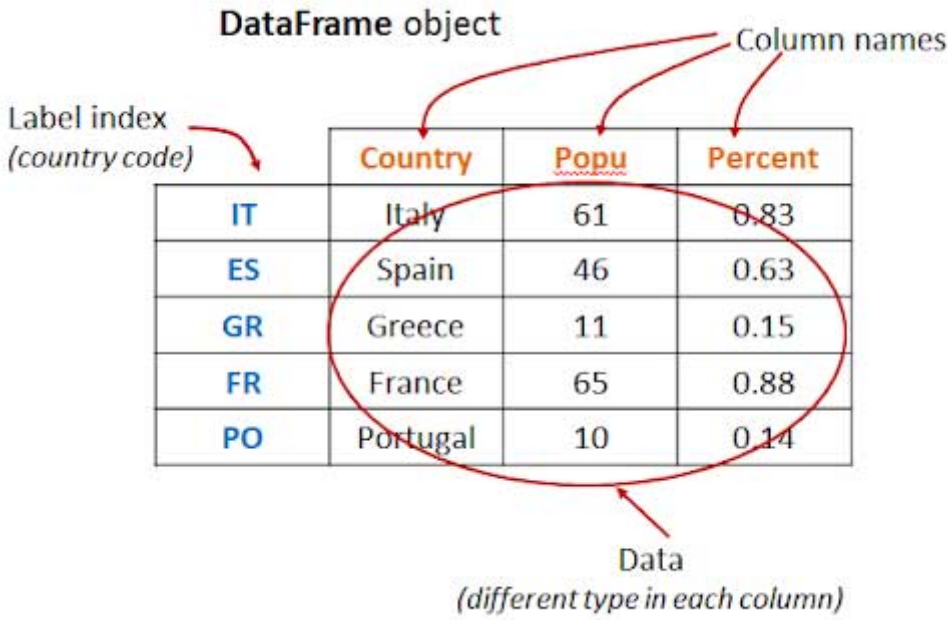
03Pandas文件索引

`Pandas` 数据的索引就像一本书的目录，让我们很快地找到想要看的章节，作为大量数据，创建合理的具有业务意义的索引对我们分析数据至关重要。

一、认识索引

下图是一个简单的 `DataFrame` 中索引的示例：



其中：

- 行索引是数据的索引，列索引指向的是一个 `Series`
- `DataFrame` 的索引也是系列形成的 `Series` 的索引
- 建立索引让数据更加直观明确，如每行数据是针对一个国家的
- 建立索引方便数据处理
- 索引允许重复，但业务上一般不会让它重复

有时一个行和列层级较多的数据会出现多层索引的情况。

二、建立索引

之前我们学习了加载数据生成 `DataFrame` 时可以指定索引

```
data = 'res/team.xlsx'
df = pd.read_excel(data, index_col='name') # 设置索引为 name
df
'''
      team  Q1  Q2  Q3  Q4
name
Liver    E  89  21  24  64
Arry     C  36  37  37  57
Ack      A  57  60  18  84
Eorge    C  93  96  71  78
Oah      D  65  49  61  86
'''
```

如果加载时没有指定索引，我们可以使用 `df.set_index()` 指定：

```
df.set_index('month') # 设置月份为索引
df.set_index(['month', 'year']) # 设置月份和年为多层索引
'''
            sale
month year
1     2012    55
4     2014    40
      2013    84
10    2014    31
'''

s = pd.Series([1, 2, 3, 4])
df.set_index(s) # 指定一个索引
df.set_index([s, 'year']) # 指定的索引和现有字段同时指定
df.set_index([s, s**2]) # 计算索引

# 其他的参数
df.set_index('month', drop=False) # 保留原列
df.set_index('month', append=True) # 保留原来的索引
df.set_index('month', inplace=True) # 建立索引并重写覆盖 df
```

三、索引类型

为了适应各种业务数据的处理，索引又针对各种类型数据定义了不同的索引类型：

1.数字索引 Numeric Index

共有以下几种：

- `RangeIndex`: 单调整数范围的不可变索引。
- `Int64Index`: `int64`类型，有序可切片集合的不可变`ndarray`。
- `UInt64Index`: 无符号整数标签的
- `Float64Index`: `Float64` 类型

```
pd.RangeIndex(1,100,2)
# RangeIndex(start=1, stop=100, step=2)
pd.Int64Index([1,2,3,-4], name='num')
# Int64Index([1, 2, 3, -4], dtype='int64', name='num')
pd.UInt64Index([1,2,3,4])
# UInt64Index([1, 2, 3, 4], dtype='uint64')
pd.Float64Index([1.2,2.3,3,4])
# Float64Index([1.2, 2.3, 3.0, 4.0], dtype='float64')
```

