

21

Joining and Merging Pandas DataFrames

Pandas 拼接合并

介绍 `concat()`、`join()`、`merge()` 三种方法



希望，是一个醒来的梦想。

Hope is a waking dream.

—— 亚里士多德 (Aristotle) | 古希腊哲学家 | 384 ~ 322 BC



- ▶ `pandas.concat()` 将多个数据帧在特定轴（行、列）方向进行拼接
- ▶ `pandas.DataFrame.drop()` 删除数据帧特定列
- ▶ `pandas.DataFrame.join()` 将两个数据集按照索引或指定列进行合并
- ▶ `pandas.DataFrame.merge()` 按照指定的列标签或索引进行数据库风格的合并



21.1 Pandas 数据帧拼接、合并

Pandas 是一种用于数据处理和分析的 Python 库，它提供了多种数据规整方法来整理和准备数据，使之能够更方便地进行分析和可视化。下面总结一些常用的数据规整方法。

将不同数据源的数据合并成一个数据集是数据规整的常见需求之一。Pandas 提供了多种方法进行数据合并和连接，比如，方法 `concat()` 将多个数据帧在特定轴方向进行拼接。方法 `join()` 将两个数据集按照索引或指定列进行合并。方法 `merge()` 按照指定的列标签或索引进行数据库风格的合并。

本章将介绍这三种方法。

21.2 拼接：pandas.concat()

`pandas.concat()` 是 `pandas` 库中的一个函数，用于将多个数据结构按照行或列的方向进行合并。它可以将数据连接在一起，形成一个新的 `DataFrame`。

这个函数的主要参数为 `pandas.concat(objs, axis=0, join='outer', ignore_index=False)`。

参数 `objs`: 这是一个需要连接的对象列表，比如 `[df1, df2, df3]`。

参数 `axis` 指定连接的轴向，可以是 0 或 1，默认为 0；0 表示按行连接（如图 2 所示），1 表示按列连接（如图 3 所示）。


参数 `join` 指定拼接的方式，可以是 'inner'、'outer'，默认是 'outer'。'inner' 表示内连接，只保留两个数据集中共有的列/行。'outer' 表示外连接，保留所有列/行，缺失值用 `NaN` 填充。

图 1 给出的代码比较 'outer' 和 'inner' 和两种拼接方式。

```
import pandas as pd
# 创建两个数据帧
df1 = pd.DataFrame({'X1': [1, 2, 3],
                    'X2': ['X', 'Y', 'Z']},
                  index=[0, 1, 2])

df2 = pd.DataFrame({'X3': ['A', 'B', 'C'],
                    'X4': [4, 5, 6]},
                  index=[1, 2, 3])

# 'outer' 方法拼接
a df_outer = pd.concat([df1, df2], join='outer', axis=1)
# 'inner' 方法拼接
b df_inner = pd.concat([df1, df2], join='inner', axis=1)
```



**用concat()
拼接**

图 1. 用 concat() 拼接，比较 'outer' 和 'inner'

a 的结果如图 4 所示，图中 × 代表 NaN 缺失值。 b 的结果如图 5 所示。
 参数 ignore_index 为布尔值，默认为 False；如果设置为 True，将会重新生成索引，忽略原来的索引。

