

类

职业 (/www.jikexueyuan.com/zhiye/)

首页 (https://www.jikexueyuan.com/)

(//passport.jikexueyuan.com/sso/login)

错误和异常

> Wiki (http://wiki.jikexueyuan.com)

模块

课程 (/www.jikexueyuan.com/course/)

> 后端开发 (http://wiki.jikexueyuan.com/list/back-end)

保存数据

> Python (http://wiki.jikexueyuan.com/list/python)

实战

企业 (/www.jikexueyuan.com/partner)

注册

用Tornado做网站

(//passport.jikexueyuan.com/sso/reg_phone)

科学计算

社区 (/wiki.jikexueyuan.com/)

> 《从零开始学 Python》(第二版) (http://wiki.jikexueyuan.com/project/start-learning-python)

为计算做准备 (http://wiki.jikexueyuan.com/project/start-learning-python/310.html)

Pandas使用(1) (http://wiki.jikexueyuan.com/project/start-learning-

老齐 (https://github.com/qiwsir) · 更新于 2018-07-02 14:00:48

PDF版 (//passport.jikexueyuan.com/sso/login)

ePub版 (//passport.jikexueyuan.com/sso/login)

Pandas 使用 (1)

Pandas 是基于 NumPy 的一个非常好用的库，正如名字一样，人见人爱。之所以如此，就在于不论是读取、处理数据，用它都非常简单。

基本的数据结构

Pandas 有两种自己独有的基本数据结构。读者应该注意的是，它固然有着两种数据结构，因为它依然是 Python 的一个库，所以，Python 中有的数据类型在这里依然适用，也同样还可以使用类自己定义数据类型。只不过，Pandas 里面又定义了两种数据类型：Series 和 DataFrame，它们让数据操作更简单了。

以下操作都是基于：

```
In [1]: from pandas import Series, DataFrame
import pandas as pd
```

为了省事，后面就不在显示了。并且如果你跟我一样是使用 ipython notebook，只需要开始引入模块即可。

Series

Series 就如同列表一样，一系列数据，每个数据对应一个索引值。比如这样一个列表：[9, 3, 8]，如果跟索引值写到一起，就是：

data	9	3	8
index	0	1	2

这种样式我们已经熟悉了，不过，在有些时候，需要把它竖过来表示：

index	data
0	9
1	3
2	8

上面两种，只是表现形式上的差别罢了。

Series 就是“竖起来”的 list：

```
In [2]: s = Series([100, "PYTHON", "Soochow", "Qiwsir"])
s
```

```
Out[2]: 0      100
        1    PYTHON
        2    Soochow
        3    Qiwsir
```

另外一点也很像列表，就是里面的元素的类型，由你任意决定（其实是由需要来决定）。

这里，我们实质上创建了一个 Series 对象，这个对象当然就有其属性和方法了。比如，下面的两个属性依次可以显示 Series 对象的数据值和索引：

```
In [3]: s.values
```

```
Out[3]: array([100, 'PYTHON', 'Soochow', 'Qiwsir'], dtype=object)
```

```
In [4]: s.index
```

```
Out[4]: Int64Index([0, 1, 2, 3], dtype=int64)
```

列表的索引只能是从 0 开始的整数，Series 数据类型在默认情况下，其索引也是如此。不过，区别于列表的是，Series 可以自定义索引：

```
s2 = Series([100, "PYTHON", "Soochow", "Qiwsir"], index=["mark", "title", "university", "name"])
s2
```

```
mark      100
title     PYTHON
university Soochow
name      Qiwsir
```

```
In [6]: s2.index
```

```
Out[6]: Index(['mark', 'title', 'university', 'name'], dtype=object)
```

自定义索引，的确比较有意思。就凭这个，也是必须的。

每个元素都有了索引，就可以根据索引操作元素了。还记得 list 中的操作吗？Series 中，也有类似的操作。先看简单的，根据索引查看其值和修改其值：

```
In [7]: s2["name"]
```

```
Out[7]: 'Qiwsir'
```

```
In [8]: s2["name"] = "AOI"
```

```
In [9]: s2
```

```
Out[9]: mark          100
        title         PYTHON
        university    Soochow
        name          AOI
```

