Operating on Data in Pandas

在Pandas中操作数据

> One of the essential pieces of NumPy is the ability to perform quick element-wise operations, both with basic arithmetic (addition, subtraction, multiplication, etc.) and with more sophisticated operations (trigonometric functions, exponential and logarithmic functions, etc.). Pandas inherits much of this functionality from NumPy, and the ufuncs that we introduced in Computation on NumPy Arrays: Universal Functions are key to this.

NumPy一个关键的能力就是它能快速的进行逐个元素运算，无论是基础算术运算（加法、减法、乘法等）还是更加复杂的运算（三角函数、幂指函数、对数函数等）。Pandas当然也继承了这种能力，我们在使用Numpy计算：通用函数中介绍的ufuncs就是提供这种能力的关键。

> Pandas includes a couple useful twists, however: for unary operations like negation and trigonometric functions, these ufuncs will *preserve index and column labels* in the output, and for binary operations such as addition and multiplication, Pandas will automatically *align indices* when passing the objects to the ufunc. This means that keeping the context of data and combining data from different sources–both potentially error-prone tasks with raw NumPy arrays–become essentially foolproof ones with Pandas. We will additionally see that there are well-defined operations between one-dimensional `Series` structures and two-dimensional `DataFrame` structures.

然而Pandas包括一些NumPy不具备的特性：对于一元运算如取负和三角函数，这些ufuncs会在结果中*保留原来的index和column标签*；对于二元运算如加法和乘法，Pandas会自动在结果中对参与运算的数据集进行*索引对齐*操作。这意味着在NumPy中对于不同数据集操作时以及需要保持数据的信息很容易发生错误的情况，在Pandas中就会很难会发生。我们还会看到针对一维的`Series`对象和二维的`DataFrame`对象都有定义良好的操作。

Ufuncs: Index Preservation

Ufuncs：保留索引

> Because Pandas is designed to work with NumPy, any NumPy ufunc will work on Pandas `Series` and `DataFrame` objects. Let's start by defining a simple `Series` and `DataFrame` on which to demonstrate this:

因为Pandas是设计和NumPy一起使用的，因此所有的NumPy通用函数都可以在Pandas的`Series`和`DataFrame`对象上使用。首先我们定义简单的`Series`和`DataFrame`对象来展示：

