pandas基础教程四\_索引对象

大家好，这里是Python知识学堂，在之前关于pandas的基础教程中我们学习了Series和DataFrame的数据结构，Series和DataFrame的数据结构中有一个索引的概念，今天我们来探讨一下。

一、什么是索引对象

索引对象的理解很简单，索引对象的作用主要是负责管理轴标签和其他元数据的。构建Series和DataFrame时，在pandas中定义的**Index**类来表示基本索引对象。我们来看两个打印，分别是索引对象所在类名输出，一个是其\_\_doc\_\_属性。

import pandas as pd  
print(pd.Index)  
print(pd.Index.\_\_doc\_\_)

输出为：

<class 'pandas.core.indexes.base.Index'>

Immutable ndarray implementing an ordered, sliceable set. The basic object

storing axis labels for all pandas objects.

Parameters

----------

data : array-like (1-dimensional)

dtype : NumPy dtype (default: object)

If dtype is None, we find the dtype that best fits the data.

If an actual dtype is provided, we coerce to that dtype if it's safe.

Otherwise, an error will be raised.

copy : bool

Make a copy of input ndarray.

name : object

Name to be stored in the index.

tupleize\_cols : bool (default: True)

When True, attempt to create a MultiIndex if possible.

See Also

--------

RangeIndex : Index implementing a monotonic integer range.

CategoricalIndex : Index of :class:`Categorical` s.

MultiIndex : A multi-level, or hierarchical Index.

IntervalIndex : An Index of :class:`Interval` s.

DatetimeIndex : Index of datetime64 data.

TimedeltaIndex : Index of timedelta64 data.

PeriodIndex : Index of Period data.

Int64Index : A special case of :class:`Index` with purely integer labels.

UInt64Index : A special case of :class:`Index` with purely unsigned integer labels.

Float64Index : A special case of :class:`Index` with purely float labels.

Notes

-----

An Index instance can \*\*only\*\* contain hashable objects

Examples

--------

>>> pd.Index([1, 2, 3])

Int64Index([1, 2, 3], dtype='int64')

>>> pd.Index(list('abc'))

Index(['a', 'b', 'c'], dtype='object')

二、索引对象的获取

我们来看一下上述输出的最后有两个例子，即使用Pd.Index()方法就能创建一个索引对象。细心的同学可能已经发现，第一个输出dtype='int64'，第二个dtype='object'，第二个输出难道不是dtype='string'，要回答这个问题，我们先看另外一个例子：

import pandas as pd  
data\_test = pd.DataFrame({**"Yes\_No"**: [True,False],  
 **"Value"**: [1.1,2],  
 **"Type"**: [**'Small'**,**'large'**]  
 })  
print(data\_test.dtypes)

输出为：

Yes\_No bool

Value float64

Type object

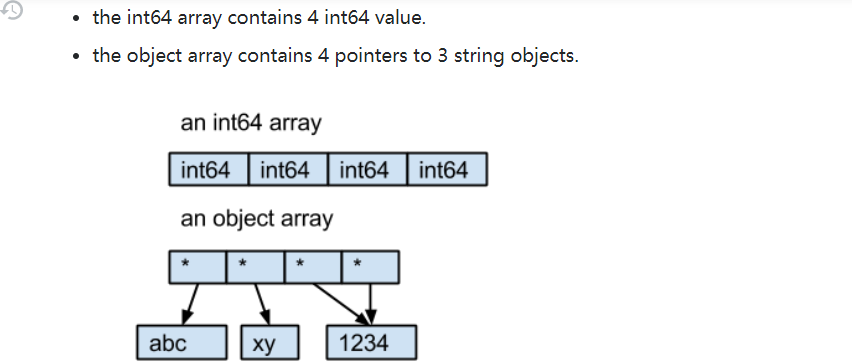
dtype: object

“Type”行数据的类型为object,并不是String,跟索引对象中的一致，这是为什么呢？这个问题的答案在这：

https://stackoverflow.com/questions/21018654/strings-in-a-dataframe-but-dtype-is-object

**简单的说就是：**

dtype对象来自NumPy，它描述ndarray中元素的类型。 ndarray中的每个元素都必须具有相同的字节大小。对于int64和float64，它们是8个字节。但是对于字符串，字符串的长度不是固定的，Pandas使用了一个对象ndarray来保存指向对象的指针，而不是直接在ndarray中保存字符串的字节。因此这种ndarray的dtype是object。



那么常见的索引对象的类型有哪些呢？

弄清楚这个问题，我们就可以看一下索引对象怎么获取了：

import pandas as pd  
df\_Test = pd.DataFrame(  
 {**'time'**:[**'2021-05-11'**,**'2021-05-12'**,**'2021-05-13'**],**'count'**:[1,2,3]},  
 index=[**'a'**,**'b'**,**'c'**]  
)  
print(df\_Test)  
print(**'获取的索引对象为：'**,df\_Test.index)  
print(**'获取的索引对象值为：'**,df\_Test.index.values)

输出为：

time count

a 2021-05-11 1

b 2021-05-12 2

c 2021-05-13 3

获取的索引对象为： Index(['a', 'b', 'c'], dtype='object')