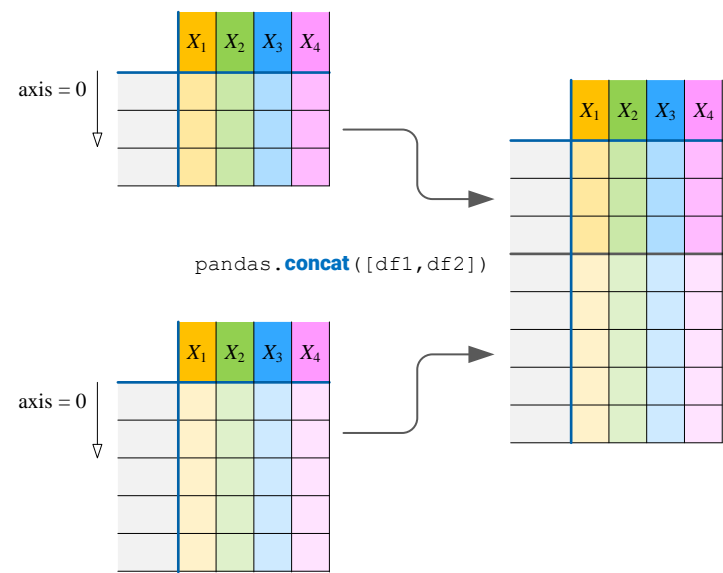


图 2. 利用 pandas.concat() 完成轴方向拼接，axis = 0（默认）

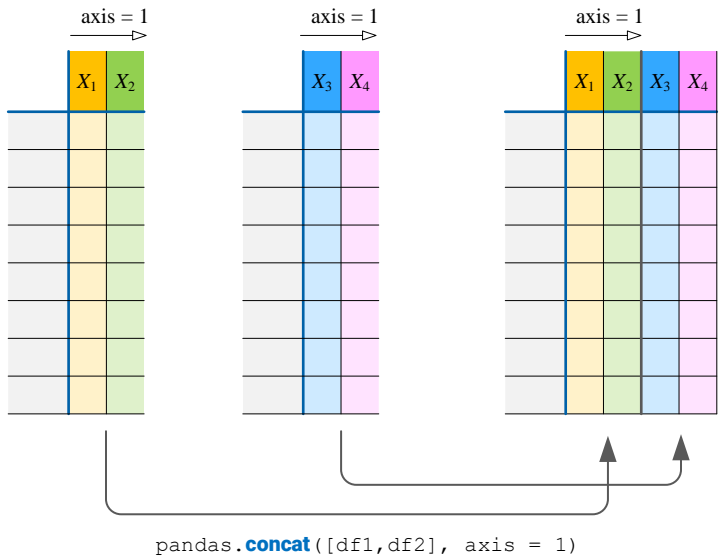


图 3. 利用 pandas.concat() 完成轴方向拼接，axis = 1

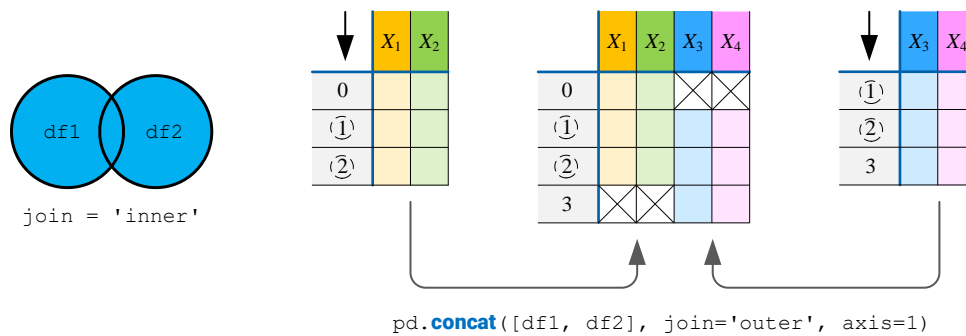


图 4. 利用 pandas.concat() 完成合并, join = 'outer'

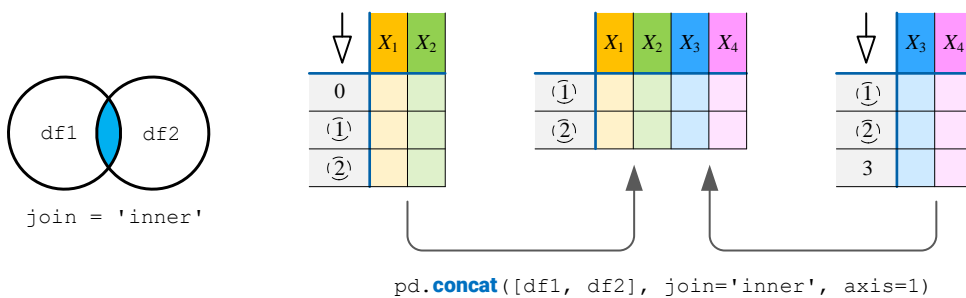


图 5. 利用 pandas.concat() 完成合并, join = 'inner'

21.3 合并: pandas.join()

