Part1 Series

In [9]:
```
import numpy as np
import pandas as pd

ar1 = np.random.rand(5)
print(ar1)

s = pd.Series(np.random.rand(5))
print(s)
print(type(s))

print(s.index) # RangeIndex(start=0, stop=5, step=1)
print(s.values)# [0.86582495 0.84940667 0.63361172 0.20719841 0.46959806]
```

```
[0.68980522 0.44477913 0.74757779 0.58221409 0.67278749]
0    0.865825
1    0.849407
2    0.633612
3    0.207198
4    0.469598
dtype: float64
<class 'pandas.core.series.Series'>
RangeIndex(start=0, stop=5, step=1)
[0.86582495 0.84940667 0.63361172 0.20719841 0.46959806]
```

In [12]:
```
# 创建Series 创建方法一：由字典创建，字典的key就是index，values就是values

s_dict = {'a':1,'b':2,'c':3,'d':4}

s = pd.Series(s_dict)

print(s)
print(s.index)
```

```
a    1
b    2
c    3
d    4
dtype: int64
Index(['a', 'b', 'c', 'd'], dtype='object')
```

In [21]:
```
# Series 创建方法二：由数组创建(一维数组)

arr = np.random.randn(5) # 正态分布
print(arr)

print("-"*30)

s= pd.Series(arr)
print(s)

print("-"*30)

s = pd.Series(arr, ['a','b','c','d','e'])
print(s)
```

```
print("-"*30)
s = pd.Series(arr, ['a','b','c','d','e'],dtype=np.object)
print(s)
```

```
[ 0.66486081 -0.7303337  -0.33169069 -0.97388709  0.92401891]
------------------------------
0    0.664861
1   -0.730334
2   -0.331691
3   -0.973887
4    0.924019
dtype: float64
------------------------------
a    0.664861
b   -0.730334
c   -0.331691
d   -0.973887
e    0.924019
dtype: float64
------------------------------
a    0.664861
b   -0.730334
c   -0.331691
d   -0.973887
e    0.924019
dtype: object
```

In [24]:
```
# Series 创建方法三：由标量创建

s = pd.Series(10, index=range(10))
print(s)

# 如果data是标量值，则必须提供索引。该值会重复，来匹配索引的长度
```

```
0    10
1    10
2    10
3    10
4    10
5    10
6    10
7    10
8    10
9    10
dtype: int64
```

Series 名称属性：name

name为Series的一个参数，创建一个数组的 名称

.name方法：输出数组的名称，输出格式为str，如果没用定义输出名称，输出为None

In [29]:
```
# Series 名称属性：name

s1 = pd.Series(np.random.randn(5))
print(s1)
print('-'*30)
```

```
s2 = pd.Series(np.random.randn(5),name="cyx")
print(s2)
print('-'*30)
print(s1.name,s2.name)
#name为Series的一个参数，创建一个数组的 名称
#.name方法：输出数组的名称，输出格式为str，如果没用定义输出名称，输出为None

s3 = s2.rename("cyx2")
print(s3)
#.rename()重命名一个数组的名称，并且新指向一个数组，原数组不变
```

```
0   -0.250166
1   -0.009366
2    1.498835
3   -0.062361
4   -1.145670
dtype: float64
------------------------------
0    0.060669
1   -0.910083
2    0.352631
3   -0.465147
4   -0.137170
Name: cyx, dtype: float64
------------------------------
None cyx
0    0.060669
1   -0.910083
2    0.352631
3   -0.465147
4   -0.137170
Name: cyx2, dtype: float64
```

In [34]:

```
#位置下标，类似序列

s = pd.Series(np.random.randn(5))
print(s)
print(s[0])
print(float(s[0]))

#print(s[-1]) 这个是会报错的
```

```
0   -1.652453
1   -1.459236
2   -0.697696
3    1.030089
4    1.131499
dtype: float64
-1.652452833093018
-1.652452833093018
```

In [37]:

```
#标签索引

s = pd.Series(np.random.randn(5),index=['a','b','c','d','e'])
print(s)
print('-'*30)
print(s['a'])
#方法类似下标索引，用[]表示，内写上index，注意index是字符串

sci = s[['b','e']]
```

```
print(sci)
#如果需要选择多个标签的值，用[[]]来表示（相当于[]中包含一个列表）
#多标签索引结果是新的数组
```

```
a    1.051399
b    0.065979
c   -0.822940
d   -0.665114
e   -0.446152
dtype: float64
------------------------------
1.0513990659101424
b    0.065979
e   -0.446152
dtype: float64
```

In [47]:

```
#切片索引

s1 = pd.Series(np.random.randn(5))
s2 = pd.Series(np.random.randn(5), index=['a','b','c','d','e'])
print(s1[1:4])
print('-'*30)
print(s2['a':'d'])
print('-'*30)
print("以上有一个很明显的区别")
print('-'*30)
print(s2[0:3],s2[3])
print('-'*30)
#注意：用index做切片是末端包含

print(s2[:-1])
print(s2[::2])
#下标索引做切片，和list写法一样
```

```
1    0.248457
2   -1.626060
3   -0.890339
dtype: float64
------------------------------
a   -2.573373
b   -0.872933
c   -0.386438
d    0.275602
dtype: float64
------------------------------
以上有一个很明显的区别
------------------------------
a   -2.573373
b   -0.872933
c   -0.386438
dtype: float64 0.2756018363408343
------------------------------
a   -2.573373
b   -0.872933
c   -0.386438
d    0.275602
dtype: float64
a   -2.573373
c   -0.386438
e   -0.090068
dtype: float64
```

