**数据结构**

**Series**

Series类对象类似于一维数组，由索引和数据构成。

#Series类构造方法

pandas.Series(data=None,index=None,dtype=None,name=None,copy=False,fastpath=False)

#data:表示传入的数据，可以是ndarry，列表，字典等;

#index:表示传入的索引，必须是唯一值，且与数据的长度相同。若没有传入索引，则创建的Series类对象会自动生成0-N的整数索引

#dtype：表示数据的类型。若未指定数据类型，pandas会根据传入的数据自动推断数据类型

#name:表示Series类对象的名称。

#copy:表示是否复制数据，默认未False

#简单实例

ser\_obj=pd.Series(['python','java','php'])

# 0 python

# 1 java

# 2 php

#自定义索引的实例

ser\_obj=pd.Series(['python','java','php'],index=['one','two','three'])

# one python

# two java

# three php

#字典的实例

ser\_obj=pd.Series({"one":'python',"two":'java',"three":'php'})

# one python

# two java

# three php

#时间序列

date\_index=pd.to\_datetime(['20180820','20180828','20180908'])#时间索引

date\_ser=pd.Series([11,22,33],index=date\_index)

# 2018-08-20 11

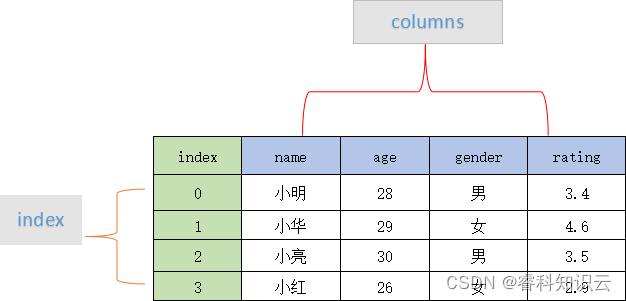
# 2018-08-28 22

# 2018-09-08 33

**DataFrame**

DataFrame类对象的结构类似于二维数组或表格。

(两组索引是行索引和列索引)



#DataFrame类构造方法

pandas.DataFrame(data=None,index=None,columns=None,dtype=None,copy=None)

#data:表示传入的数据，可以是ndarry，列表，字典等;

#index:表示行索引，默认生成0-N的整数索引

#columns:表示列索引，默认生成0-N的整数索引

#dtype:表示数据的类型

#copy:表示是否复制数据,默认未False

#简单创建二维数列实例

demo\_arr=np.array([['a','b','c'],['d','e','f']])

df\_obj=pd.DataFrame(demo\_arr)

# 0 1 2

# 0 a b c

# 1 d e f

#指定行列索引的实例

demo\_arr=np.array([['a','b','c'],['d','e','f']])

df\_obj=pd.DataFrame(demo\_arr，index=['row\_1','row\_2'],columns=['col\_1','col\_2','col\_3'])