在 Pandas 中, join 是 DataFrame 对象的一个方法, 用于按照索引 (默认) 或指定列合并两个 DataFrame。

这个函数的主要参数为 DataFrame.join(other, on=None, how='left', lsuffix="", rsuffix=")。

参数 other 是要连接的另一个 DataFrame。

参数 on 是指定连接的列名或列标签级别 (多级列标签的情况) 的名称。如果不指定, 将会以两个 DataFrame 的索引为连接依据。

参数 how 指定连接方式, 可以是 'left' (左连接)、'right' (右连接)、'outer' (外连接)、'inner' (内连接) 或 'cross' (交叉连接), 默认是 'left'。图 6 代表比较 'left'、'right'、'outer'、'inner' 这四种方法。

如图 7 所示, 'left' 使用左侧 DataFrame 的索引或指定列进行合并。

如图 8 所示, 'right' 使用右侧 DataFrame 的索引或指定列进行合并。

如图 9 所示, 'outer' 使用两个 DataFrame 的并集索引或指定列进行合并, 缺失值用 NaN 填充。

如图 10 所示, 'inner' 使用两个 DataFrame 的交集索引或指定列进行合并。

如图 11 代码所示，'cross' 连接是一种笛卡尔积的连接方式，它会将两个 DataFrame 的所有行进行组合，从而得到两个 DataFrame 之间的所有可能组合。图 12 给出这种合并方法的图解。


'cross' 这种连接方式在 SQL 中称为 "CROSS JOIN"。'cross' 连接方式适用于较小的 DataFrame，因为连接后的结果行数会呈指数增长。如果 DataFrame 较大，这种连接方式可能会导致非常庞大的结果，从而占用大量的内存和计算资源。因此，在使用 'cross' 连接时，应该谨慎操作，确保不会导致资源耗尽。

当连接的两个 DataFrame 中存在同名的列时，可以通过 lsuffix 和 rsuffix 这两个参数为左边和右边的列名添加后缀 (suffix)，避免列名冲突。

```
import pandas as pd
# 创建两个数据帧
df1 = pd.DataFrame({'X1': [1, 2, 3],
                    'X2': ['X', 'Y', 'Z']},
                    index=[0, 1, 2])

df2 = pd.DataFrame({'X3': ['A', 'B', 'C'],
                    'X4': [4, 5, 6]},
                    index=[1, 2, 3])

# 'left' 方法合并
a df_left = df1.join(df2, how='left')
# 'right' 方法合并
b df_right = df1.join(df2, how='right')
# 'outer' 方法合并
c df_outer = df1.join(df2, how='outer')
# 'inner' 方法合并
d df_inner = df1.join(df2, how='inner')
```



用join()合并

图 6. 用 join() 合并，比较 'left'、'right'、'outer'、'inner'

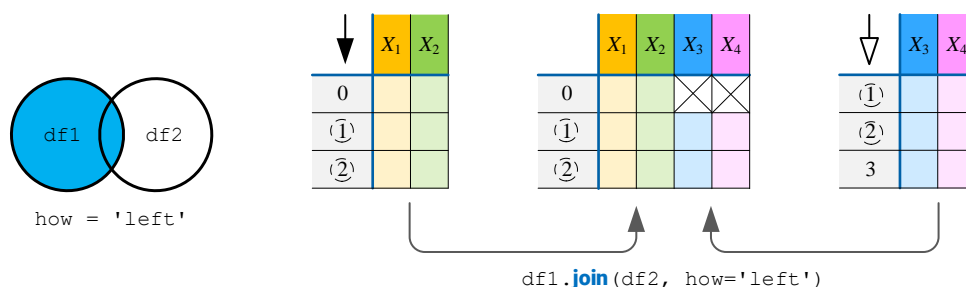


图 7. 利用 pandas.join() 完成合并，join = 'left'

