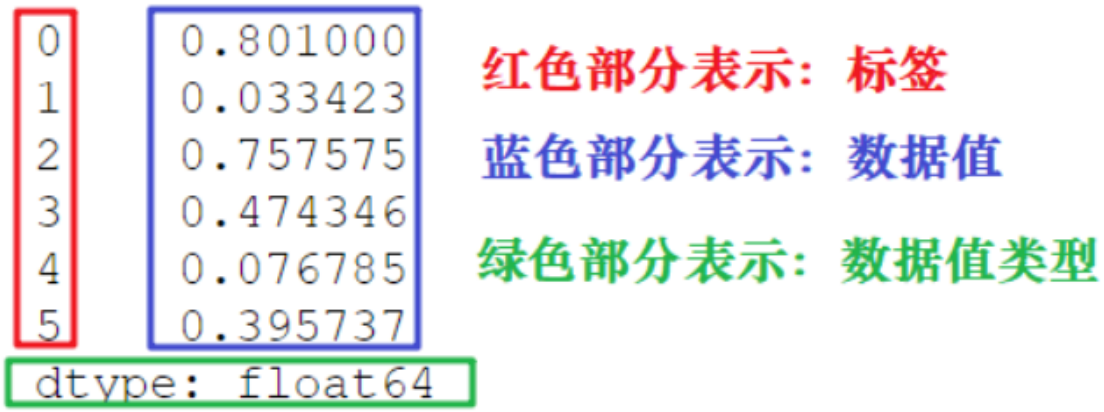
Series 结构

Series 结构，也称 Series 序列，是 Pandas 常用的数据结构之一，它是一种类似于一维数组的结构，由一组数据值（value）和一组标签组成，其中标签与数据值具有对应关系。

标签不必是唯一的，但必须是可哈希类型。该对象既支持基于整数的索引，也支持基于标签的索引，并提供了许多方法来执行涉及索引的操作。ndarray的统计方法已被覆盖，以自动排除缺失的数据（目前表示为NaN）

Series 可以保存任何数据类型，比如整数、字符串、浮点数、Python 对象等，它的标签默认为整数，从 0 开始依次递增。Series 的结构图，如下所示：



通过标签我们可以更加直观地查看数据所在的索引位置。

```
# 引用numpy
import numpy as np
# 引入pandas
import pandas as pd
```

数据结构Series创建

```
pd.Series(data=None, index=None, dtype=None, name=None, copy=False)
```

- data　输入的数据，可以是列表、常量、ndarray 数组等,如果是字典,则保持参数顺序

- index 索引值,必须是可散列的(不可变数据类型（str，bytes和数值类型）)，并且与数据具有相同的长度,允许使用非唯一索引值。如果未提供，将默认为RangeIndex（0，1，2，…，n）
- dtype 输出系列的数据类型。如果未指定，将从数据中推断
- name 为Series定义一个名称
- copy　表示对 data 进行拷贝，默认为 False,仅影响Series和ndarray数组

1. 创建

1) 列表/数组作为数据创建Series

```
# 列表作为数据创建Series
ar_list = [3,10,3,4,5]
print(type(ar_list))
# 使用列表创建Series
s1 = pd.Series(ar_list)
print(s1)
print(type(s1))
```

```
<class 'list'>
0     3
1    10
2     3
3     4
4     5
dtype: int64
<class 'pandas.core.series.Series'>
```

```
# 数组作为数据源
np_rand = np.arange(1,6)
# 使用数组创建Series
s1 = pd.Series(np_rand)
s1
```

- 通过index 和values属性取得对应的标签和值

```
# 默认为RangeIndex (0, 1, 2, …, n)
s1.index
```

```
# 可以强制转化为列表输出
list(s1.index)
```

```
# 返回Series所有值,数据类型为ndarray
print(s1.values, type(s1.values))
```

- 通过索引取得对应的值,或者修改对应的值

```
s1[1]  # 取得索引为1 的数据
```

```
s1[2] = 50 # 改变索引为2的数据值
s1
```

```
s1[-1]
```

```

```

- 和列表索引区别:
- 默认的索引RangeIndex,不能使用负值,来表示从后往前找元素,
- 获取不存在的索引值对应数据,会报错,但是可以赋值,相当于新增数据
- 可以新增不同类型索引的数据,新增不同类型索引的数据,索引的类型会发生自动变化

```
# ①.默认的索引RangeIndex,不能使用负值,来表示从后往前找元素,
s1[-1]
```

```
# ②当前索引为-1,不存在,不会报错
s1[-1] = 20
print(s1)
print(s1.index)
```

```
# ③新增不同类型索引的数据,索引的类型会发生自动变化
s1["a"] = 40
s1.index
```

```
print(s1)
s1[-1]
```

