Table of Contents

In [13]:
```
#全部行都能输出
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

1 Series的定义与使用

- Series 是一个带有 名称 和 索引 的**一维数组。**
- Series 中包含的数据类型可以是整数、浮点、字符串、列表、元组、ndarray等。
- 假定有一个场景是：存储一些用户的信息，暂时只包括年龄信息。
- 我们可以通过 Series 来存储，这里我们通过 Series 存储了四个年龄：18/30/25/40，只需将要存储的数据构建成一个数组，然后赋值给data参数即可。

pd.Series(['data=None', 'index=None', 'dtype=None', 'name=None'],)

In [14]:
```
# 导入相关库（一般一起导入两个包）
import numpy as np
import pandas as pd
```

In [3]:
```
sr=pd.Series(range(10,20))
sr
```

Out[3]:
```
0    10
1    11
2    12
3    13
4    14
5    15
6    16
7    17
8    18
9    19
dtype: int64
```

从上面显示的dtype可以看出，和ndarray一样，Series中只能保存一种数据类型，如果数据类型不一致的话，也会自动转化为一致，转化的规则和ndarray类似：

In [20]:
```
pd.Series([1,2.1,3,4])
pd.Series([1,"2",3,4])
```

Out[20]:
```
0    1.0
1    2.1
2    3.0
3    4.0
dtype: float64
```

Out[20]:
```
0    1
1    2
2    3
3    4
dtype: object
```

Series只能用来定义一维数组（及其索引），如果将多维数组强制转换为Series的话：

In [26]:
```
a=np.random.randint(1,10,(2,2))

pd.Series(a)
```

...

In [5]:
```
a=[[[1],[2]],[[2],[3]]]
pd.Series(a)
```

Out[5]:
```
0    [[1], [2]]
1    [[2], [3]]
dtype: object
```

1.1 Series索引标签的添加

1.1.1 方法一

In [5]:
```
name = ['蜘蛛侠', '灭霸', '奇异博士', '钢铁侠', '蝙蝠侠', '索尔']
age = [22, 3000, 33, 37, 40, 1500]

heroes_age = pd.Series(age,index=name)    #索引index作为pd.Series()中的参数来为heroes_age指定索引
heroes_age
```

Out[5]:
```
蜘蛛侠      22
灭霸      3000
奇异博士      33
钢铁侠      37
蝙蝠侠      40
索尔      1500
dtype: int64
```

定义之后就可以查看Series的索引标签：

In [6]:
```
heroes_age.index
```

Out[6]:
```
Index(['蜘蛛侠', '灭霸', '奇异博士', '钢铁侠', '蝙蝠侠', '索尔'], dtype='object')
```

1.1.2 方法二

建立好Series之后，用一个新的列表（或者其他有序序列）赋值到该Series的索引对象Index中。

In [7]:
```
user_age = pd.Series([22, 3000, 33, 37, 40, 1500])

user_age.index = ['蜘蛛侠', '灭霸', '奇异博士', '钢铁侠', '蝙蝠侠', '索尔']

user_age
```

Out[7]:
```
蜘蛛侠      22
灭霸      3000
奇异博士      33
钢铁侠      37
蝙蝠侠      40
索尔      1500
dtype: int64
```

1.2 Series及其索引名字的添加

1.2.1 Series名字的添加

方法一：

In [28]:
```
#直接用pd.Series()中的name参数来设置
name = ['蜘蛛侠', '灭霸', '奇异博士', '钢铁侠', '蝙蝠侠', '索尔']
age = [22, 3000, 33, 37, 40, 1500]
heroes_age = pd.Series(age,index=name,name='英雄年龄')

heroes_age
heroes_age.name
```

Out[28]:
```
蜘蛛侠      22
灭霸      3000
奇异博士      33
钢铁侠      37
蝙蝠侠      40
索尔      1500
Name: 英雄年龄, dtype: int64
```

Out[28]:
```
'英雄年龄'
```

方法二：

In [29]:
```
#赋值到该Series的索引对象中
name = ['蜘蛛侠', '灭霸', '奇异博士', '钢铁侠', '蝙蝠侠', '索尔']
age = [22, 3000, 33, 37, 40, 1500]
heroes_age = pd.Series(age,index=name)