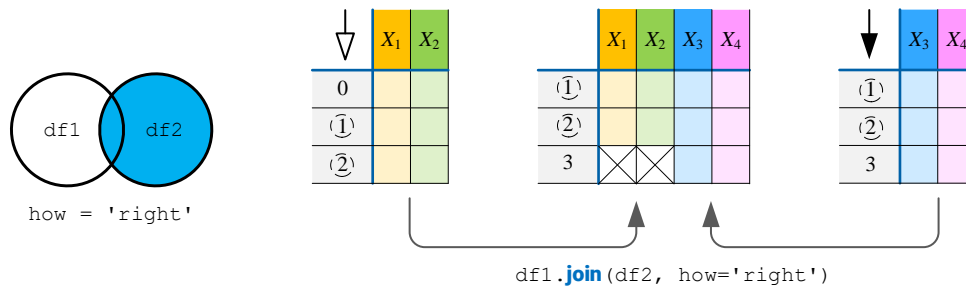


图 8. 利用 pandas.join() 完成合并, join = 'right'

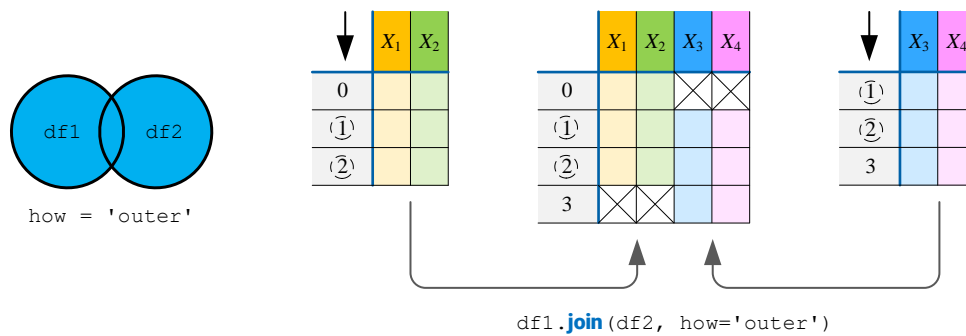


图 9. 利用 pandas.join() 完成合并, join = 'outer'

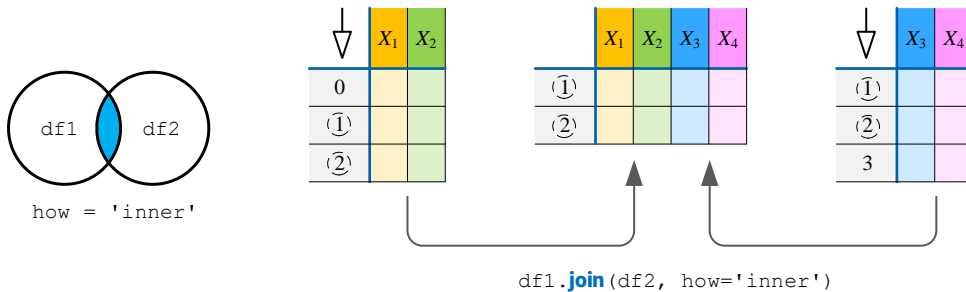


图 10. 利用 pandas.join() 完成合并, join = 'inner'

```
import pandas as pd
# 创建两个数据帧
df1 = pd.DataFrame({'A': ['X', 'Y', 'Z']})
df2 = pd.DataFrame({'B': [1, 2]})

# 使用 'cross' 连接
df_cross = df1.join(df2, how='cross')
```

用join()合并，笛卡尔积

图 11. 用 join() 合并, how = 'cross'

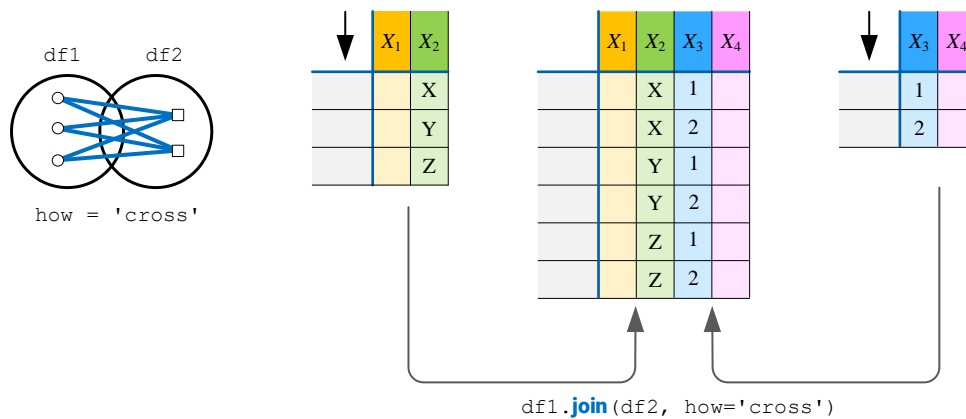


图 12. 利用 pandas.join() 完成合并, join = 'cross'