这是不是又有点类似 dict 数据了呢？的确如此。看下面就理解了。

读者是否注意到，前面定义 Series 对象的时候，用的是列表，即 Series() 方法的参数中，第一个列表就是其数据值，如果需要定义 index，放在后面，依然是一个列表。除了这种方法之外，还可以用下面的方法定义 Series 对象：

```
In [15]: sd = {"python":8000, "c++":8100, "c#":4000}
         s4 = Series(sd)
         s4
```

```
Out[15]: c#          4000
         c++         8100
         python      8000
```

现在是否理解为什么前面那个类似 dict 了？因为本来就是可以这样定义的。

这时候，索引依然可以自定义。Pandas 的优势在这里体现出来，如果自定义了索引，自定的索引会自动寻找原来的索引，如果一样的，就取原来索引对应的值，这个可以简称为“自动对齐”。

```
In [19]: s6 = Series(sd, index=["java", "python", "c++", "c#"])
s6
```

```
Out[19]: java      NaN
         python    8000
         c++       8100
         c#        4000
```

在 sd 中，只有 'python':8000, 'c++':8100, 'c#':4000，没有"java"，但是在索引参数中有，于是其它能够“自动对齐”的照搬原值，没有的那个"java"，依然在新 Series 对象的索引中存在，并且自动为其赋值 NaN。在 Pandas 中，如果没有值，都对齐赋给 NaN。来一个更特殊的：

```
In [17]: ilst = ["java", "perl"]
s5 = Series(sd, index=ilst)
s5
```

```
Out[17]: java      NaN
         perl      NaN
```

新得到的 Series 对象索引与 sd 对象一个也不对应，所以都是 NaN。

Pandas 有专门的方法来判断值是否为空。

```
In [20]: pd.isnull(s6)
```

```
Out[20]: java      True  
         python    False  
         c++       False  
         c#        False
```

```
In [21]: pd.notnull(s6)|
```

```
Out[21]: java      False  
         python    True  
         c++       True  
         c#        True
```

此外，Series 对象也有同样的方法：

```
In [22]: s6.isnull()
```

```
Out[22]: java      True  
         python    False  
         c++       False  
         c#        False
```

其实，对索引的名字，是可以从新定义的：

```
In [38]: s6.index = ["p1", "p2", "p3", "p4"]  
s6
```

```
Out[38]: p1      NaN  
         p2     8000  
         p3     8100  
         p4     4000
```

对于 Series 数据，也可以做类似下面的运算（关于运算，后面还要详细介绍）：

```
In [10]: s3 = Series([3,9,4,7], index=['a','b','c','d'])  
s3
```

```
Out[10]: a    3  
        b    9  
        c    4  
        d    7
```

```
In [11]: s3[s3 > 5]
```

```
Out[11]: b    9  
        d    7
```

```
In [12]: s3 * 5
```

```
Out[12]: a    15  
        b    45  
        c    20  
        d    35
```

```
In [28]: s5 + s6
```

```
Out[28]: c#      NaN  
        c++     8188  
        java     NaN  
        python  8030
```

上面的演示中，都是在 ipython notebook 中进行的，所以截图了。在学习 Series 数据类型同时了解了 ipython notebook。对于后面的所有操作，读者都可以在 ipython notebook 中进行。但是，我的讲述可能会在 Python 交互模式中进行。

DataFrame

DataFrame 是一种二维的数据结构，非常接近于电子表格或者类似 mysql 数据库的形式。它的竖行称之为 columns，横行跟前面的 Series 一样，称之为 index，也就是说可以通过 columns 和 index 来确定一个主句的位置。（有人把 DataFrame 翻译为“数据框”，是不是还可以称之为“筐”呢？向里面装数据嘛。）

	c1	c2	c3	c4
ix1				
ix2				
ix3		(c2, ix3)		
ix4				

下面的演示，是在 Python 交互模式下进行，读者仍然可以在 ipython notebook 环境中测试。

```
>>> import pandas as pd
>>> from pandas import Series, DataFrame

>>> data = {"name":["yahoo","google","facebook"], "marks":[200,400,800], "price":[9, 3, 7]}
>>> f1 = DataFrame(data)
>>> f1
   marks  name  price
0    200  yahoo     9
1    400  google     3
2    800 facebook     7
```