获取的索引对象值为： ['a' 'b' 'c']

即使用df.index就能很容易的获取索引对象，通过values就能很容易的获取索引对象数据。

三、索引的可重复性

索引对象的索引是可以重复的，这句话是什么意思呢？我们在构建DataFrame的时候传入的index是可以重复的，可以这样: index=[**'a'**,**'b'**,**'c'**]，可以这样index=[**'a'**,**'b'**,**'a'**]，还可以这样

index=[**'a'**,**'a'**,None]等等。

import pandas as pd  
df\_Test\_2 = pd.DataFrame(  
 {**'time'**:[**'2021-05-11'**,**'2021-05-12'**,**'2021-05-13'**],**'count'**:[1,2,3]},  
 index=[**'a'**,**'a'**,**'b'**]  
)  
print(df\_Test\_2)  
print(df\_Test\_2.index.is\_unique)  
print(df\_Test\_2.index.isna())

输出为：

time count

a 2021-05-11 1

a 2021-05-12 2

b 2021-05-13 3

False

[False False False]

第一行输出包含重复索引的dataframe。

第二行代码使用is\_unique方法来判断索引是否是唯一不重复的，返回bool值。

第三行代码使用isna方法来判断索引是否是None值, index=[**'a'**,**'a'**,None]。

看了代码和解释是不是觉得这个还是很简单的。

四、索引的不可修改性

也就是说这个索引是不支持进行修改的，但是可以进行重置。换句话说索引是不支持部分修改的。我们来看一下：

import pandas as pd  
df\_Test\_3 = pd.DataFrame(  
 {**'time'**:[**'2021-05-11'**,**'2021-05-12'**,**'2021-05-13'**],**'count'**:[1,2,3]},  
 index = [0,1,2]  
)  
print(df\_Test\_3.index[2])  
df\_Test\_3.index[2] = 100

print()输出2，之后df\_Test\_3.index[2] = 100报不支持修改的错：

raise TypeError("Index does not support mutable operations")

TypeError: Index does not support mutable operations

那么该如何进行这个索引的修改的呢？毕竟在工程实践中我们有这样的要求，接下来我们介绍该如何进行重置索引：

**直接赋值替换**

import pandas as pd  
df\_Test\_3 = pd.DataFrame(  
 {**'time'**:[**'2021-05-11'**,**'2021-05-12'**,**'2021-05-13'**],**'count'**:[1,2,3]},  
 index = [0,1,2]  
)df\_Test\_3.index=[**'a'**,**'b'**,**5**]  
print(df\_Test\_3)

输出:

time count

a 2021-05-11 1

b 2021-05-12 2

5 2021-05-13 3

这样就可以将数据的索引进行“改变了”，注意看我使用的是[**'a'**,**'b'**,**5**]，这个列表的里面的数据的数据类型是不一样的。

**使用set\_index()，reset\_index()和reindex()方法**

*DataFrame.set\_index : Set row labels.  
DataFrame.reset\_index : Remove row labels or move them to new columns.  
DataFrame.reindex : Change to new indices or expand indices.*

**set\_index()方法的定义如下：**

def set\_index(  
 self, keys, drop=True, append=False, inplace=False, verify\_integrity=False  
)

keys：类似标签或数组的标签或标签/数组的列表  
drop：默认为True删除要用作新索引的列  
append：True追加到现有索引。

inplace：是否在原数据中修改  
verify\_integrity：True检测新的索引是否重复。

import pandas as pd  
df\_Test = pd.DataFrame(  
 {**'time'**:[**'2021-05-11'**,**'2021-05-12'**,**'2021-05-13'**],**'count'**:[1,2,3]},index=[1,2,3])  
  
N\_df\_Test = df\_Test.set\_index([**'time'**],drop=True)  
print(N\_df\_Test)  
  
new\_index = N\_df\_Test.index.rename(**'这是索引哦'**)  
N\_df\_Test.index=new\_index  
print(N\_df\_Test)

输出为：

count

time

2021-05-11 1

2021-05-12 2

2021-05-13 3

count

这是索引哦

2021-05-11 1

2021-05-12 2

2021-05-13 3

上述例子大家要仔细的研究一下，另外我只举例使用**drop=True**这个参数，大家可以试一下使用其他的参数会有什么不同之处。

**reset\_index()方法的定义如下：**  
def reset\_index(

level=None, drop=False, inplace=False, col\_level=0, col\_fill=”

)

reset\_index可以还原索引，重新变为默认的整型索引,和set\_index()有一种相反的感觉

level：控制了具体要还原的那个等级的索引  
drop:为False则索引列会被还原为普通列，否则会丢失

import pandas as pd  
df\_Test = pd.DataFrame(  
 {**'time'**:[**'2021-05-11'**,**'2021-05-12'**,**'2021-05-13'**],**'count'**:[1,2,3]},  
 index=list(**'abc'**))  
print(df\_Test)  
print(df\_Test.reset\_index())  
print(df\_Test.reset\_index(drop=True))

