第六章 连接

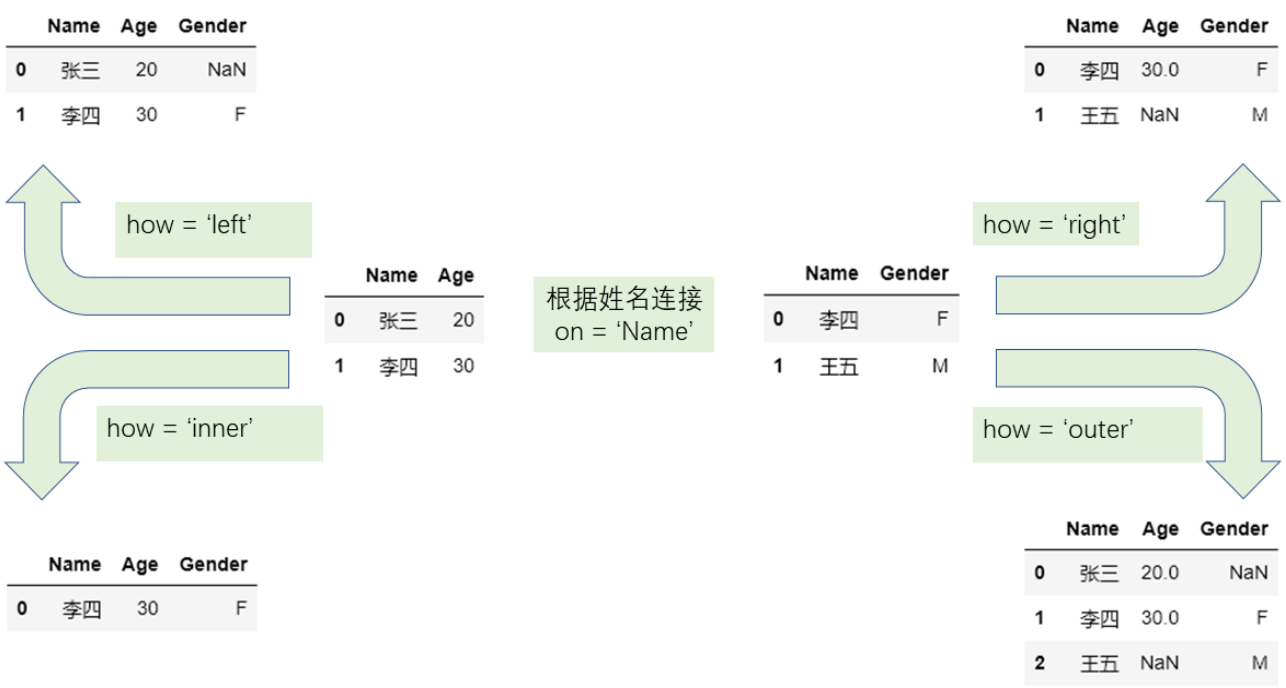
```
In [1]: import numpy as np

In [2]: import pandas as pd
```