这是定义一个 DataFrame 对象的常用方法——使用 dict 定义。字典的“键”（"name", "marks", "price"）就是 DataFrame 的 columns 的值（名称），字典中每个“键”的“值”是一个列表，它们就是那一竖列中的具体填充数据。上面的定义中没有确定索引，所以，按照惯例（Series 中已经形成的惯例）就是从 0 开始的整数。从上面的结果中很明显表示出来，这就是一个二维的数据结构（类似 excel 或者 mysql 中的查看效果）。

上面的数据显示中，columns 的顺序没有规定，就如同字典中键的顺序一样，但是在 DataFrame 中，columns 跟字典键相比，有一个明显不同，就是其顺序可以被规定，向下面这样做：


```
>>> f2 = DataFrame(data, columns=['name','price','marks'])
>>> f2
```

	name	price	marks
0	yahoo	9	200
1	google	3	400
2	facebook	7	800

跟 Series 类似的，DataFrame 数据的索引也能够自定义。

```
>>> f3 = DataFrame(data, columns=['name', 'price', 'marks', 'debt'], index=['a','b','c','d'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/pymodules/python2.7/pandas/core/frame.py", line 283, in __init__
    mgr = self._init_dict(data, index, columns, dtype=dtype)
  File "/usr/lib/pymodules/python2.7/pandas/core/frame.py", line 368, in _init_dict
    mgr = BlockManager(blocks, axes)
  File "/usr/lib/pymodules/python2.7/pandas/core/internals.py", line 285, in __init__
    self._verify_integrity()
  File "/usr/lib/pymodules/python2.7/pandas/core/internals.py", line 367, in _verify_integrity
    assert(block.values.shape[1:] == mgr_shape[1:])
AssertionError
```

报错了。这个报错信息就太不友好了，也没有提供什么线索。这就是交互模式的不利之处。修改之，错误在于 index 的值——列表——的数据项多了一个，data 中是五行，这里给出了四个项（['a','b','c','d']）。

```
>>> f3 = DataFrame(data, columns=['name', 'price', 'marks', 'debt'], index=['a','b','c'])
>>> f3
```

	name	price	marks	debt
a	yahoo	9	200	NaN
b	google	3	400	NaN
c	facebook	7	800	NaN

读者还要注意观察上面的显示结果。因为在定义 f3 的时候，columns 的参数中，比以往多了一项('debt')，但是这项在 data 这个字典中并没有，所以 debt 这一竖列的值都是空的，在 Pandas 中，空就用 NaN 来代表了。

定义 DataFrame 的方法，除了上面的之外，还可以使用“字典套字典”的方式。

```
>>> newdata = {"lang":{"firstline":"python","secondline":"java"}, "price":{"firstline":8000}}
>>> f4 = DataFrame(newdata)
>>> f4
```

	lang	price
firstline	python	8000
secondline	java	NaN

在字典中就规定好数列名称（第一层键）和每横行索引（第二层字典键）以及对应的数据（第二层字典值），也就是在字典中规定好了每个数据格子中的数据，没有规定的都是空。

```
>>> DataFrame(newdata, index=["firstline","secondline","thirdline"])
```

	lang	price
firstline	python	8000
secondline	java	NaN
thirdline	NaN	NaN

如果额外确定了索引，就如同上面显示一样，除非在字典中有相应的索引内容，否则都是 NaN。

前面定义了 DataFrame 数据（可以通过两种方法），它也是一种对象类型，比如变量 f3 引用了一个对象，它的类型是 DataFrame。承接以前的思维方法：对象有属性和方法。

```
>>> f3.columns
Index(['name', 'price', 'marks', 'debt'], dtype=object)
```

