对象中删除，Pandas库的drop\_duplicateds()函数实现了删除功能，该函数返回删除重复行后的DataFrame对象。

## 删除Series对象数据

我们想从[Series](https://geek-docs.com/pandas/pandas-tutorials/python_pandas_series.html)对象中删除一项，为此我们先定义一个含有四个元素的Series对象，各个元素标签均不相同。假如我们想删除标签为yellow项，用标签作为drop()函数的参数，就可以删除这一项，如下所示：

import pandas as pd

import numpy as np

ser = pd.Series(np.arange(4.), index=['red','blue','yellow','white'])

print(ser)

print('------------')

print(ser.drop('yellow'))

Python

Copy

输出结果如下:  


传入一个由多个标签组成的数组，可以删除多项，如下所示：

import pandas as pd

import numpy as np

ser = pd.Series(np.arange(4.), index=['red','blue','yellow','white'])

print(ser)

print('------------')

print(ser.drop(['blue','white']))

Python

Copy

输出结果如下：  


## 删除DataFrame对象数据

要删除[DataFrame](https://geek-docs.com/pandas/pandas-tutorials/pandas-dataframe-read-add-delete.html)中的元素，需要指定元素两个轴的轴标签，如下所示:

import pandas as pd

import numpy as np

frame = pd.DataFrame(np.arange(16).reshape((4,4)),

index=['red','blue','yellow','white'],

columns=['ball','pen','pencil','paper'])

print(frame)

print('------------')

print(frame.drop(['blue','yellow']))

Python

Copy

输出结果如下:  
  
要删除列，需要指定列的索引，还需要用axis选项指定从哪个轴删除元素（从列的方向删除，axis的值为1）,如下所示：

import pandas as pd

import numpy as np

frame = pd.DataFrame(np.arange(16).reshape((4,4)),

index=['red','blue','yellow','white'],

columns=['ball','pen','pencil','paper'])

print(frame)

print('------------')

print(frame.drop(['pen','pencil'], axis=1))

Python

Copy

输出结果:  


## del 删除数据

[Pandas DataFrame 读取 添加和删除](https://geek-docs.com/pandas/pandas-tutorials/pandas-dataframe-read-add-delete.html#DataFrame-3)有介绍另外一种删除数据列的方式，就是对DataFrame对象应用del命令，指定列名，如下所示：

import pandas as pd

import numpy as np

frame = pd.DataFrame(np.arange(16).reshape((4,4)),

index=['red','blue','yellow','white'],

columns=['ball','pen','pencil','paper'])

print(frame)

print('------------')

del frame['ball']

print(frame)

Python

Copy

输出结果如下:  


## 删除重复元素

出于多种原因，DataFrame对象可能包含重复的行，DataFrame对象的duplicated()函数可用来检测重复的行，返回元素为**布尔型**的Series对象，每个元素对应一行，如果该行与前面行**重复**，则元素为True，如何跟前面行**不重复**，则元素放回False，如下所示：

import pandas as pd

dframe = pd.DataFrame({'color':['white','white','red','red','white'],

'value':[2,1,3,3,2]})

print(dframe)

print('------------')

print(dframe.duplicated())

Python

Copy

输出结果如下:  
  
通常情况，所有重复的行都需要从DataFrame对象中删除，Pandas库的drop\_duplicateds()函数实现了删除功能，该函数返回删除重复行后的DataFrame对象，如下所示：

import pandas as pd

dframe = pd.DataFrame({'color':['white','white','red','red','white'],

'value':[2,1,3,3,2]})

print(dframe)

print('------------')

print(dframe.drop\_duplicates())

Python

Copy

输出结果如下:  


[Pandas 旋转数据](https://geek-docs.com/pandas/pandas-data-process/pandas-rotation-data.html)

**Pandas 旋转数据**，实际应用中，按行或列调整元素并不总能满足目标，有时，需要按照行重新调整列的元素或者按照列调整行的元素，本章介绍DataFrame转置,通过stack()和unstack()实现按索引旋转，pivot()实现长格式向宽格式的旋转。