输出：

time count

a 2021-05-11 1

b 2021-05-12 2

c 2021-05-13 3

index time count

0 a 2021-05-11 1

1 b 2021-05-12 2

2 c 2021-05-13 3

time count

0 2021-05-11 1

1 2021-05-12 2

2 2021-05-13 3

大家好好体会drop=True和False的这两种情况。

这个情景非常的常见，因为在进行数据清洗的时候，我们会清洗掉一些包含NaN行的一些数据，这会造成索引不在是连贯的数值了（0,1,2…n）的数值了，这个时候我们使用上述操作就可以使得索引是连贯的。

**reset\_index()方法的定义如下：**

DataFrame.reindex(

labels=None, index=None, columns=None, axis=None, method=None, copy=True, level=None, fill\_value=nan, limit=None, tolerance=None

)

reindex方法可以为series和dataframe添加或者删除索引,即可以重新定义索引。

如果定义的索引没有匹配的数据，默认将已缺失值填充。

对于Series和DataFrame两个数据类型都可以通过fill\_value参数填充默认值，也可以通过method参数设置填充方法。而method参数可选以下几种：

None (默认): 不做任何填充；

pad / ffill: 用上一行的有效数据来填充；

backfill / bfill: 用下一行的有效数据来填充；

nearest: 用临近行的有效数据来填充。

我们来看几个例子就明白了：

import pandas as pd  
df\_Test\_3 = pd.DataFrame(  
 {**'time'**:[**'2021-05-11'**,**'2021-05-12'**,**'2021-05-13'**],**'count'**:[1,2,3]},  
 index = [0,1,2]  
)  
  
print(df\_Test\_3)  
print(df\_Test\_3.reindex([1,2,3,4,5]))  
print(df\_Test\_3.reindex([**'a'**,**'b'**,5]))

输出为：

time count

0 2021-05-11 1

1 2021-05-12 2

2 2021-05-13 3

time count

1 2021-05-12 2.0

2 2021-05-13 3.0

3 NaN NaN

4 NaN NaN

5 NaN NaN

time count

a NaN NaN

b NaN NaN

5 NaN NaN

可以发现，新增的索引数据不在原索引对象中将引入新NaN值。

使用method参数的时候：

print(df\_Test\_3.reindex([1,2,3,4,5],method=**'nearest'**))

输出：

time count

1 2021-05-12 2

2 2021-05-13 3

3 2021-05-13 3

4 2021-05-13 3

5 2021-05-13 3

另外，值得注意的是reindex()可以修改列名：

print(df\_Test\_3.reindex(columns=[**'New\_time'**,**'New\_count'**]))

输出为：

time New\_count

0 2021-05-11 NaN

1 2021-05-12 NaN

2 2021-05-13 NaN

由于新的列 New\_count没有定义值所以填充为NaN了。

竟然还可以进行取值的操作，好比切片似的：

print(df\_Test\_3.reindex(index = [0,2],columns=[**'time'**,**'count'**]))

输出为：

time count

0 2021-05-11 1

2 2021-05-13 3

四、索引对象的方法和属性

|  |  |
| --- | --- |
| **方法** | **说明** |
| append | 索引对象的连接 |
| delete | 删除传入的值 |
| diff | 计算差集 |
| intersection | 计算交集 |
| union | 计算并集 |
| isin | 计算一个指示各值是否都包含在参数集合中的布尔型数值 |
| unique | 判断唯一性 |
| insert | 将元素插入到指定位置 |
| is\_unique | 检测索引值有误重复 |

我们举例几种的一些方法和属性：

import pandas as pd  
*# 创建两个索引对象*index\_1 = pd.Index([**'a'**,**'a'**,**'b'**,**'c'**,**'d'**,1,2,3,4,5,6])  
index\_2 = pd.Index([1,2,3,4,5,6])  
*# 创建一个series*series\_1 = pd.Series(range(11),index=index\_1)  
  
*# 判断目标索引是否存在*print(**'a'** in index\_1)  
print(1 in index\_1)  
  
*# 长度*print(len(index\_1))  
  
*# 切片操作,跟列表的切片是一样的*print(index\_1[:3])  
  
*# 两个索引对象的交集*print(index\_1.intersection(index\_2))  
  
*# 两个索引对象的差集*print(index\_1.difference(index\_2))  
print(index\_2.difference(index\_1))

*# 索引对象的连接*print(index\_1.append(index\_2))  
  
*# 删除索引(传入列表，列表中的值为删除的位置)*print(index\_1.delete([0]))  
print(index\_1.delete([0,1,3]))  
  
*# 插入*print(index\_1.insert(0,**'aaa'**))  
  
*# 去重*print(index\_1.is\_unique)  
print(index\_1.unique())

以上几种是比较常见Index的方法和属性举例，很好理解，这里就不解释了。

五、总结

本次文章我们讲解了在Pandas中最为常见的一个知识点——索引对象。当然了本次我们所见到的索引对象是一维度的，下次我们将讲解多重索引对象，所以小伙伴们先要将本次的文章的内容好好消化。MultiIndex我们下期见，先给大家看一下文档：

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.MultiIndex.html?highlight=multiindex#pandas.MultiIndex