DataFrame 对象的 columns 属性，能够显示素有的 columns 名称。并且，还能用下面类似字典的方式，得到某竖列的全部内容（当然包含索引）：

```
>>> f3['name']
a      yahoo
b      google
c      facebook
Name: name
```

这是什么？这其实就是一个 Series，或者说，可以将 DataFrame 理解为是有一个一个的 Series 组成的。

一直耿耿于怀没有数值的那一列，下面的操作是统一给那一列赋值：

```
>>> f3['debt'] = 89.2
>>> f3
```

	name	price	marks	debt
a	yahoo	9	200	89.2
b	google	3	400	89.2
c	facebook	7	800	89.2

除了能够统一赋值之外，还能够“点对点”添加数值，结合前面的 Series，既然 DataFrame 对象的每竖列都是一个 Series 对象，那么可以先定义一个 Series 对象，然后把它放到 DataFrame 对象中。如下：

```
>>> sdebt = Series([2.2, 3.3], index=["a","c"])    #注意索引
>>> f3['debt'] = sdebt
```

将 Series 对象(sdebt 变量所引用) 赋给 f3['debt']列，Pandas 的一个重要特性——自动对齐——在这里起作用了，在 Series 中，只有两个索引 ("a","c")，它们将和 DataFrame 中的索引自动对齐。于是乎：

```
>>> f3
```

	name	price	marks	debt
a	yahoo	9	200	2.2
b	google	3	400	NaN
c	facebook	7	800	3.3

自动对齐之后，没有被复制的依然保持 NaN。

还可以更精准的修改数据吗？当然可以，完全仿照字典的操作：

```
>>> f3["price"]["c"] = 300
>>> f3
```

	name	price	marks	debt
a	yahoo	9	200	2.2
b	google	3	400	NaN
c	facebook	300	800	3.3

这些操作是不是都不陌生呀，这就是 Pandas 中的两种数据对象。

总目录 ([./index.html](#)) | 上节：为计算做准备 ([./310.html](#)) | 下节：Pandas 使用 (2) ([./312.html](#))

如果你认为有必要打赏我，请通过支付宝：**qiwsir@126.com**，不胜感激。

上一篇：为计算做准备 ([/project/start-learning-python/310.html](#))

下一篇：Pandas使用(2) ([/project/start-learning-python/312.html](#))



(<http://my.jikexueyuan.com/0xqXjaajP/>)
qq_m2tpw8nj (<http://my.jikexueyuan.com/0xqXjaajP/>)

#1

这个教材也太蠢了 开头就没说清楚 怎么启动 用什么软件 简直混蛋

2016年6月19日 1 回复



(<http://my.jikexueyuan.com/0igjjqqjP/>)
qq_x6b5p487 (<http://my.jikexueyuan.com/0igjjqqjP/>)

#2

#1@qq_m2tpw8nj

我觉得它写得很好啊，很浅显易懂，自己不能写教程教别人也不要这样用这样的话骂人把

2016年8月7日 0 回复



(<http://my.jikexueyuan.com/0yqqqPUqj/>)
jike_7935973 (<http://my.jikexueyuan.com/0yqqqPUqj/>)

#3

很清晰啊~1楼 是你自己理解能力不足吧.....

2016年12月19日 0 回复



(<http://my.jikexueyuan.com/0xgagqXkP/>)
qq_sur12ifl (<http://my.jikexueyuan.com/0xgagqXkP/>)

#4

#1@qq_m2tpw8nj

你是自己脑子秀逗了吧???? 先不论写的是否好坏, 起码我看懂了。。另外就算真是写的不好, 用得着你来教训? 别人欠你的? 不想看, 走就是了。这么牛逼, 你去写书啊

2017年1月10日 1 回复



(<http://my.jikexueyuan.com/pythonlearner/>)
weibo_z98ip6cj (<http://my.jikexueyuan.com/pythonlearner/>)