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
rng = np.random.RandomState(42)
ser = pd.Series(rng.randint(0, 10, 4))
ser
```

Out[2]:
```
0    6
1    3
2    7
3    4
dtype: int64
```

In [3]:
```
df = pd.DataFrame(rng.randint(0, 10, (3, 4)),
                  columns=['A', 'B', 'C', 'D'])
df
```

Out[3]:

	A	B	C	D
0	6	9	2	6
1	7	4	3	7
2	7	2	5	4

> If we apply a NumPy ufunc on either of these objects, the result will be another Pandas object *with the indices preserved:*

如果我们对上面的一个对象使用一元ufunc运算，结果会产生另一个Pandas对象，且*保留了索引*：

In [4]:
```
np.exp(ser)
```

Out[4]:
```
0     403.428793
1      20.085537
2    1096.633158
3      54.598150
dtype: float64
```

> Or, for a slightly more complex calculation:

下面是一个更加复杂的计算：

In [5]:
```
np.sin(df * np.pi / 4)
```

Out[5]:

	A	B	C	D
0	-1.000000	7.071068e-01	1.000000	-1.000000e+00
1	-0.707107	1.224647e-16	0.707107	-7.071068e-01
2	-0.707107	1.000000e+00	-0.707107	1.224647e-16

> Any of the ufuncs discussed in Computation on NumPy Arrays: Universal Functions can be used in a similar manner.

任何我们在使用Numpy计算：通用函数中讨论过的ufuncs都可以按照类似的方式进行运算。

UFuncs: Index Alignment

Ufuncs：索引对齐

> For binary operations on two `Series` or `DataFrame` objects, Pandas will align indices in the process of performing the operation. This is very convenient when working with incomplete data, as we'll see in some of the examples that follow.

对于两个`Series`或`DataFrame`进行二元运算操作，Pandas会在运算过程中会自动将两个数据集的索引进行对齐操作。这对于我们处理不完整的数据集的情况下非常方便，下面我们来看一些例子。

Index alignment in Series

Series对象中的索引对齐

> As an example, suppose we are combining two different data sources, and find only the top three US states by *area* and the top three US states by *population*:

假设我们从两个不同的数据源分别获得美国前三大面积和前三大人口的州，作为下面的例子：

In [6]:
```
area = pd.Series({'Alaska': 1723337, 'Texas': 695662,
                  'California': 423967}, name='area')
population = pd.Series({'California': 38332521, 'Texas': 26448193,
                        'New York': 19651127}, name='population')
```

> Let's see what happens when we divide these to compute the population density:

然后我们将人口和面积相除，计算各州的人口密度：

In [7]:
```
population / area
```

Out[7]:
```
Alaska              NaN
California    90.413926
New York            NaN
Texas         38.018740
dtype: float64
```

> The resulting array contains the *union* of indices of the two input arrays, which could be determined using standard Python set arithmetic on these indices:

结果数组中的索引包含了两个输入数组的并集，你可以通过标准的Python集合运算获得：

In [8]:
```
area.index | population.index
```

Out[8]:
```
Index(['Alaska', 'California', 'New York', 'Texas'], dtype='object')
```

> Any item for which one or the other does not have an entry is marked with `NaN`, or "Not a Number," which is how Pandas marks missing data (see further discussion of missing data in Handling Missing Data). This index matching is implemented this way for any of Python's built-in arithmetic expressions; any missing values are filled in with NaN by default:

两个任意输入数据集中对应的另一个数据集不存在的元素都会被设置为`NaN`（非数字的缩写），也就是Pandas标示缺失数据的方法（在处理空缺数据一节中会详细讨论）。索引的对齐方式会应用在任何Python內建的算术运算上，任何缺失的值都会被填充成NaN：

In [9]:
```
A = pd.Series([2, 4, 6], index=[0, 1, 2])
B = pd.Series([1, 3, 5], index=[1, 2, 3])
A + B
```

Out[9]:
```
0    NaN
1    5.0
2    9.0
3    NaN
dtype: float64
```

> If using NaN values is not the desired behavior, the fill value can be modified using appropriate object methods in place of the operators. For example, calling `A.add(B)` is equivalent to calling `A + B`, but allows optional explicit specification of the fill value for any elements in `A` or `B` that might be missing:

如果填充成NaN值不是你需要的结果，你可以使用相应的ufunc函数来计算，然后在函数中设置相应的填充值参数。例如，调用`A.add(B)`等同于调用`A + B`，但是可以提供额外的参数来设置用来缺失的替换值：

In [10]:
```
A.add(B, fill_value=0)
```

Out[10]:
```
0    2.0
1    5.0
2    9.0
3    5.0
dtype: float64
```

Index alignment in DataFrame