**DataFrame 转置**

[pandas教程](https://geek-docs.com/pandas/pandas-tutorials)中的[DataFrame 属性和方法](https://geek-docs.com/pandas/pandas-tutorials/pandas-dataframe-values-function.html)有简单介绍DataFrame转置，转置实现行和列的交换，通过T完成，如下所示：

import pandas as pd

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Minsu','Jack']),

'Age':pd.Series([25,26,25,23,30,29,23]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

df = pd.DataFrame(d)

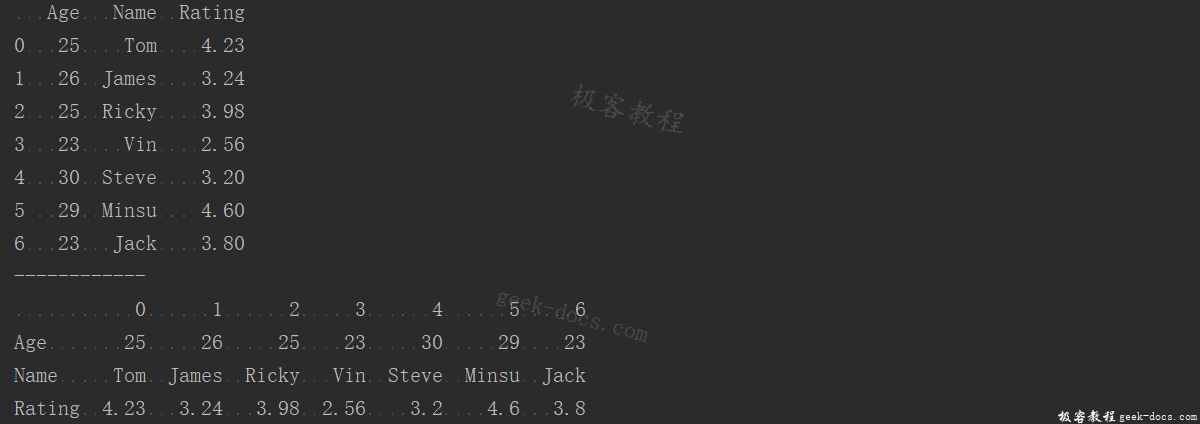
print (df)

print('------------')

print (df.T)

Python

Copy

输出结果如下:  


**按等级索引旋转**

前面讲过，DataFrame对象支持[等级索引](https://geek-docs.com/pandas/pandas-data-process/pandas-splicing-concat.html),利用这一点，可以重新调整DataFrame对象中的数据，轴向旋转有两个基础操作。

* 入栈(Stacking)：旋转数据结构，把列转换为行。
* 出栈(unStacking):把行转换为列。

如下所示,对DataFrame()对象应用stack()函数，会把列转换为行，从而得到一个Series对象。

import pandas as pd

import numpy as np

dframe = pd.DataFrame(np.arange(9).reshape(3,3),

index=['white', 'black', 'red'],

columns=['ball', 'pen', 'pencil'])

print(dframe)

print('------------')

print(dframe.stack())

Python

Copy

输出结果如下：  


在这个具有等级索引结构的Series对象上执行unstack()操作，可以重建之前的DataFrame对象，从而可以以数据透视表的形式来展示Series对象中的等级索引结构。

import pandas as pd

import numpy as np

dframe = pd.DataFrame(np.arange(9).reshape(3,3),

index=['white', 'black', 'red'],

columns=['ball', 'pen', 'pencil'])

ser = dframe.stack()

print(ser.unstack())

Python

Copy

输出结果如下:  


出栈操作可以应用不同的层级，为unstack()函数传入表示层级的编号或名称，即可对相应层级进行操作，如下所示：

import pandas as pd

import numpy as np

dframe = pd.DataFrame(np.arange(9).reshape(3,3),

index=['white', 'black', 'red'],

columns=['ball', 'pen', 'pencil'])

ser = dframe.stack()

print(ser.unstack(0))

print('------------')

print(ser.unstack(1))