一、关系型连接

1. 连接的基本概念

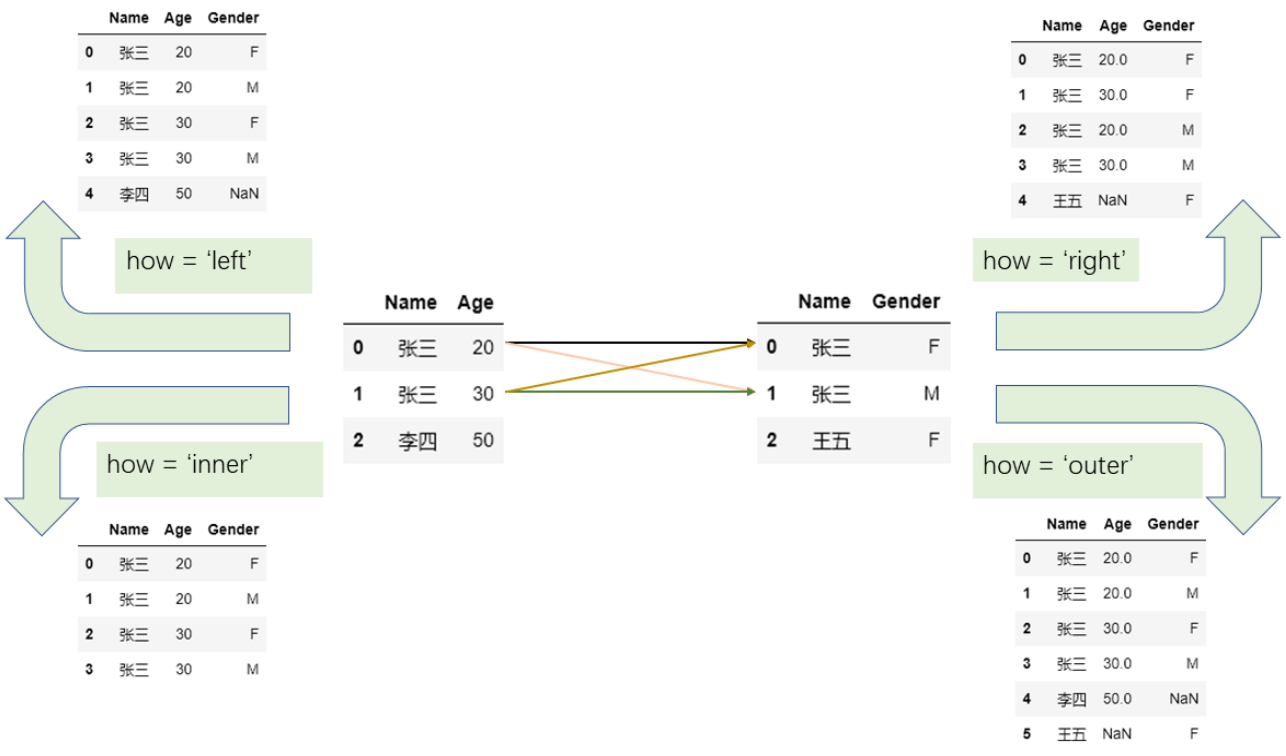
把两张相关的表按照某一个或某一组键连接起来是一种常见操作，例如学生期末考试各个科目的成绩表按照 姓名 和 班级 连接成总的成绩表，又例如对企业员工的各类信息表按照 员工ID号 进行连接汇总。由此可以看出，在关系型连接中， 键 是十分重要的，往往用 on 参数表示。

另一个重要的要素是连接的形式。在 pandas 中的关系型连接函数 merge 和 join 中提供了 how 参数来代表连接形式，分为左连接 left 、右连接 right 、内连接 inner 、外连接 outer ， 它们的区别可以用如下示意图表示：



从图中可以看到，所谓左连接即以左边的键为准，如果右边表中的键于左边存在，那么就添加到左边，否则则处理为缺失值，右连接类似处理。内连接只负责合并两边同时出现的键，而外连接则会在内连接的基础上包含只在左边出现以及只在右边出现的值，因此外连接又叫全连接。

上面这个简单的例子中，同一个表中的键没有出现重复的情况，那么如果出现重复的键应该如何处理？只需把握一个原则，即只要两边同时出现的值，就以笛卡尔积的方式加入，如果单边出现则根据连接形式进行处理。其中，关于笛卡尔积可用如下例子说明：设左表中键 张三 出现两次，右表中的 张三 也出现两次，那么逐个进行匹配，最后产生的表必然包含 2*2 个姓名为 张三 的行。下面是一个对应例子的示意图：



显然在不同的场合应该使用不同的连接形式。其中左连接和右连接是等价的，由于它们的结果中的键是被一侧的表确定的，因此常常用于有方向性地添加到目标表。内外连接两侧的表，经常是地位类似的，想取出键的交集或者并集，具体的操作还需要业务的需求来判断。