2.类别索引 CategoricalIndex

类别只能包含有限数量的（通常是固定的）可能值（类别）。 可以理解成枚举，比如性别只有男女，但在数据中每行都有，如果按文本处理会效率不高。类别的底层是`pandas.Categorical`。

```
pd.CategoricalIndex(['a', 'b', 'a', 'b'])
# CategoricalIndex(['a', 'b', 'a', 'b'], categories=['a', 'b'],
ordered=False, dtype='category')
```

类别后边后有专门的讲解，只有在体量非常大的数据面前才能显示其优势。

3.间隔索引 IntervalIndex

```
pd.interval_range(start=0, end=5)
'''
IntervalIndex([(0, 1], (1, 2], (2, 3], (3, 4], (4, 5]],
              closed='right',
              dtype='interval[int64]')
'''
```

4.时间索引 DatetimeIndex

```
# 从一个日期连续到另一个日期
pd.date_range(start='1/1/2018', end='1/08/2018')
# 指定开始时间和周期
pd.date_range(start='1/1/2018', periods=8)
# 以月为周期
pd.period_range(start='2017-01-01', end='2018-01-01', freq='M')
# 周期嵌套
pd.period_range(start=pd.Period('2017Q1', freq='Q'),
                end=pd.Period('2017Q2', freq='Q'), freq='M')
```

5.时间差 TimedeltaIndex

```
pd.TimedeltaIndex(data =['06:05:01.000030', '+23:59:59.999999',
                         '22 day 2 min 3us 10ns',
'+23:29:59.999999',
                         '+12:19:59.999999'])
# 使用 datetime
pd.TimedeltaIndex(['1 days', '1 days, 00:00:05',
                   np.timedelta64(2, 'D'),
                   datetime.timedelta(days=2, seconds=2)])
```

6.周期索引 PeriodIndex

```
t = pd.period_range('2020-5-1 10:00:05', periods=8, freq='S')
pd.PeriodIndex(t,freq='S')
```

四、索引对象

行和列的索引在 `Pandas` 里其实是一个 `Index` 对象，以下是创建一个 `index` 对象的方法：

1.创建对象

```
pd.Index([1, 2, 3])
# Int64Index([1, 2, 3], dtype='int64')
pd.Index(list('abc'))
# Index(['a', 'b', 'c'], dtype='object')
# 可以定义一相 name
pd.Index(['e', 'd', 'a', 'b'], name='something')
```

2.查看

```
df.index
# RangeIndex(start=0, stop=4, step=1)
df.columns
# Index(['month', 'year', 'sale'], dtype='object')
```

3.属性

以下方法也适用于 `df.columns`, 因为都是 `index` 对象：

```
# 属性
df.index.name # 名称
df.index.array # array 数组
df.index.dtype # 数据类型
df.index.shape # 形状
df.index.size # 元素数量
df.index.values # array 数组
# 其他，不常用
df.index.empty # 是否为空
df.index.is_unique # 是否不重复
df.index.names # 名称列表
```

```python
df.index.is_all_dates # 是否全是日期时间
df.index.has_duplicates # 是否有重复值
df.index.values # 索引的值 array
```