Python

Copy

输出结果如下：  


**长格式向宽格式旋转**

如果数据来自仪器的读数，或是通过迭代计算得到的，或是由人工输入的一系列元素组成，该类数据集的特点是各列都有数据项，每列后面的数据常常会跟前面的有所重复，并且这类数据常常为列表形式，可以把它称作**长格式**或**栈格式**，如下所示：

import pandas as pd

import numpy as np

dframe = pd.DataFrame({'color':['white', 'white', 'white', 'red', 'red', 'red', 'black', 'black', 'black'],

'item':['ball', 'pen', 'mug', 'ball', 'pen', 'mug', 'ball', 'pen', 'mug'],

'value': np.random.rand(9)})

print(dframe)

Python

Copy

输出结果如下:  
  
因为一些字段具有多样性和重复性，这种格式的数据**可读性比较差**，出了长格式，还有一种把数据调整为列表形式的**宽格式**。这种模式可读性强，占用空间比较少，一般而言，用它存储效率也更高。

Pandas 提供了能够把长格式DataFrame转换成宽格式的pivot()函数，它以用作键的一列或多列作为参数，如下所示：

import pandas as pd

import numpy as np

longframe = pd.DataFrame({'color':['white', 'white', 'white', 'red', 'red', 'red', 'black', 'black', 'black'],

'item':['ball', 'pen', 'mug', 'ball', 'pen', 'mug', 'ball', 'pen', 'mug'],

'value': np.random.rand(9)})

print(longframe)

print('------------')

wideframe = longframe.pivot('color', 'item')

print(wideframe)

Python

Copy

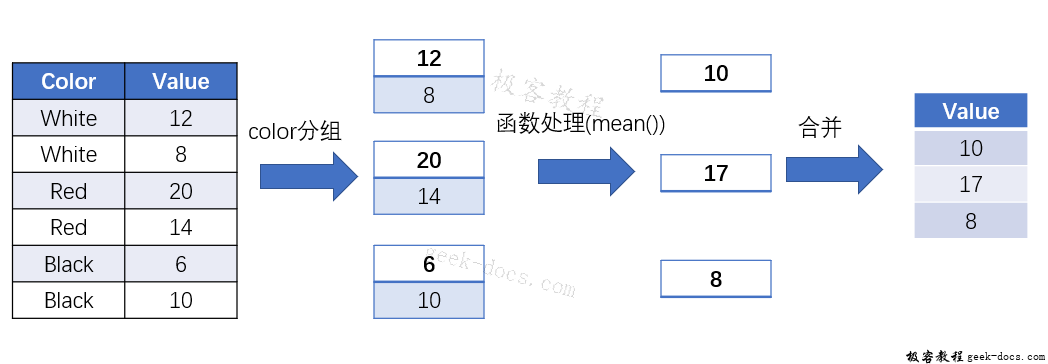
输出结果如下：  
  
如上所示，宽格式的DataFrame对象更加紧凑，可读性也更强。

[Pandas GroupBy 用法](https://geek-docs.com/pandas/pandas-data-process/pandas-groupby-usage.html)

**Pandas GroupBy用法**，现在，我们来深入分析GoupBy过程及其工作原理，它的操作模式由三个阶段组成：

* 分组：将数据集分成多个组
* 用函数处理：用函数处理每一个组
* 合并：把不同组得到的结果合并起来

第一阶段，也就是分组阶段，根据给定标准，把Series或DataFrame等[数据结构](https://geek-docs.com/pandas/pandas-tutorials/python_pandas_introduction_to_data_structures-2.html)中的数据分成不同的组，分组标准常与索引或某一列具体的元素相关。  
第二阶段也称为“用函数处理”，使用函数处理或者执行由函数定义的计算，为每组数组生成单一的值。  
第三阶段为合并，把来自每一组的结果汇集到一起，合并成一个新对象。



**GroupBy 实例**

Pandas 并没有使用三个函数来表示这个过程，而只使用了groupby()函数，它生成的GroupBy对象是整个过程的核心。通过如下例子来帮助理解，首先定义一个既包含数值又包含字符串的DataFrame对象。

import pandas as pd

df = pd.DataFrame({'color' : ['white', 'red', 'green', 'red', 'green'],

'object': ['pen', 'pencil', 'pencil', 'ashtray', 'pen'],

'price1': [5.56, 4.20, 1.30, 0.56, 2.75],

'price2': [4.75, 4.12, 1.60, 0.75, 3.15]})

print(df)