DataFrame中的索引对齐

> A similar type of alignment takes place for *both* columns and indices when performing operations on `DataFrame`s:

类似的对齐方式在对`DataFrame`操作当中会同时发生在列和行上：

In [11]:
```
A = pd.DataFrame(rng.randint(0, 20, (2, 2)),
                 columns=list('AB'))
A
```

Out[11]:

	A	B
0	1	11
1	5	1

In [13]:
```
B = pd.DataFrame(rng.randint(0, 10, (3, 3)),
                 columns=list('BAC'))
B
```

Out[13]:

	B	A	C
0	3	8	2
1	4	2	6
2	4	8	6

In [14]:
```
A + B
```

Out[14]:

	A	B	C
0	9.0	14.0	NaN
1	7.0	5.0	NaN
2	NaN	NaN	NaN

> Notice that indices are aligned correctly irrespective of their order in the two objects, and indices in the result are sorted. As was the case with `Series`, we can use the associated object's arithmetic method and pass any desired `fill_value` to be used in place of missing entries. Here we'll fill with the mean of all values in `A` (computed by first stacking the rows of `A`):

注意不管索引在输入数据集中的顺序并不会影响结果当中索引的对齐情况。与`Series`的情况一样，我们可以使用相应的ufunc函数来代替标准运算操作，然后代入你需要的`fill_value`参数来代替缺失值。这里我们会使用`A`中所有值的平均值来替代空值，我们首先堆叠（stack）`A`的所有行来计算平均值：

In [15]:
```
fill = A.stack().mean()
A.add(B, fill_value=fill)
```

Out[15]:

	A	B	C
0	9.0	14.0	6.5
1	7.0	5.0	10.5
2	12.5	8.5	10.5

> The following table lists Python operators and their equivalent Pandas object methods:

下面列出了Python的运算操作及其对应的Pandas方法：

Python运算符	Pandas方法
`+`	`add()`
`-`	`sub()`, `subtract()`
`*`	`mul()`, `multiply()`
`/`	`truediv()`, `div()`, `divide()`
`//`	`floordiv()`
`%`	`mod()`
`**`	`pow()`

Ufuncs: Operations Between DataFrame and Series

Ufuncs：DataFrame和Series之间的操作

> When performing operations between a `DataFrame` and a `Series`, the index and column alignment is similarly maintained. Operations between a `DataFrame` and a `Series` are similar to operations between a two-dimensional and one-dimensional NumPy array. Consider one common operation, where we find the difference of a two-dimensional array and one of its rows:

当在`DataFrame`和`Series`之间进行运算操作时，行和列的标签对齐机制依然有效。`DataFrame`和`Series`之间的操作类似于在一维数组和二维数组之间进行操作。例如一个很常见的操作，我们想要找出一个二维数组和它其中一行的差：

In [16]:
```
A = rng.randint(10, size=(3, 4))
A
```

Out[16]:
```
array([[1, 3, 8, 1],
       [9, 8, 9, 4],
       [1, 3, 6, 7]])
```

In [17]:
```
A - A[0]
```

Out[17]:
```
array([[ 0,  0,  0,  0],
       [ 8,  5,  1,  3],
       [ 0,  0, -2,  6]])
```

> According to NumPy's broadcasting rules (see Computation on Arrays: Broadcasting), subtraction between a two-dimensional array and one of its rows is applied row-wise.

依据NumPy的广播规则（参见在数组上计算：广播），二维数组的每一行都会减去它自身的第一行。

> In Pandas, the convention similarly operates row-wise by default:

Pandas中，默认也是采用这种广播机制：

In [18]:
```
df = pd.DataFrame(A, columns=list('QRST'))
df - df.iloc[0]
```

Out[18]:

	Q	R	S	T
0	0	0	0	0
1	8	5	1	3
2	0	0	-2	6

> If you would instead like to operate column-wise, you can use the object methods mentioned earlier, while specifying the `axis` keyword:

如果你希望能够按照列进行减法，你需要使用对应的ufunc函数，然后指定`axis`参数：

In [19]:
```
df.subtract(df['R'], axis=0)
```

Out[19]:

	Q	R	S	T
0	-2	0	5	-2
1	1	0	1	-4
2	-2	0	3	4

> Note that these `DataFrame`/`Series` operations, like the operations discussed above, will automatically align indices between the two elements:

上面介绍的这些`DataFrame`或者`Series`操作，都会自动对运算的数据集进行索引对齐：

In [20]:
```
halfrow = df.iloc[0, ::2] # 第一行的Q和S列
halfrow
```

Out[20]:
```
Q    1
S    8
Name: 0, dtype: int64
```

In [21]:
```
df - halfrow
```

Out[21]:

	Q	R	S	T
0	0.0	NaN	0.0	NaN
1	8.0	NaN	1.0	NaN
2	0.0	NaN	-2.0	NaN

> This preservation and alignment of indices and columns means that operations on data in Pandas will always maintain the data context, which prevents the types of silly errors that might come up when working with heterogeneous and/or misaligned data in raw NumPy arrays.

本节介绍的行与列索引保留和对齐机制说明Pandas在进行数据操作时会保持数据的上下文信息，因此可以避免同样情况下，使用NumPy数组操作不同形状和异构数据时会发生的错误。