2. 值连接

在上面示意图中的例子中，两张表根据某一列的值来连接，事实上还可以通过几列值的组合进行连接，这种基于值的连接在 `pandas` 中可以由 `merge` 函数实现，例如第一张图的左连接：

```
In [3]: df1 = pd.DataFrame({'Name':['San Zhang','Si Li'],
   ...:                     'Age':[20,30]})
   ...:

In [4]: df2 = pd.DataFrame({'Name':['Si Li','Wu Wang'],
   ...:                     'Gender':['F','M']})
   ...:

In [5]: df1.merge(df2, on='Name', how='left')
Out[5]:
        Name  Age Gender
0  San Zhang   20    NaN
1      Si Li   30      F
```

如果两个表中想要连接的列不具备相同的列名，可以通过 left_on 和 right_on 指定：

```
In [6]: df1 = pd.DataFrame({'df1_name':['San Zhang','Si Li'],
   ...:                     'Age':[20,30]})
   ...:

In [7]: df2 = pd.DataFrame({'df2_name':['Si Li','Wu Wang'],
   ...:                     'Gender':['F','M']})
   ...:

In [8]: df1.merge(df2, left_on='df1_name', right_on='df2_name', how='left')
Out[8]:
    df1_name  Age df2_name Gender
0  San Zhang   20      NaN    NaN
1      Si Li   30    Si Li      F
```

如果两个表中的列出现了重复的列名，那么可以通过 `suffixes` 参数指定。例如合并考试成绩的时候，第一个表记录了语文成绩，第二个是数学成绩：

```
In [9]: df1 = pd.DataFrame({'Name':['San Zhang'],'Grade':[70]})

In [10]: df2 = pd.DataFrame({'Name':['San Zhang'],'Grade':[80]})

In [11]: df1.merge(df2, on='Name', how='left', suffixes=['_Chinese','_Math'])
Out[11]:
        Name  Grade_Chinese  Grade_Math
0  San Zhang             70          80
```

在某些时候出现重复元素是麻烦的，例如两位同学来自不同的班级，但是姓名相同，这种时候就要指定 `on` 参数为多个列使得正确连接：

```
In [12]: df1 = pd.DataFrame({'Name':['San Zhang', 'San Zhang'],
   ....:                     'Age':[20, 21],
   ....:                     'Class':['one', 'two']})
   ....:

In [13]: df2 = pd.DataFrame({'Name':['San Zhang', 'San Zhang'],
   ....:                     'Gender':['F', 'M'],
   ....:                     'Class':['two', 'one']})
   ....:

In [14]: df1
Out[14]:
        Name  Age Class
0  San Zhang   20   one
1  San Zhang   21   two

In [15]: df2
Out[15]:
        Name Gender Class
0  San Zhang      F   two
1  San Zhang      M   one

In [16]: df1.merge(df2, on='Name', how='left') # 错误的结果
Out[16]:
        Name  Age Class_x Gender Class_y
0  San Zhang   20     one      F     two
1  San Zhang   20     one      M     one
2  San Zhang   21     two      F     two
3  San Zhang   21     two      M     one

In [17]: df1.merge(df2, on=['Name', 'Class'], how='left') # 正确的结果
Out[17]:
        Name  Age Class Gender
0  San Zhang   20   one      M
1  San Zhang   21   two      F
```

从上面的例子来看，在进行基于唯一性的连接下，如果键不是唯一的，那么结果就会产生问题。举例中的行数很少，但如果实际数据中有几十万到上百万行的进行合并时，如果想要保证唯一性，除了用 `duplicated` 检查是否重复外， `merge` 中也提供了 `validate` 参数来检查连接的唯一性模式。这里共有三种模式，即一对一连接 `1:1` ，一对多连接 `1:m` ，多对一连接 `m:1` 连接，第一个是指左右表的键都是唯一的，后面两个分别指左表键唯一和右表键唯一。

> **练一练**
>
> 上面以多列为键的例子中，错误写法显然是一种多对多连接，而正确写法是一对一连接，请修改原表，使得以多列为键的正确写法能够通过 `validate='1:m'` 的检验，但不能通过 `validate='m:1'` 的检验。