输出结果如下:

color object price1 price2

0 white pen 5.56 4.75

1 red pencil 4.20 4.12

2 green pencil 1.30 1.60

3 red ashtray 0.56 0.75

4 green pen 2.75 3.15

假如想使用Color列的组标签，计算price1列的均值，你可以先获取到price1列，然后调用groupby()函数，参数指定为color列。

df = pd.DataFrame({'color' : ['white', 'red', 'green', 'red', 'green'],

'object': ['pen', 'pencil', 'pencil', 'ashtray', 'pen'],

'price1': [5.56, 4.20, 1.30, 0.56, 2.75],

'price2': [4.75, 4.12, 1.60, 0.75, 3.15]})

group = df['price1'].groupby(df['color'])

print(group)

输出结果如下:

<pandas.core.groupby.groupby.SeriesGroupBy object at 0x000002107EE089E8>

得到的对象为GroupBy对象，刚进行的操作其实就是分组操作，把含有相同颜色的行分到同一个组中。  
可以调用GroupBy对象的group属性，查看DataFrame各行的分组情况，每个组指定好它所包含的行，就可以对每组进行操作获取结果了。

import pandas as pd

df = pd.DataFrame({'color' : ['white', 'red', 'green', 'red', 'green'],

'object': ['pen', 'pencil', 'pencil', 'ashtray', 'pen'],

'price1': [5.56, 4.20, 1.30, 0.56, 2.75],

'price2': [4.75, 4.12, 1.60, 0.75, 3.15]})

group = df['price1'].groupby(df['color'])

print(group.groups)

print(group.mean())

print(group.sum())

输出结果如下:

{'red': Int64Index([1, 3], dtype='int64'), 'white': Int64Index([0], dtype='int64'), 'green': Int64Index([2, 4], dtype='int64')}

color

green 2.025

red 2.380

white 5.560

Name: price1, dtype: float64

color

green 4.05

red 4.76

white 5.56

Name: price1, dtype: float64

**等级分组**

前面介绍了用一列元素作为键为数据分组，同理，也可以使用多列，也就是使用多个键，按照等级关系分组。如下例所示：

import pandas as pd

df = pd.DataFrame({'color' : ['white', 'red', 'green', 'red', 'green'],

'object': ['pen', 'pencil', 'pencil', 'ashtray', 'pen'],

'price1': [5.56, 4.20, 1.30, 0.56, 2.75],

'price2': [4.75, 4.12, 1.60, 0.75, 3.15]})

group = df['price1'].groupby([df['color'], df['object']])

print(group.groups)

print(group.sum())

输出结果如下:

{('white', 'pen'): Int64Index([0], dtype='int64'), ('green', 'pen'): Int64Index([4], dtype='int64'), ('red', 'ashtray'): Int64Index([3], dtype='int64'), ('red', 'pencil'): Int64Index([1], dtype='int64'), ('green', 'pencil'): Int64Index([2], dtype='int64')}

color object

green pen 2.75

pencil 1.30

red ashtray 0.56

pencil 4.20

white pen 5.56

Name: price1, dtype: float64

我们也可以按照多列数据或整个DataFrame把数据分成几组，如果你不想反复多次使用GroupBy对象，最方便的办法就是一次就把所有的分组依据和计算方法都指定好，无需定义任何中间变量，如下列所示：

import pandas as pd

df = pd.DataFrame({'color' : ['white', 'red', 'green', 'red', 'green'],

'object': ['pen', 'pencil', 'pencil', 'ashtray', 'pen'],

'price1': [5.56, 4.20, 1.30, 0.56, 2.75],

'price2': [4.75, 4.12, 1.60, 0.75, 3.15]})

print(df[['price1','price2']].groupby(df['color']).mean())

print(df.groupby(df['color']).mean())

输出结果如下:

price1 price2

color

green 2.025 2.375

red 2.380 2.435

white 5.560 4.750

price1 price2

color

green 2.025 2.375

red 2.380 2.435

white 5.560 4.750