2) 字典作为数据源创建Series

```
d = {'a': 1, 'b': 2, 'c': 3}
ser = pd.Series(data=d)
ser
```

- 通过index 和values属性取得对应的标签和值

```
# 标签索引
ser.index
```

```
# Series值
ser.values
```

- 通过索引取得对应的值,或者修改对应的值

```
ser['a']
```

```
print(ser)
#ser[10] = 50  # 错误
ser["s"] = 50
ser
```

```
ser.index
```

```
ser["a"]
```

```
ser[1]
```

```
d = {'a': 1, 0: 2, 'c': 3}
ser1 = pd.Series(data=d)
ser1
```

```
a    1
0    2
c    3
dtype: int64
```

```
ser1[2]
```

- 取得数据时,先进行标签的检查,如果标签中没有,再进行索引的检查,都不存在则报错

```
d = {'a': 1, 'b': 2, 'c': 3}
ser = pd.Series(data=d)
print(ser)
print("==============")
# 取得第一个元素
print('ser["a"]:%s'% ser["a"],' ser[0]:%s'% ser[0])
# 取得最后一个元素
print('ser["c"]:%s'% ser["c"],' ser[-1]:%s'% ser[-1])
```

```

```

3) 通过标量创建

```
s = pd.Series(100,index=range(5))
s
```

2. 参数说明

- a. index 参数

索引值,必须是可散列的(不可变数据类型（str，bytes和数值类型），并且与数据具有相同的长度,允许使用非唯一索引值。如果未提供，将默认为RangeIndex（0，1，2，…，n）

- 使用“显式索引”的方法定义索引标签

```
data = np.array(['a','b','c','d'])
#自定义索引标签（即显示索引），需要和数据长度一致
s = pd.Series(data,index=[100,101,102,103])
s
```

- 从指定索引的字典构造序列

```
d = {'a': 1, 'b': 2, 'c': 3}
ser = pd.Series(d, index=['a', 'b', 'c'])
ser
```

- 当传递的索引值未匹配对应的字典键时，使用 NaN（非数字）填充。

```
d = {'a': 1, 'b': 2, 'c': 3}
ser = pd.Series(data=d, index=['x', 'b', 'z'])
ser
```

请注意，索引是首先使用字典中的键构建的。在此之后，用给定的索引值对序列重新编制索引，因此我们得到所有NaN。

- 通过匹配的索引值,改变创建Series数据的顺序

```
d = {'a': 1, 'b': 2, 'c': 3}
ser = pd.Series(data=d, index=['c', 'b', 'a'])
ser
```

- b. name参数

我们可以给一个Series对象命名，也可以给一个Series数组中的索引列起一个名字，pandas为我们设计好了对象的属性，并在设置了name属性值用来进行名字的设定。以下程序可以用来完成该操作。

```
dict_data1 = {
    "Beijing":2200,
    "Shanghai":2500,
    "Shenzhen":1700
}
data1 = pd.Series(dict_data1)
data1
```

```
data1 = pd.Series(dict_data1)
data1.name = "City_Data"
data1.index.name = "City_Name"
print(data1)
```

```
data1.name
```

```
data1.index.name
```

```
data1
```

> 序列的名称，如果是DataFrame的一部分，还包括列名

- 如果用于形成数据帧，序列的名称将成为其索引或列名。每当使用解释器显示序列时，也会使用它。

```
# 使用Series创建DataFrame类型
df = pd.DataFrame(data1)
print(df,type(df))
print("="*20)
# 输入City_Data列的数据和类型
print(df['City_Data'],type(df['City_Data']))
```

- c. copy参数

copy 表示对 data 进行拷贝，默认为 False,仅影响Series和ndarray数组

```
# 数组作为数据源
np_rand = np.arange(1,6)
# 使用数组创建Series
s1 = pd.Series(np_rand)
s1
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int32
```

```
# 改变Series标签为1的值
s1[1] = 50

# 输出Series对象s1
print("s1:",s1)

# 输出数组对象np_rand
print("np_rand:",np_rand)
```

```
s1: 0     1
1    50
2     3
3     4
4     5
dtype: int32
np_rand: [ 1 50  3  4  5]
```

```
# 当源数据非Series和ndarray类型时,
# 数组作为数据源
my_list = [1,2,3,4,5,6]
# 使用数组创建Series
s2 = pd.Series(my_list)
s2
```

```
0    1
1    2
2    3
3    4
4    5
5    6
dtype: int64
```

```
# 改变Series标签为1的值
s2[1] = 50

# 输出Series对象s2
print("s2:",s2)

# 输出列表对象my_list
print("my_list:",my_list)
```

```
s2: 0     1
1    50
2     3
3     4
4     5
5     6
dtype: int64
my_list: [1, 2, 3, 4, 5, 6]
```

```