另外，还有两个与 `merge` 函数相关的方法，它们是 `merge_ordered` 和 `merge_asof` ，它们一般会在时间序列问题中使用得比较频繁，因此将其安排在第十章中介绍。

3. 索引连接

所谓索引连接，就是把索引当作键，因此这和值连接本质上没有区别， `pandas` 中利用 `join` 函数来处理索引连接，它的参数选择要少于 `merge` ，除了必须的 `on` 和 `how` 之外，可以对重复的列指定左右后缀 `lsuffix` 和 `rsuffix` 。其中， `on` 参数指索引名，单层索引时省略参数表示按照当前索引连接。

```
In [18]: df1 = pd.DataFrame({'Age':[20,30]},
   ....:                     index=pd.Series(
   ....:                     ['San Zhang','Si Li'],name='Name'))
   ....:

In [19]: df2 = pd.DataFrame({'Gender':['F','M']},
   ....:                     index=pd.Series(
   ....:                     ['Si Li','Wu Wang'],name='Name'))
   ....:

In [20]: df1.join(df2, how='left')
Out[20]:
           Age Gender
Name
San Zhang   20    NaN
Si Li       30      F
```

仿照第2小节的例子，写出语文和数学分数合并的 `join` 版本：

```
In [21]: df1 = pd.DataFrame({'Grade':[70]},
   ....:                    index=pd.Series(['San Zhang'],
   ....:                    name='Name'))
   ....:

In [22]: df2 = pd.DataFrame({'Grade':[80]},
   ....:                    index=pd.Series(['San Zhang'],
   ....:                    name='Name'))
   ....:

In [23]: df1.join(df2, how='left', lsuffix='_Chinese', rsuffix='_Math')
Out[23]:
           Grade_Chinese  Grade_Math
Name
San Zhang             70          80
```

如果想要进行类似于 `merge` 中以多列为键的操作的时候， `join` 需要使用多级索引，例如在 `merge` 中的最后一个例子可以如下写出：

```
In [24]: df1 = pd.DataFrame({'Age':[20,21]},
   ....:                    index=pd.MultiIndex.from_arrays(
   ....:                    [['San Zhang', 'San Zhang'],['one', 'two']],
   ....:                    names=('Name','Class')))
   ....:

In [25]: df2 = pd.DataFrame({'Gender':['F', 'M']},
   ....:                    index=pd.MultiIndex.from_arrays(
   ....:                    [['San Zhang', 'San Zhang'],['two', 'one']],
   ....:                    names=('Name','Class')))
   ....:

In [26]: df1
Out[26]:
                  Age
Name      Class
San Zhang one      20
          two      21

In [27]: df2
Out[27]:
                Gender
Name      Class
San Zhang two        F
          one        M

In [28]: df1.join(df2)
Out[28]:
                  Age Gender
Name      Class
San Zhang one      20      M
          two      21      F
```

二、方向连接

1. concat

前面介绍了关系型连接，其中最重要的参数是 `on` 和 `how` ，但有时候用户并不关心以哪一列为键来合并，只是希望把两个表或者多个表按照纵向或者横向拼接，为这种需求， `pandas` 中提供了 `concat` 函数来实现。

在 `concat` 中，最常用的有三个参数，它们是 `axis, join, keys` ，分别表示拼接方向，连接形式，以及在新表中指示来自于哪一张旧表的名字。这里需要特别注意， `join` 和 `keys` 与之前提到的 `join` 函数和键的概念没有任何关系。

在默认状态下的 `axis=0` ，表示纵向拼接多个表，常常用于多个样本的拼接；而 `axis=1` 表示横向拼接多个表，常用于多个字段或特征的拼接。

例如，纵向合并各表中人的信息：

```
In [29]: df1 = pd.DataFrame({'Name':['San Zhang','Si Li'],
   ....:                     'Age':[20,30]})
   ....:

In [30]: df2 = pd.DataFrame({'Name':['Wu Wang'], 'Age':[40]})

In [31]: pd.concat([df1, df2])
Out[31]:
        Name  Age
0  San Zhang   20
1      Si Li   30
0    Wu Wang   40
```