heroes_age.name="英雄年龄"

heroes_age
heroes_age.name
```

Out[29]:
```
蜘蛛侠      22
灭霸      3000
奇异博士      33
钢铁侠      37
蝙蝠侠      40
索尔      1500
Name: 英雄年龄, dtype: int64
```

Out[29]:
```
'英雄年龄'
```

1.2.2 索引名字的添加

In [8]:
```
user_age.index = ['蜘蛛侠', '灭霸', '奇异博士', '钢铁侠', '蝙蝠侠', '索尔']
user_age.index.name = '英雄姓名'
user_age
```

Out[8]:
```
英雄姓名
蜘蛛侠      22
灭霸      3000
奇异博士      33
钢铁侠      37
蝙蝠侠      40
索尔      1500
dtype: int64
```

In [9]:
```
user_age.index.name
```

Out[9]:
```
'英雄姓名'
```

1.3 Pandas的Index对象

综上所述, 一个Series包括了data, index以及name

但是按照上面的方法，定义一个完整元素的Series的索引，需要先定义索引标签，再定义索引的名字，未免太繁琐。
可以通过定义Index对象，再将其赋值到Series中的index参数，这样可以一次性定义一个有完整元素的Series索引。

In [30]:
```
#通过pd.Index方法先创建一个索引，再将索引添加到series中去
data=[22, 3000, 33, 37, 40, 1500]
index = pd.Index(['蜘蛛侠', '灭霸', '奇异博士', '钢铁侠','蝙蝠侠', '索尔' ], name="英雄姓名")  #
# index=['蜘蛛侠', '灭霸', '奇异博士', '钢铁侠','蝙蝠侠', '索尔']

user_age = pd.Series(data=data, index=index, name="英雄年龄")

user_age
```

Out[30]:
```
英雄姓名
蜘蛛侠      22
灭霸      3000
奇异博士      33
钢铁侠      37
蝙蝠侠      40
索尔      1500
Name: 英雄年龄, dtype: int64
```

1.4 练习

自由构建几个Series。

In [ ]:
```
```

1.5 Series的索引和切片

In [32]:
```
name = ['蜘蛛侠', '灭霸', '奇异博士', '钢铁侠', '蝙蝠侠', '索尔']
age = [22, 3000, 33, 37, 40, 1500]
heroes_age = pd.Series(age,index=name)
heroes_age
```

Out[32]:
```
蜘蛛侠      22
灭霸      3000
奇异博士      33
钢铁侠      37
蝙蝠侠      40
索尔      1500
dtype: int64
```

In [15]:
```
heroes_age[0]
heroes_age['蜘蛛侠']
```

Out[15]:
```
22
```

Out[15]:
```
22
```

In [16]:
```
heroes_age[0::2]
heroes_age['蜘蛛侠':'索尔':2]
```

Out[16]:
```
蜘蛛侠      22
奇异博士     33
蝙蝠侠      40
dtype: int64
```

Out[16]:
```
蜘蛛侠      22
奇异博士     33
蝙蝠侠      40
dtype: int64
```

In [17]:
```
#单独抽取某些数据
heroes_age[[1,2,4]]
heroes_age[['灭霸','奇异博士','蝙蝠侠']]
```

Out[17]:
```
灭霸       3000
奇异博士       33
蝙蝠侠        40
dtype: int64
```

Out[17]:
```
灭霸       3000
奇异博士       33
蝙蝠侠        40
dtype: int64
```

Series 和字典非常类似, 我们可以将index和其标签看成是key, 对应的值看成是value。

两个有很多类似的操作,比如Series同样可以使用.get()方法，且如果在Series中无法找到要找的值，可以设定返回默认值:

In [18]:
```
user_age["蜘蛛侠"]
user_age.get("蜘蛛侠")
```

Out[18]:
```
22
```

Out[18]:
```
22
```

In [19]:
```
user_age.get("闪电侠", '不存在')
```

Out[19]:
```
'不存在'
```

2 Series掩码提取

Series值筛选与提取的方法和ndarray基本一致:

In [34]:
```
name = ['蜘蛛侠', '灭霸', '奇异博士', '钢铁侠', '蝙蝠侠', '索尔']
age = [22, 3000, 33, 37, 40, 1500]
heroes_age = pd.Series(age,index=name)
heroes_age
```

Out[34]:
```
蜘蛛侠      22
灭霸       3000
奇异博士      33
钢铁侠       37
蝙蝠侠       40
索尔       1500
dtype: int64
```

In [35]:
```
# heroes_age[heroes_age>100]