21.4 合并: pandas.merge()

实践中, 相较本章前文介绍的两种方法, merge() 更灵活, 可以处理更多种合并情况。merge() 可以通过指定列标签合并 (参数 left_on 和 right_on, 或 on), 可以指定索引 (left_index 和 right_index) 合并。merge() 还支持 'left'、'right'、'outer'、'inner' 或 'cross' 五种合并方法。

基于单个列合并

图 13 所示为 merge() 通过参数 on 指定同名列标签, 完成 df_left 和 df_right 两个数据帧合并, 合并方法为 how = 'left'。如图 14 所示, 当两个数据帧有同名列标签时, 合并后同名标签会加后缀以便区分, 默认标签为 (“_x”, “_y”)。

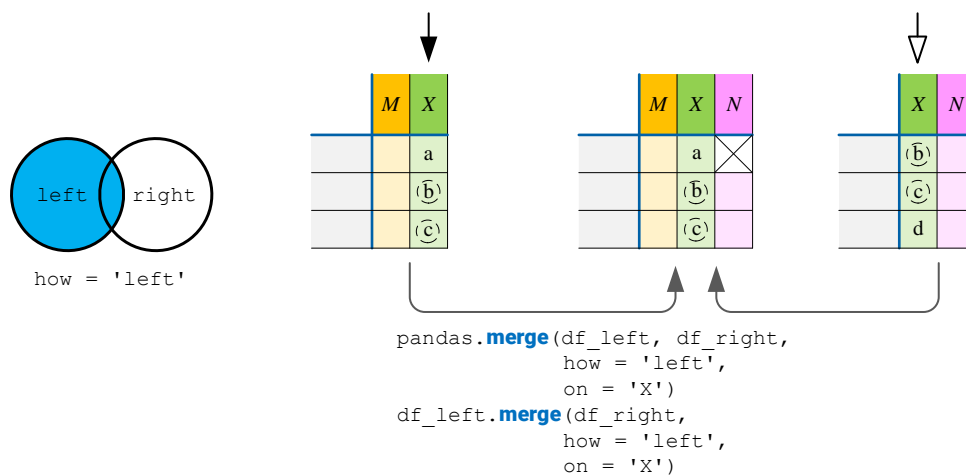


图 13. 利用 pandas.merge() 完成合并, how = 'left'

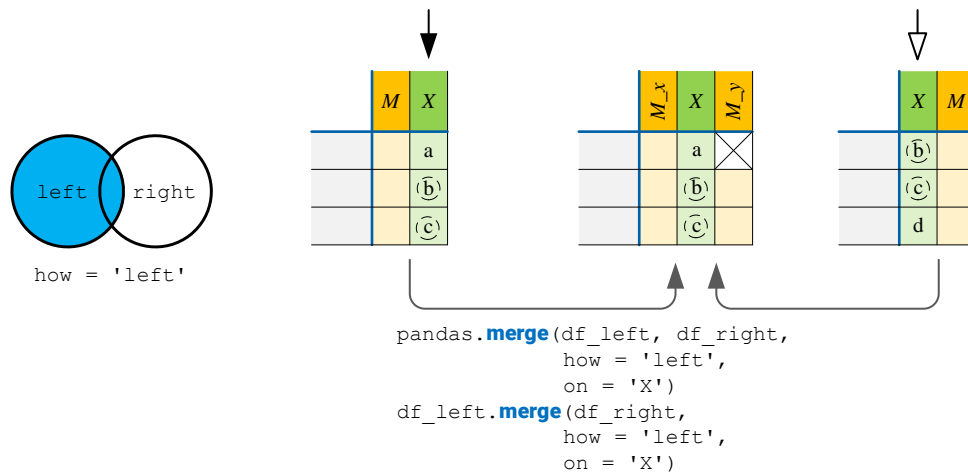


图 14. 利用 pandas.merge() 完成合并，how = 'left'，有列标签重名的情况

基于左右列合并

图 15 ~ 图 18 所示为 merge() 通过指定左右数据帧的列标签 (left_on 和 right_on) 完成合并。此外，merge() 还可以指定多个列标签进行合并操作。