4.操作

以下方法也适用于 `df.columns`, 因为都是 `index` 对象，有些也支持 `Series`：

```python
# 方法
df.index.astype('int64') # 转换类型
df.index.isin() # 是否存在，见下方示例
df.index.rename('number') # 修改索引名称
df.index.nunique() # 不重复值的数量
df.index.sort_values(ascending=False,) # 排序,倒序
df.index.map(lambda x:x+'_') # map 函数处理
df.index.str.replace('_', '') # str 替换
df.index.str.split('_') # 分隔
df.index.to_list() # 转为列表
df.index.to_frame(index=False, name='a') # 转成 DataFrame
df.index.to_series() # 转 series
df.index.to_numpy() # 转为 numpy
df.index.unique() # 去重
df.index.value_counts() # 去重及数量
df.index.where(df.index=='a') # 筛选
df.index.rename('grade', inplace=False) # 重命名索引名称
df.index.rename(['species', 'year']) # 多层，重命名索引名称
df.index.max() # 最大值
df.index.argmax() # 最大索引值
df.index.any()
df.index.all()
df.index.T # 转置，多层索引里很有用

# 其他，不常用
df.index.append(pd.Index([4,5])) # 追加
df.index.repeat(2) # 重复几次
df.index.inferred_type # 推测数据类型
df.index.hasnans # 有没有空值
df.index.is_monotonic_decreasing # 是否单调递减
df.index.is_monotonic # 是否单调递增
df.index.is_monotonic_increasing # 是否单调递增
df.index.nbytes # 基础数据中的字节数
df.index.ndim # 维度数，维数
df.index.nlevels # 索引层级数，通常为 1
df.index.min() # 最小值
df.index.argmin() # 最小索引值
df.index.argsort() # 顺序值组成的数组
df.index.asof(2) # 返回最近的索引
# numpy dtype or pandas type
df.index.astype('int64', copy=True) # 深拷贝
# 拷贝
df.index.copy(name='new', deep=True, dtype='int64')
```

```python
df.index.delete(1) # 删除指定位置
# 对比不同
df.index.difference(pd.Index([1,2,4]), sort=False)
df.index.drop('a', errors='ignore') # 删除
df.index.drop_duplicates(keep='first') # 去重值
df.index.droplevel(0) # 删除层级
df.index.dropna(how='all') # 删除空值
df.index.duplicated(keep='first') # 重复值在结果数组中为True
df.index.equals(df.index) # 与另外一个索引对象是否相同
df.index.factorize() # 分解成（array:0-n, Index）
df.index.fillna(0, {0:'nan'}) # 填充空值
# 字符列表，把 name 值加在第一位，每个值加10
df.index.format(name=True, formatter=lambda x:x+10)

# 返回一个 array，指定值的索引位数组，不在的为 -1
df.index.get_indexer([2,9])
# 获取 指定层级 Index 对象
df.index.get_level_values(0)
# 指定索引的位置，见示例
df.index.get_loc('b')
df.index.insert(2, 'f') # 在索引位 2 插入 f
df.index.intersection(df.index) # 交集
df.index.is_(df.index) # 类似 is 检查
df.index.is_categorical() # 是否分类数据
df.index.is_type_compatible(df.index) # 类型是否兼容
df.index.is_type_compatible(1) # 类型是否兼容

df.index.isna() # array 是否为空
df.index.isnull() # array 是否缺失值
df.index.join(df.index, how='left') # 连接
df.index.notna() # 是否不存在的值
df.index.notnull() # 是否不存在的值
df.index.ravel() # 展平值的ndarray
df.index.reindex(['a','b']) # 新索引 (Index,array:0-n)
df.index.searchsorted('f') # 如果插入这个值排序后在哪个索引位
df.index.searchsorted([0, 4]) # array([0, 3]) 多个
df.index.set_names('quarter') # 设置索引名称
df.index.set_names('species', level=0)
df.index.set_names(['kind', 'year'], inplace=True)
df.index.shift(10, freq='D') # 日期索引向前移动 10 天
idx1.symmetric_difference(idx2) # 两个索引不同的内容
idx1.union(idx2) # 拼接

df.add_prefix('t_') # 表头加前缀
df.add_suffix('_d') # 表头加后缀
df.first_valid_index() # 第一个有值的索引
df.last_valid_index() # 最后一个有值的索引
```

五、索引重命名

对行和列的索引名进行修改。

```
# 一一对应修改列索引
df.rename(columns={"A": "a", "B": "c"})
df.rename(str.lower, axis='columns')
# 修改行索引
df.rename(index={0: "x", 1: "y", 2: "z"})
df.rename({1: 2, 2: 4}, axis='index')
# 修改数据类型
df.rename(index=str)
# 重新修改索引
replacements = {l1:l2 for l1, l2 in zip(list1, list2)}
df.rename(replacements)
# 列名加前缀
df.rename(lambda x:'t_' + x, axis=1)
# 利用 iter() 函数的 next 特性修改
df.rename(lambda x, y=iter('abcdef'): next(y), axis=1)
# 修改列名，用解包形式生成新旧字段字典
df.rename(columns=dict(zip(df, list('abcd'))))
```

六、索引示例

```
# idx.isin() 是否存在
idx = pd.Index([1,2,3])
df.index.isin(idx)
# array([False, False, False, False])
df.index.isin(['a','b'])
# array([ True,  True, False, False])
midx = pd.MultiIndex.from_arrays([[1,2,3],
                                  ['red', 'blue', 'green']],
                                  names=('number', 'color'))
midx.isin([(1, 'red'), (3, 'red')])
# array([ True, False, False])
dates = ['2000-03-11', '2000-03-12', '2000-03-13']
dti = pd.to_datetime(dates)
dti.isin(['2000-03-11'])
# array([ True, False, False])

# i.argsort() 排序
# 将对索引进行排序的整数索引，见下文示例
idx = pd.Index(['b', 'a', 'd', 'c'])
order = idx.argsort() # array([1, 0, 3, 2])
idx[order] # Index(['a', 'b', 'c', 'd'], dtype='object')

# i.asof(2) 返回最近的索引，支持日期，可实现找最近日期
# 从索引中返回标签；如果不存在，则返回前一个标签
idx2 = pd.Index([1,3,6])
idx2.asof(5) # 3
idx2.asof(6) # 5
idx2.asof(-1) # nan
```

```
29
30  # index.get_loc 指定索引的位置，见示例
31  unique_index = pd.Index(list('abc'))
32  unique_index.get_loc('b') # 1
33  monotonic_index = pd.Index(list('abbc'))
34  monotonic_index.get_loc('b') # slice(1, 3, None)
35  non_monotonic_index = pd.Index(list('abcb'))
36  non_monotonic_index.get_loc('b')
37  # array([False,  True, False,  True], dtype=bool)
```

更多的操作可以看官方文档。