heroes_age[(heroes_age>1000)*(heroes_age<2000)]
```

Out[35]:
```
索尔     1500
dtype: int64
```

In [42]:
```
# 提取年龄为偶数的数据
#提取年龄为偶数，且年龄小于100的英雄年龄
heroes_age[heroes_age%2==0]

heroes_age[(heroes_age<100)]

heroes_age[(heroes_age%2==0)&(heroes_age<100)]    # 在多个逻辑条件下，用&(交集）或者|（并集）
```

Out[42]:
```
蜘蛛侠      22
灭霸       3000
蝙蝠侠       40
索尔       1500
dtype: int64
```

Out[42]:
```
蜘蛛侠      22
奇异博士      33
钢铁侠       37
蝙蝠侠       40
dtype: int64
```

Out[42]:
```
蜘蛛侠     22
蝙蝠侠     40
dtype: int64
```

In [49]:
```
heroes_age[(heroes_age%2==0)|(heroes_age<100)]     #换成“+”号也可以得到一致的结果
```

Out[49]:
```
蜘蛛侠      22
灭霸       3000
奇异博士      33
钢铁侠       37
蝙蝠侠       40
索尔       1500
dtype: int64
```

因为Series底层封装的也ndarray数组结构，因此同样支持向量化操作，可以利用

3 Series运算符和广播方法

Series支持ndarray的运算符和广播方法，包括numpy中的各种运算函数、聚合函数等。

In [53]:
```
user_age
```

Out[53]:
```
英雄姓名
蜘蛛侠        22
灭霸        3000
奇异博士        33
钢铁侠         37
蝙蝠侠         40
索尔        1500
Name: 英雄年龄, dtype: int64
```

In [54]:
```
user_age + range(6)
```

Out[54]:
```
英雄姓名
蜘蛛侠        22
灭霸        3001
奇异博士        35
钢铁侠         40
蝙蝠侠         44
索尔        1505
Name: 英雄年龄, dtype: int64
```

In [50]:
```
user_age + 1
```

Out[50]:
```
英雄姓名
蜘蛛侠        23
灭霸        3001
奇异博士        34
钢铁侠         38
蝙蝠侠         41
索尔        1501
Name: 英雄年龄, dtype: int64
```

In [23]:
```
user_age ** 2
```

Out[23]:
```
蜘蛛侠          484
灭霸        9000000
奇异博士         1089
钢铁侠          1369
蝙蝠侠          1600
索尔        2250000
Name: 英雄年龄, dtype: int64
```

In [24]:
```
np.log(user_age)
```

Out[24]:
```
蜘蛛侠       3.091042
灭霸        8.006368
奇异博士      3.496508
钢铁侠       3.610918
蝙蝠侠       3.688879
索尔        7.313220
Name: 英雄年龄, dtype: float64
```

In [25]:
```
np.exp(user_age)
```

Out[25]:
```
蜘蛛侠       3.584913e+09
灭霸                 inf
奇异博士      2.146436e+14
钢铁侠       1.171914e+16
蝙蝠侠       2.353853e+17
索尔                 inf
Name: 英雄年龄, dtype: float64
```

In [26]:
```
np.mean(user_age)

np.max(user_age)

np.std(user_age)
```

Out[26]:
```
772.0
```

Out[26]:
```
3000
```

Out[26]:
```
1131.2705246756852
```

In [27]:
```
# 我们注意到原来的user_age并没有发生变化
user_age
```

Out[27]:
```
蜘蛛侠         22
灭霸         3000
奇异博士         33
钢铁侠         37
蝙蝠侠         40
索尔         1500
Name: 英雄年龄, dtype: int64
```