# col\_1 col\_2 col\_3

# row\_1 a b c

# row\_2 d e f

#用字段列表化为df对象

new\_df = pd.DataFrame.from\_dict(字段列表1,字段列表2...)#传入字段数据

**索引操作**

**索引对象**

index类的常见子类及其说明

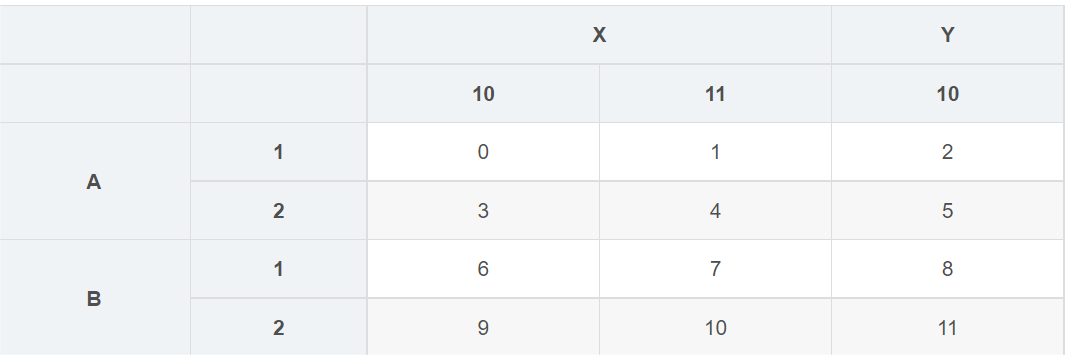
|  |  |
| --- | --- |
| 子类 | 说明 |
| Int64Index | 整数索引 |
| Float64Index | 浮点型索引 |
| DatetimeIndex | 纳秒级时间索引 |
| PeriodIndex | 时间间隔索引 |
| MultiIndex | 分层索引  (只有这个索引用于多层结构的数据) |

**MultiIndex类**

Series的分层索引实例



dataframe的分层索引实例



**创建MultiIndex类的方法**

|  |  |
| --- | --- |
| 方法 | 说明 |
| from\_tuples() | 根据元组创建分层索引 |
| from\_arrays() | 根据数组创建分层索引 |
| from\_product() | 从集合的笛卡尔乘积中创建分层索引 |
| from\_frame() | 根据dataframe对象创建分层索引 |

根据元组创建分层索引的实例

tuple\_clo=[('a',0),('a',1)('b',2),('b',2)]

tuple\_row=[('a',0),('a',1)('b',2),('b',2)]

multi\_index\_col=pd.MultiIndex.from\_tuples(tuples=tuple\_clo)

multi\_index\_row=pd.MultiIndex.from\_tuples(tuples=tuple\_row)

data=[['A','B','C','D'],['E','F','G','H'],['I','J','K','L'],['M','N','O','P']]

df=pd.DateFrame(data,index=multi\_index\_col,columns=multi\_index\_row)

# a b

# 0 1 2 2

# a 0 A B C D

# 1 E F G H

# b 2 I J K L

# 2 M N O P

**使用单层索引访问数据**

**使用[]访问数据**

变量[索引]

#Series对象

ser=pd.Series(['A','B','C','D'],index=['one','two','three'])

ser['one']

#A

#DataFrame对象

df=pd.DataFrame([[0,2,3],[0,4,1],[10,20,30]],index=[4,5,6],columns=['A','B','C'])

# A B C

# 4 0 2 3

# 5 0 4 1

# 6 10 20 30

df['A']

# 4 0

# 5 0

# 6 10

**使用loc和iloc访问数据**

变量.loc[索引]必须是自定义的标签索引

变量.iloc[索引]必须是自动生成的整数索引

ser.loc['two']

ser.iloc[1]

**使用at和iat访问数据**

变量.at[行索引,列索引]必须是自定义的标签索引

变量.iat[行索引,列索引]必须是自动生成的整数索引

来获取dataframe的单个元素

df.at[5,'B']

df.iat[1,1]

**使用分层索引访问数据**

**使用[]访问数据**

变量[第一层索引]

变量[第一层索引][第二层索引]

#Series实例

mult\_series=pd.Series([95,103,80,80,90,91,91],index=[['计算机专业','计算机专业','计算机专业','计算机专业','体育专业','体育专业','体育专业'],['物联网工程','软件工程','网络安全','信息安全','体育教育','休闲体育','运动康复']])

# 计算机专业 物联网工程 95

# 软件工程 103

# 网络安全 80

# 信息安全 80

# 体育专业 体育教育 90

# 休闲体育 91

# 运动康复 91

mult\_series['计算机专业']

#物联网工程 95

#软件工程 103

#网络安全 80

#信息安全 80

mult\_series['计算机专业']['软件工程']