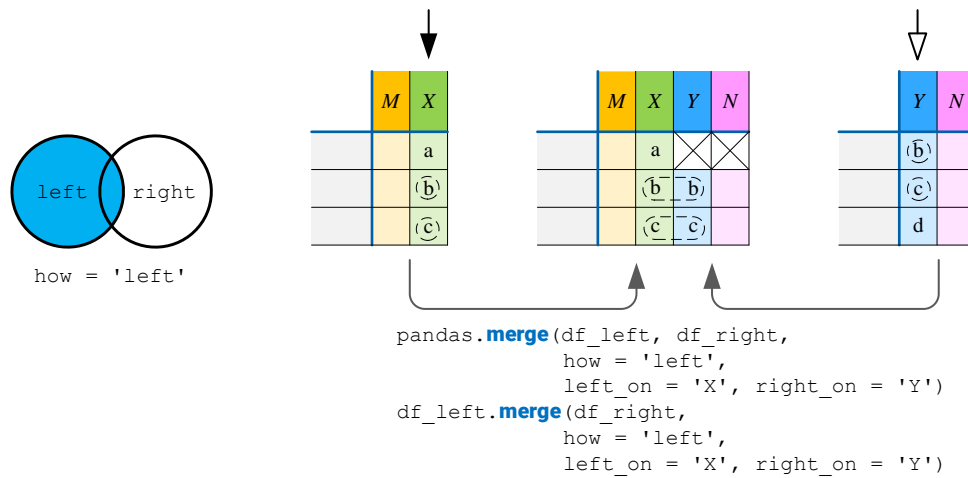


图 15. 利用 pandas.merge() 完成合并，how = 'left'