横向合并各表中的字段：

```
In [32]: df2 = pd.DataFrame({'Grade':[80, 90]})

In [33]: df3 = pd.DataFrame({'Gender':['M', 'F']})

In [34]: pd.concat([df1, df2, df3], 1)
Out[34]:
        Name  Age  Grade Gender
0  San Zhang   20     80      M
1      Si Li   30     90      F
```

虽然说 `concat` 不是处理关系型合并的函数，但是它仍然是关于索引进行连接的。纵向拼接会根据列索引对其，默认状态下 `join=outer` ，表示保留所有的列，并将不存在的值设为缺失； `join=inner` ，表示保留两个表都出现过的列。横向拼接则根据行索引对齐， `join` 参数可以类似设置。

```
In [35]: df2 = pd.DataFrame({'Name':['Wu Wang'], 'Gender':['M']})

In [36]: pd.concat([df1, df2])
Out[36]:
        Name   Age Gender
0  San Zhang  20.0    NaN
1      Si Li  30.0    NaN
0    Wu Wang   NaN      M

In [37]: df2 = pd.DataFrame({'Grade':[80, 90]}, index=[1, 2])

In [38]: pd.concat([df1, df2])
Out[38]:
        Name   Age  Grade
0  San Zhang  20.0    NaN
1      Si Li  30.0    NaN
1        NaN   NaN   80.0
2        NaN   NaN   90.0

In [39]: pd.concat([df1, df2], join='inner')
Out[39]:
Empty DataFrame
Columns: []
Index: [0, 1, 1, 2]
```

因此，当确认要使用多表直接的方向合并时，尤其是横向的合并，可以先用 `reset_index` 方法恢复默认整数索引再进行合并，防止出现由索引的误对齐和重复索引的笛卡尔积带来的错误结果。

最后， `keys` 参数的使用场景在于多个表合并后，用户仍然想要知道新表中的数据来自于哪个原表，这时可以通过 `keys` 参数产生多级索引进行标记。例如，第一个表中都是一班的同学，而第二个表中都是二班的同学，可以使用如下方式合并：

```
In [40]: df1 = pd.DataFrame({'Name':['San Zhang','Si Li'],
   ....:                     'Age':[20,21]})
   ....:

In [41]: df2 = pd.DataFrame({'Name':['Wu Wang'],'Age':[21]})

In [42]: pd.concat([df1, df2], keys=['one', 'two'])
Out[42]:
            Name  Age
one 0  San Zhang   20
    1      Si Li   21
two 0    Wu Wang   21
```

2. 序列与表的合并

利用 concat 可以实现多个表之间的方向拼接，如果想要把一个序列追加到表的行末或者列末，则可以分别使用 append 和 assign 方法。

在 append 中，如果原表是默认整数序列的索引，那么可以使用 ignore_index=True 对新序列对应索引的自动标号，否则必须对 Series 指定 name 属性。

```
In [43]: s = pd.Series(['Wu Wang', 21], index = df1.columns)

In [44]: df1.append(s, ignore_index=True)
Out[44]:
         Name  Age
0  San Zhang   20
1      Si Li   21
2    Wu Wang   21
```

对于 assign 而言，虽然可以利用其添加新的列，但一般通过 df['new_col'] = ... 的形式就可以等价地添加新列。同时，使用 [] 修改的缺点是它会直接在原表上进行改动，而 assign 返回的是一个临时副本：

```
In [45]: s = pd.Series([80, 90])

In [46]: df1.assign(Grade=s)
Out[46]:
         Name  Age  Grade
0  San Zhang   20     80
1      Si Li   21     90

In [47]: df1['Grade'] = s

In [48]: df1
Out[48]:
         Name  Age  Grade
0  San Zhang   20     80
1      Si Li   21     90
```