#103

**使用loc和iloc访问数据**

变量.loc[第一层索引] #访问第一层索引对应的数据

变量.loc[第一层索引][第二层索引] #访问第二层索引对应的数据

变量.iiloc[整数索引] #访问整数索引对应的数据

**重新设定索引**

reindex(labels=None,index=None,colums=None,axis=None,method=None,copy=True,level=None,fill\_value=nan,limit=None,tolerance=None)

df.reindex(['a','b','c'])#这是对行索引重新设置

df.reindex(columns=['a','b','c'])#这是对列重新设置

df.reindex(['a','b','c']，fill\_value='missing')#这是对行索引重新设置,且用指定值进行缺失值的填充

**其他函数**

df.head(5)#查看前5行

df.tail(5)#查看后5行

#设置最大显示列数

pd.set\_option('display.max\_columns',20)

#设置最大显示行数

pd.set\_option('display.max\_rows',10)

df.loc[:,['time','min','max']]#看所有行的这三列数据

df[['time','min','max']][:5]#选择列的方括号多一对，就可以选择多列了

df.loc[(df.cloud == '多云'),['time','cloud']]#选出符合条件的行(或:|，且:&)

df[~(df['cloud']=='2')].head(5)#获取符合条件(df['cloud']!='2')的前5个

df[df['cloud'].between('1','2',inclusive=True)]#获取cloud等于1或2的行

df[df['cloud'].isin(['1','2'])]#获取cloud等于1或2的行

**数据排序**

**按索引排序**

sort\_index(axis=0,level=None,ascending=True,kind='quicksort')

axis：表示排序方向，0是按行，1是按列

level：表示按哪个索引层级排序，默认为None

ascending：True为升序，False为降序

kind：表示排序算法，默认为quicksort

**按值排序**

sort\_values(by,axis=0,ascending=True,kind='quicksort',na\_position='last',ignore\_index=False)

by：表示根据指定的列索引名(axis=0或index)或行索引名(axis=1或columns)进行排序

axis：表示轴编号(排序的方向)，0代表按行排序，1代表按列排序

ascending：True为升序，False为降序

na\_position：last表示缺省值显示在末尾，first表示缺省值显示在首位

**统计计算与统计描述**

**统计计算**

常见的统计计算方法及其说明

ser.sum()#计算和

ser.mean()#计算平均值

ser.max()#计算最大值

ser.min()#计算最小值

ser.idxmax()#计算最大索引

ser.idxmin()#计算最小索引

ser.count()#计算非NaN值的个数

ser.var()#计算方差

ser.std()#计算样本差

ser.cumsum()#计算累计和

ser.cumprod()#计算累计积

ser.cummin()#累计最小值

ser.cummax()#累计最大值

**统计描述**

df.describe()#显示常用统计值的值

**绘制图表**

plot(x=None,y=None,kind='line',figsize=None,title=None,grid=None,xlabel=None,ylabel=None,rot=None)

x,y：表示x轴和y轴的数据

kind：表示绘图的类型(line是折线图，bar是柱状图，barh是条形图，hist是直方图，box是箱型图，pie是饼图)

figsize：表示图表尺寸的大小，需要元组数据(其中有两元素分别为高度和宽度)

title：表示图表的标题

grid：True为显示网格线，False为不显示网格线

xlabel：表示x轴的标签

ylabel：表示y轴的标签

rot：表示轴标签旋转的角度

#实例

df.plot(kind='box',ylabel='销售额(万元)')

**其他**

**遍历对象中的所有元素并用函数进行处理**

map和apply

df.apply(lambda x: sum(x.isnull())/len(x),axis= 0) #缺失比例

df2['性别'] = df2['gender'].map({'0':'女','1':'男','2':'未知'})#效果和f方法一样

**df对象增加数据**

df.insert(0,'sunlight',sun\_0)#0表示第一个列名，'sunlight'表示是列名，sun\_0是要传入的列

**改列名或行索引名称**

df.rename(columns={'max':'max\_0','min':'min\_0'},inplace=True)#改列名

df.rename(index={1:'one',0:'zero'},inplace=True)#修改行索引名称

**重置索引**

df.reset\_index(drop=True,inplace=True)#重置索引