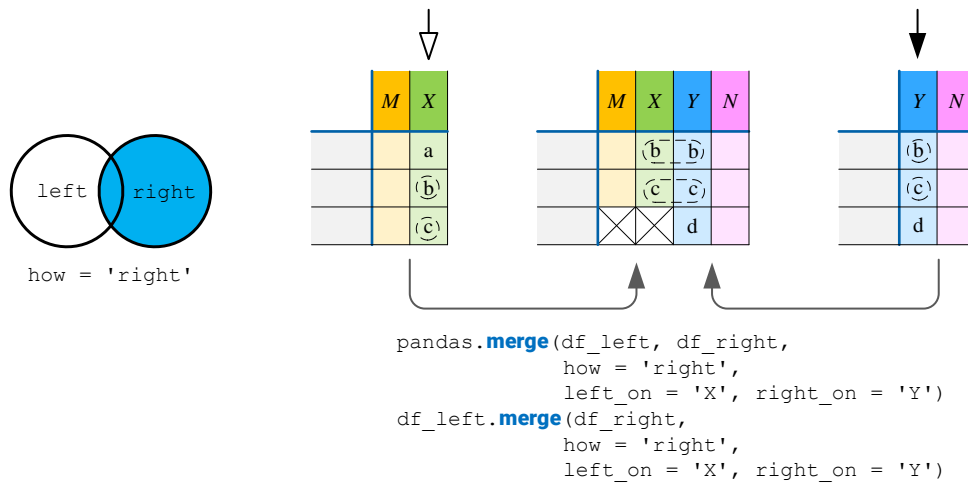


图 16. 利用 `pandas.merge()` 完成合并, `how = 'right'`

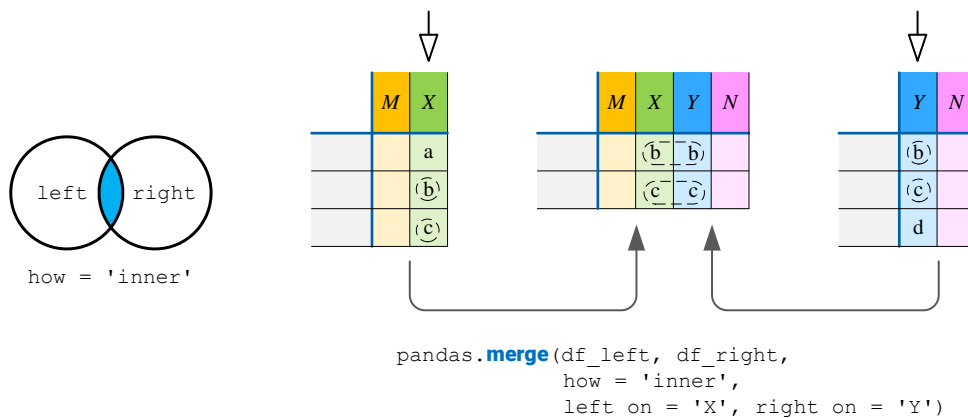


图 17. 利用 `pandas.merge()` 完成合并, `how = 'inner'`

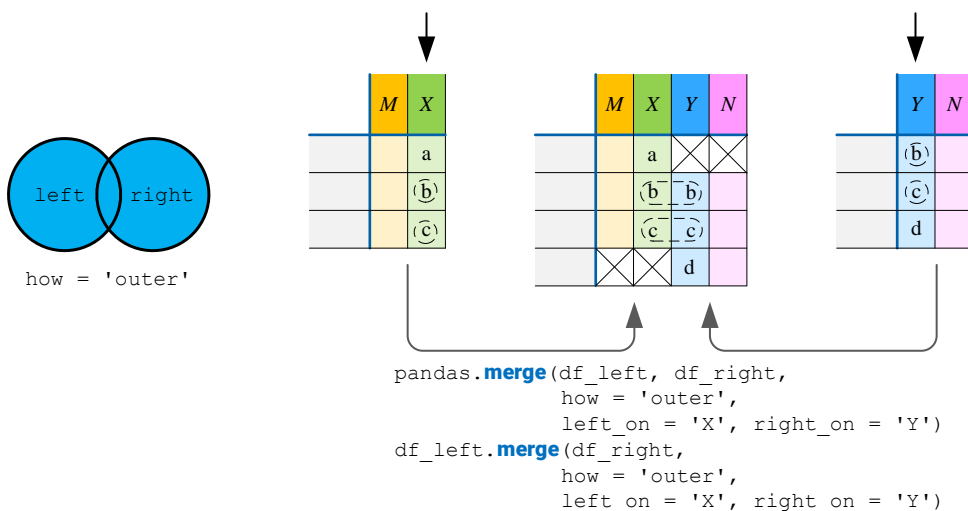


图 18. 利用 `pandas.merge()` 完成合并, `how = 'outer'`

独有

图 19 总结常用几种合并几何运算，`merge()` 可以直接完成前 5 种，目前 `merge()` 暂不直接支持剩下 3 种。这 3 种合并集合运算为：

左侧独有 (left exclusive)：只保留左侧 DataFrame 中存在，而右侧 DataFrame 中不存在的行。

右侧独有 (right exclusive)：只保留右侧 DataFrame 中存在，而左侧 DataFrame 中不存在的行。

全外独有 (full outer exclusive)：保留左侧 DataFrame 中不存在于右侧 DataFrame，同时右侧 DataFrame 中不存在于左侧 DataFrame 的行。