#5

写的很好 感谢总结。。。。

2017年3月21日 0 回复



(<http://my.jikexueyuan.com/0xqWPUkPk/>)
qq_3vpayki9 (<http://my.jikexueyuan.com/0xqWPUkPk/>)

#6

一楼, 脑子是个好东西, 但你没有, 以后也不会有

2017年3月29日 0 回复



(<http://my.jikexueyuan.com/0NgUqgPag/>)
wx_pk3ft4qm (<http://my.jikexueyuan.com/0NgUqgPag/>)

#7

#1@qq_m2tpw8nj

连这都看不懂, 还骂人, 这智商不要写代码了

2017年5月4日 0 回复



(<http://my.jikexueyuan.com/0yqqjkPPk/>)
qq_dc49av02 (<http://my.jikexueyuan.com/0yqqjkPPk/>)

#8

#1@qq_m2tpw8nj
你还是回家撸个JB吧 😊

2017年8月7日 0 回复



(<http://my.jikexueyuan.com/0yqqjkPPk/>)
qq_dc49av02 (<http://my.jikexueyuan.com/0yqqjkPPk/>)

#9

#1@qq_m2tpw8nj
疝你爹的个蛋 😊

2017年9月11日 0 回复



(<http://my.jikexueyuan.com/0HgXUjajk/>)
qq_wt10r5i4 (<http://my.jikexueyuan.com/0HgXUjajk/>)

#10

顶一个，反正我看懂了，不懂的也和一楼一起玩蛋蛋吧

2018年1月2日 0 回复



(<http://my.jikexueyuan.com/0HqXXVqkP/>)
杨星 (<http://my.jikexueyuan.com/0HqXXVqkP/>)

#11

#1@qq_m2tpw8nj
这都看不懂你还配撸代码？智商低不适合代码。

2018年2月12日 0 回复



(<http://my.jikexueyuan.com/0JjXagWUX/>)

jike_17866441 (<http://my.jikexueyuan.com/0JjXagWUX/>)

#12

非常感谢作者分享，循序渐进、浅显易懂。😊

2018年3月13日 0 回复



(<http://my.jikexueyuan.com/0JWakPPjP/>)

枫华远致 (<http://my.jikexueyuan.com/0JWakPPjP/>)

#13

浅显易懂，适合初学，感谢分享 😊

2018年3月14日 1 回复



(<http://my.jikexueyuan.com/0xWakXkjg/>)

qq_gi58jqia (<http://my.jikexueyuan.com/0xWakXkjg/>)

#14



涨姿势了....谢谢楼主

2018年3月14日 0 回复



(<http://my.jikexueyuan.com/0HgVXgUVj/>)

jike_d5o739jt (<http://my.jikexueyuan.com/0HgVXgUVj/>)

#15

#1@qq_m2tpw8nj

呵呵 python的库，你这脑子还学什么计算机，那个python ID不能写，那个系统不能写？

2018年6月20日 0 回复

[首页](#)[上一页](#)[1](#)[2](#)[下一页](#)[末页](#)[共 2 页](#)[当前 1 页](#)[确定](#)

只有登录了才能参与评论，快 登录 (<http://passport.jikexueyuan.com/sso/login>) ! 如果你还没有账号你可以注册 (http://passport.jikexueyuan.com/sso/reg_phone) 一个账号。

[关于我们 \(//help.jikexueyuan.com/\)](http://help.jikexueyuan.com/) [加入我们 \(//help.jikexueyuan.com/join.html\)](http://help.jikexueyuan.com/join.html)
[联系我们 \(//help.jikexueyuan.com/contact.html\)](http://help.jikexueyuan.com/contact.html)
[讲师合作 \(//j.jikexueyuan.com/evangelist/apply\)](http://j.jikexueyuan.com/evangelist/apply) [帮助中心 \(//help.jikexueyuan.com/\)](http://help.jikexueyuan.com/)
[黑板报 \(//blog.jikexueyuan.com/\)](http://blog.jikexueyuan.com/) [友情链接 \(//www.jikexueyuan.com/friendlink.html\)](http://www.jikexueyuan.com/friendlink.html) [意见反馈](#)