```

```

```

Series的索引/切片

1.下标索引

类似于 列表索引

```
s = pd.Series(np.random.rand(5))
print(s)
print(s[3], type(s[3]), s[3].dtype)
```

上面的位置索引和标签索引刚好一致,会使用标签索引

当使用负值时,实际并不存在负数的标签索引

2. 标签索引

当索引为object类型时,既可以使用标签索引也可以使用位置索引

Series 类似于固定大小的 dict，把 index 中的索引标签当做 key，而把 Series 序列中的元素值当做 value，然后通过 index 索引标签来访问或者修改元素值。

使用索标签访问单个元素值：

```
s = pd.Series(np.random.rand(5),index=list("abcde"))
print(s["b"], type(s["b"]), s["b"].dtype)
```

使用索引标签访问多个元素值

```
s = pd.Series([6,7,8,9,10],index = ['a','b','c','d','e'])
print(s)
# 注意需要选择多个标签的值,用[[]]来表示(相当于[]中包含一个列表)
print(s[['a','c','d']])
```

```
s1
```

多标签会创建一个新的数组

```
s1 = s[["b","a","e"]]
s1["b"] = 10
print("s1:",s1)
print("s源数据:",s)
```

3. 切片

- Series使用标签切片运算与普通的Python切片运算不同：Series使用标签切片时，其末端是包含的。
- Series使用python切片运算即使用位置数值切片，其末端是不包含。

通过下标切片的方式访问 Series 序列中的数据，示例如下：

```
s = pd.Series(np.random.rand(10))
s
```

```
0    0.558287
1    0.514535
2    0.921289
3    0.106795
4    0.690850
5    0.892483
6    0.998690
7    0.518394
8    0.322016
9    0.006498
dtype: float64
```

```
# 位置索引和标签索引刚好一致,使用切片时,如果是数值会认为是python切片运算,不包含末端
s[1:5]
```

```
1    0.514535
2    0.921289
3    0.106795
4    0.690850
dtype: float64
```

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

print(s[1:4])
# print(s[0])  #位置下标
# print(s['a']) #标签
```

```
b    2
c    3
d    4
dtype: int64
```

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s[:3])
```

```
# 如果想要获取最后三个元素，也可以使用下面的方式：
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s[-3:])
```

通过标签切片的方式访问 Series 序列中的数据，示例如下：

- Series使用标签切片时，其末端是包含的

```
s1= pd.Series([6,7,8,9,10],index = ['a','b','c','d','e'])
s1["b":"d"]
```

```
b    7
c    8
d    9
dtype: int64
```

```
s1= pd.Series([6,7,8,9,10],index = ['e','d','c','b','a'])
s1["c":"a"]
```

```
s1["c":"c"]
```

```
s1= pd.Series([6,7,8,9,10],index = ['e','d','a','b','a'])
```

注意：

在上面的索引方式，我们知道了位置索引和名称索引在index为数值类型时候的不同，

- 当index为数值类型的时候，使用位置索引会抛出keyerror的异常，也就是说当index为数值类型的时候，索引使用的是名称索引。
- 但是在切片的时候，有很大的不同，如果index为数值类型的时候，切片使用的是位置切片。总的来说，当index为数值类型的时候：
- 进行索引的时候，相当于使用的是名称索引；
- 进行切片的时候，相当于使用的是位置切片；

Series数据结构 基本技巧

1. 查看前几条和后几条数据

```
s = pd.Series(np.random.rand(15))
s
```

```

print(s.head()) # 默认查看前5条数据
print(s.head(1)) # 默认查看前1条数据
```

```
print(s.tail()) # 默认查看后5条数据
```

2. 重新索引**: reindex**

使用可选填充逻辑, 使Series符合新索引

将NaN放在上一个索引中没有值的位置。除非新索引等同于当前索引,并且生成新对象。

```
s = pd.Series(np.random.rand(5),index=list("abcde"))
print("============s=========")
print(s)

# 新索引在上一个索引中不存在,生成新对象时,对应的值,设置为NaN
s1 = s.reindex(list("cde12"))
print("============s1=========")
print(s1)

```

```
# 设置填充值
s2 = s.reindex(list("cde12"), fill_value=0)
print(s2)
```

3.对齐运算

是数据清洗的重要过程，可以按索引对齐进行运算，如果没对齐的位置则补NaN，最后也可以填充NaN

```
s1 = pd.Series(np.random.rand(3), index=["Kelly","Anne","T-C"])

s2 = pd.Series(np.random.rand(3), index=["Anne","Kelly","LiLy"])

print("==========s1=========")
print(s1)
print("==========s2=========")
print(s2)
print("==========s1+s2=========")
print(s1+s2)
```