In [96]:
```
#布尔型索引

s = pd.Series(np.random.rand(3)*100)
s[4] = None
print(s)
print('-'*30)
bs1 = s > 50
bs2 = s.isnull()
bs3 = s.notnull()
print(bs1)
print('-'*30)
print(bs2)
print('-'*30)
print(bs3)
#数组做判断之后，返回的是一个由布尔值组成的新的数组
#.isnull() / .notnull() 判断是否为空值 (None代表空值，NaN代表有问题的数值，两个都会识别为空值)

print('-'*30)

print(s[s>50])
print(s[bs3])
```

```
0    76.4737
1    32.9561
2    91.5164
4       None
dtype: object
------------------------------
0     True
1    False
2     True
4    False
dtype: bool
------------------------------
0    False
1    False
2    False
4     True
dtype: bool
------------------------------
0     True
1     True
2     True
4    False
dtype: bool
------------------------------
0    76.4737
2    91.5164
dtype: object
0    76.4737
1    32.9561
2    91.5164
dtype: object
```

Pandas数据结构Series：基本技巧

数据查看 / 重新索引 / 对齐 / 添加、修改、删除值

In [104]:
```
#数据查看
```

```
s = pd.Series(np.random.rand(15))
print(s)
print('*'*30)
print(s.head(5))
print(s.tail(5))
#.head()查看头部数据
#.tail()查看尾部数据
#默认查看5条
```

```
0     0.566268
1     0.384991
2     0.471614
3     0.985232
4     0.293593
5     0.423737
6     0.402084
7     0.180807
8     0.176709
9     0.871237
10    0.031719
11    0.339411
12    0.180365
13    0.824339
14    0.852797
dtype: float64
**********
0    0.566268
1    0.384991
2    0.471614
3    0.985232
4    0.293593
dtype: float64
10    0.031719
11    0.339411
12    0.180365
13    0.824339
14    0.852797
dtype: float64
```

In [107]:

```
#重新索引reindex
#.reindex将会根据索引重新排序，如果当前索引不存在，则引入缺失值

s = pd.Series(np.random.rand(3), index = ['a','b','c'])
print(s)
s1 = s.reindex(['c','b','a','d'])
print(s1)
#.reindex()中也是写列表
#这里'd'索引不存在，所以值为NaN

s2 = s.reindex(['c','b','a','d'], fill_value = 0)
print(s2)
#fill_value参数：填充缺失值的值
```

```
a    0.654757
b    0.071198
c    0.710437
dtype: float64
c    0.710437
b    0.071198
a    0.654757
```

```
d       NaN
dtype: float64
c    0.710437
b    0.071198
a    0.654757
d    0.000000
dtype: float64
```

In [124]:
```
# Series对齐

# Series对齐

s1 = pd.Series(np.random.rand(3), index = ['Jack','Marry','Tom'])
s2 = pd.Series(np.random.rand(3), index = ['Wang','Jack','Marry'])
print(s1)
print(s2)
print(s1+s2)
# Series 和 ndarray 之间的主要区别是，Series 上的操作会根据标签自动对齐
# index顺序不会影响数值计算，以标签来计算
# 空值和任何值计算结果扔为空值
```

```
Jack     0.092190
Marry    0.579345
Tom      0.854684
dtype: float64
Wang     0.036855
Jack     0.711834
Marry    0.340141
dtype: float64
Jack     0.804024
Marry    0.919486
Tom           NaN
Wang          NaN
dtype: float64
```

In [125]:
```
# 删除：.drop

s = pd.Series(np.random.rand(5), index = list('ngjur'))
print(s)
s1 = s.drop('n')
s2 = s.drop(['g','j'])
print(s1)
print(s2)
print(s)
# drop 删除元素之后返回副本(inplace=False)
```

```
n    0.784218
g    0.127934
j    0.186821
u    0.492626
r    0.068913
dtype: float64
g    0.127934
j    0.186821
u    0.492626
r    0.068913
dtype: float64
n    0.784218
u    0.492626
r    0.068913
dtype: float64
```

```
n    0.784218
g    0.127934
j    0.186821
u    0.492626
r    0.068913
dtype: float64
```

In [127]:
```
#添加

s1 = pd.Series(np.random.rand(5))
s2 = pd.Series(np.random.rand(5), index = list('ngjur'))
print(s1)
print(s2)
s1[5] = 100
s2['a'] = 100
print(s1)
print(s2)
print('-----')
#直接通过下标索引/标签index添加值

s3 = s1.append(s2)
print(s3)
#通过.append方法，直接添加一个数组
#.append方法生成一个新的数组，不改变之前的数组
```

```
0    0.977752
1    0.480619
2    0.659356
3    0.220181
4    0.138124
dtype: float64
n    0.007651
g    0.186073
j    0.317048
u    0.206858
r    0.605039
dtype: float64
0      0.977752
1      0.480619
2      0.659356
3      0.220181
4      0.138124
5    100.000000
dtype: float64
n      0.007651
g      0.186073
j      0.317048
u      0.206858
r      0.605039
a    100.000000
dtype: float64
-----
0      0.977752
1      0.480619
2      0.659356
3      0.220181
4      0.138124
5    100.000000
n      0.007651
g      0.186073
j      0.317048
```

```
u      0.206858
r      0.605039
a    100.000000
dtype: float64
```

In [128]:
```
#修改

s = pd.Series(np.random.rand(3), index = ['a','b','c'])
print(s)
s['a'] = 100
s[['b','c']] = 200
print(s)
#通过索引直接修改，类似序列
```

```
a    0.748182
b    0.218597
c    0.931674
dtype: float64
a    100.0
b    200.0
c    200.0
dtype: float64
```

In [ ]:
```

```