三、类连接操作

除了上述介绍的若干连接函数之外， pandas 中还设计了一些函数能够对两个表进行某些操作，这里把它们统称为类连接操作。

1. 比较

compare 是在 1.1.0 后引入的新函数，它能够比较两个表或者序列的不同处并将其汇总展示：

```
In [49]: df1 = pd.DataFrame({'Name':['San Zhang', 'Si Li', 'Wu Wang'],
   ....:                     'Age':[20, 21 ,21],
   ....:                     'Class':['one', 'two', 'three']})
   ....:

In [50]: df2 = pd.DataFrame({'Name':['San Zhang', 'Li Si', 'Wu Wang'],
   ....:                     'Age':[20, 21 ,21],
   ....:                     'Class':['one', 'two', 'Three']})
   ....:

In [51]: df1.compare(df2)
Out[51]:
    Name          Class
    self  other   self  other
1  Si Li  Li Si    NaN    NaN
2    NaN    NaN  three  Three
```

结果中返回了不同值所在的行列，如果相同则会被填充为缺失值 NaN ，其中 other 和 self 分别指代传入的参数表和被调用的表自身。

如果想要完整显示表中所有元素的比较情况，可以设置 keep_shape=True ：

```
In [52]: df1.compare(df2, keep_shape=True)
Out[52]:
    Name          Age         Class
    self  other self other   self  other
0    NaN    NaN  NaN   NaN    NaN    NaN
1  Si Li  Li Si  NaN   NaN    NaN    NaN
2    NaN    NaN  NaN   NaN  three  Three
```

2. 组合

`combine` 函数能够让两张表按照一定的规则进行组合，在进行规则比较时会自动进行列索引的对齐。对于传入的函数而言，每一次操作中输入的参数是来自两个表的同名 `Series`，依次传入的列是两个表列名的并集，例如下面这个例子会依次传入 `A,B,C,D` 四组序列，每组为左右表的两个序列。同时，进行 `A` 列比较的时候， `s1` 指代的就是一个全空的序列，因为它在被调用的表中并不存在，并且来自第一个表的序列索引会被 `reindex` 成两个索引的并集。具体的过程可以通过在传入的函数中插入适当的 `print` 方法查看。

下面的例子表示选出对应索引位置较小的元素：

```
In [53]: def choose_min(s1, s2):
   ....:     s2 = s2.reindex_like(s1)
   ....:     res = s1.where(s1<s2, s2)
   ....:     res = res.mask(s1.isna()) # isna表示是否为缺失值，返回布尔序列
   ....:     return res
   ....:

In [54]: df1 = pd.DataFrame({'A':[1,2], 'B':[3,4], 'C':[5,6]})

In [55]: df2 = pd.DataFrame({'B':[5,6], 'C':[7,8], 'D':[9,10]}, index=[1,2])

In [56]: df1.combine(df2, choose_min)
Out[56]:
    A    B    C   D
0 NaN  NaN  NaN NaN
1 NaN  4.0  6.0 NaN
2 NaN  NaN  NaN NaN
```

练一练

请在上述代码的基础上修改，保留 `df2` 中4个未被 `df1` 替换的相应位置原始值。

此外，设置 `overtwrite` 参数为 `False` 可以保留 被调用表 中未出现在传入的参数表中的列，而不会设置未缺失值：

```
In [57]: df1.combine(df2, choose_min, overwrite=False)
Out[57]:
     A    B    C   D
0  1.0  NaN  NaN NaN
1  2.0  4.0  6.0 NaN
2  NaN  NaN  NaN NaN
```

练一练

除了 `combine` 之外， `pandas` 中还有一个 `combine_first` 方法，其功能是在对两张表组合时，若第二张表中的值在第一张表中对应索引位置的值不是缺失状态，那么就使用第一张表的值填充。下面给出一个例子，请用 `combine` 函数完成相同的功能。

```
In [58]: df1 = pd.DataFrame({'A':[1,2], 'B':[3,np.nan]})

In [59]: df2 = pd.DataFrame({'A':[5,6], 'B':[7,8]}, index=[1,2])

In [60]: df1.combine_first(df2)
Out[60]:
     A    B
0  1.0  3.0
1  2.0  7.0
2  6.0  8.0
```