但是，我们可以利用 `merge()` 完成图 19，具体代码如图 20 所示。

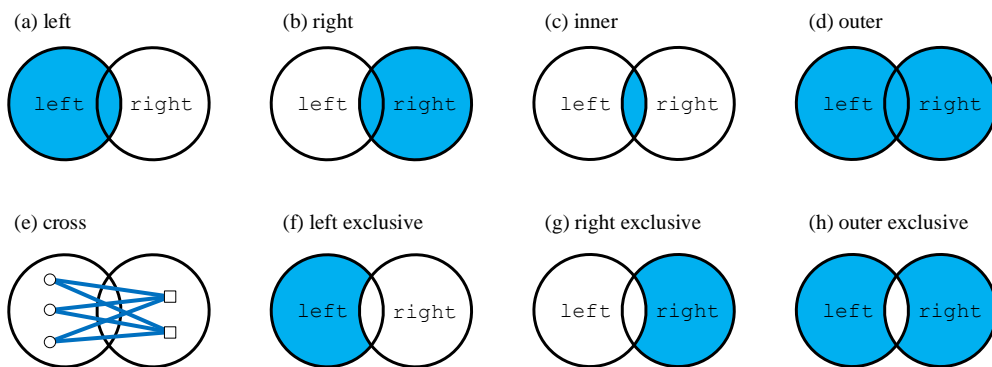


图 19. 总结常用合并集合运算

图 20 中的 **a** 首先利用 `merge()` 完成左连接合并。在 `pandas` 的 `merge()` 方法中，`indicator` 参数用于指定是否添加一个特殊的列，该列记录了每行的合并方式。这个特殊的列名可以通过 `indicator` 参数进行自定义，默认为 `"_merge"`。`"_merge"` 列可以取三个值：

`"left_only"`：表示该行只在左边的 DataFrame 中存在，即左连接中独有的行。

`"right_only"`：表示该行只在右边的 DataFrame 中存在，即右连接中独有的行。

`"both"`：表示该行在两个 DataFrame 中都存在，即连接方式中共有的行。

在 **b** 中，通过设定筛选条件，`left_exl['_merge'] == 'left_only'`，我们可以保留合并后的“左侧独有”行。结果如图 21 所示。

同理，**c** 完成右连接合并，**d** 通过设定筛选条件保留数据帧中“右侧独有”行，结果如图 22 所示。类似地，**e** 完成外连接合并，**f** 通过设定筛选条件保留“全外独有”行，结果如图 23 所示。

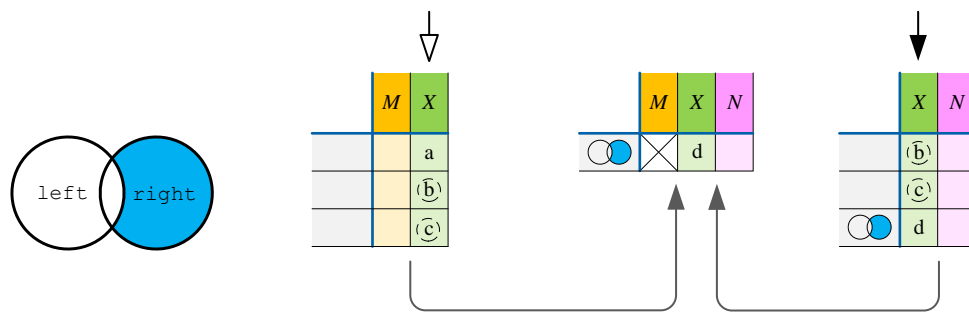


图 22. 利用 pandas.merge() 完成合并，右侧独有

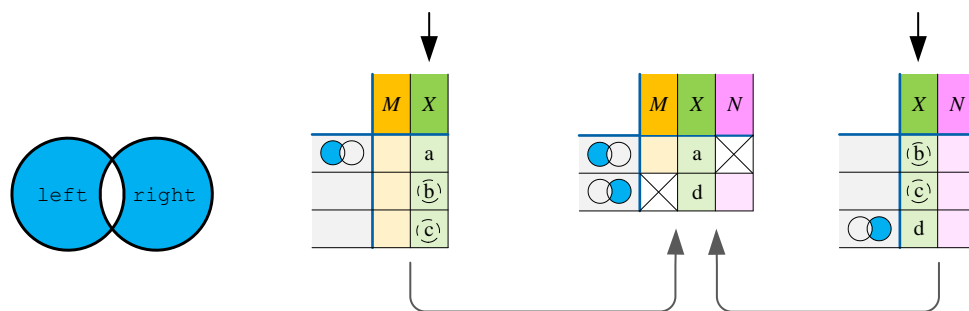


图 23. 利用 pandas.merge() 完成合并，全外独有



更多有关合并、比较的方法，请参考：

https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html