4.删除和添加

- 删除

```
s = pd.Series(np.random.rand(5),index=list("abcde"))
s
```

```
a    0.153331
b    0.048485
c    0.356934
d    0.291750
e    0.992515
dtype: float64
```

```

print(s)
s1 = s.drop("a") # 返回删除后的值,原值不改变 ,默认inplace=False
print(s1)
print(s)
```

```
s = pd.Series(np.random.rand(5),index=list("abcde"))
s1 = s.drop("a",inplace=True) # 原值发生变化,返回None
print(s1)
print(s)

# inplace默认默认为True,返回None
```

- 添加

```
import pandas as pd
# 添加
s1 = pd.Series(np.random.rand(5),index=list("abcde"))
print(s1)

s1["s"] = 100
print(s1)

```

5.检测缺失值

isnull() 和 nonull() 用于检测 Series 中的缺失值。所谓缺失值，顾名思义就是值不存在、丢失、缺少。

- isnull()：如果为值不存在或者缺失，则返回 True。
- notnull()：如果值不存在或者缺失，则返回 False。

其实不难理解，在实际的数据分析任物中，数据的收集往往要经历一个繁琐的过程。在这个过程中难免会因为一些不可抗力，或者人为因素导致数据丢失的现象。这时，我们可以使用相应的方法对缺失值进行处理，比如均值插值、数据补齐等方法。上述两个方法就是帮助我们检测是否存在缺失值。示例如下：

```
import pandas as pd
#None代表缺失数据
s=pd.Series([1,2,5,None])
print(pd.isnull(s))  #是空值返回True
print(pd.notnull(s)) #空值返回False
```


```
import pandas as pd
#None代表缺失数据
s=pd.Series([1,2,5,None])
print(pd.isnull(s))  #是空值返回True
print(pd.notnull(s)) #空值返回False
```


在对数据进行清洗的时候，一般都需要处理数据集中的空值。首先需要查看各列是否存在空值，然后就可以使用 .fillna() 来填补空值或者用.dropna()来丢弃数据表中包含空值的某些行或者列。

对于查看各列是否存在空值，有两种方法：Pandas.DataFrame isna()和isnull()。事实上，这两种方法并没有什么区别，他们做的是相同的事情

然而，在python中，pandas是构建在numpy之上的。在numpy中，既没有na也没有null，而只有NaN(意思是“Not a Number”)，因此，pandas也沿用NaN值。

简单的说：

- numpy用isnan()检查是否存在NaN。
- pandas用.isna()或者.isnull()检查是否存在NaN。

6.填补:fillna

使用指定的值填充,更复杂的再学习DataFrame时再讲解

返回:填充了缺失值的对象,或者当inplace=True时,返回空

```
s=pd.Series([1,2,5,None])
print("==========s========")
print(s)
s1 = s.fillna(0)
print("==========s========")
print(s)
print("==========s1========")
print(s1)
```

```
s=pd.Series([1,2,5,None])
print("==========s========")
print(s)
# 当设置inplace=True ,这将修改此对象上的任何其他视图
s1 = s.fillna(0,inplace=True)
print("==========s========")
print(s)
print("==========s1========")
print(s1)
```

7.删除空数据：

`Series.dropna(axis=0, inplace=False)`

返回已删除缺失值的新序列

- axis:{0或'index'}，默认值为0 只有一个轴可以从中删除值。
- inplace:将修改此对象上的任何其他视图并返回空

```
ser = pd.Series([1., 2., np.nan])
ser
```

```
# 从Series中删除 NA v值
ser.dropna()
```

```
# pd.NaT --- >代表了缺失日期或空日期的值，类似于浮点数的 np.nan
ser = pd.Series([np.NaN, 2, pd.NaT, '', None, 'I stay'])
print(ser)
print("=============删除数据将返回的内容=============")
ser.dropna()
```

空字符串不被视为NA值。None被视为NA值。

```

```

```

```

课堂作业

1.分别有字典/数组的方式,创建以下要求的Series

Zhangsan 90.0

wangwu 89.5

lilei 68.0

Name:作业1, dtye:float64

```
2.创建一个Series,包含1个元素,且每个值为0-100的均匀分布随机值,index为a-j,请分别筛选出
- ① 标签为b,c的值为多少
- ②.Series中第4到第6个值是那些
- ③ Series中大于50 的值有那些
```

3.创建以下Series 并按照要求修改得到的结果

创建s:

b 1

c 2

d 3

e 4

f 5

g 6

h 7

i 8

j 9

dtype:int32

---

s修改后:

a 100

c 2

d 3

e 100

f 100

g 6

i 8

j 9

dtype:int32