ch01.基本对象的数据结构

pandas中需要了解的4个基本对象：Index，MultiIndex，Series，DataFrame

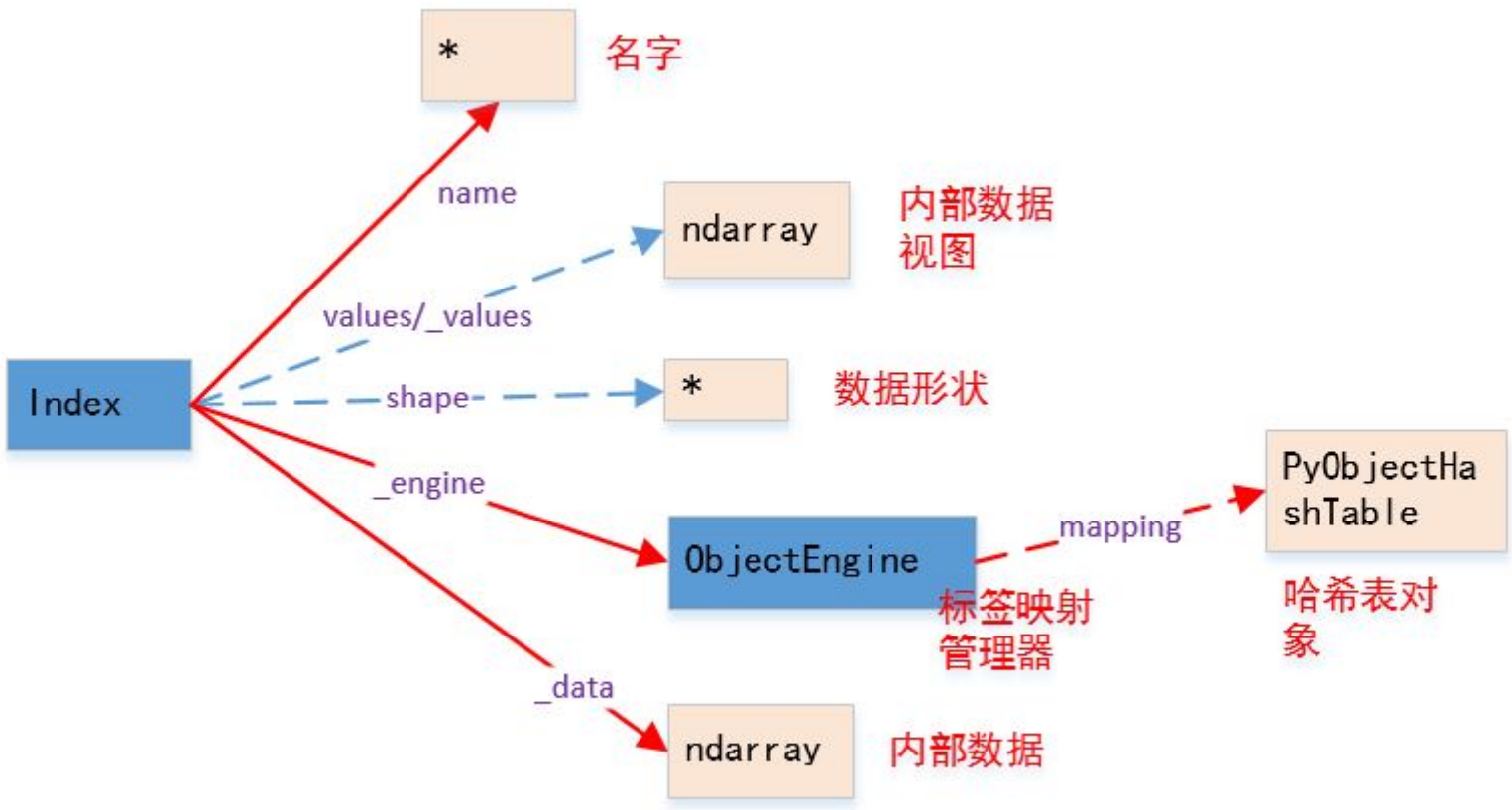
为什么需要了解它们的数据结构（内存管理，存储结构）：
有助于帮我们了解它们与其他数据类型的数据或者文件的数据交换，也能帮我们更好的了解它们的基本方法（算法）

注1：这4个最最基本的对象必须学懂，它们是后面一切的基础
注2：下面的图可能是老版本的pandas，个别属性已经被移除了，有别的属性或者方法来代替，参考最新版官方API和1.2版本API
注3： 有些东西跟API上展示的不太一样，不清楚为什么

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

Out[1]:
```
'1.2.4'
```

一、Index的数据结构



其中，
实线表示一般属性，虚线表示property创建的属性
工作机制：
虚线所指属性的值，一般是从实线那里获取，实线属性是我们关注的重点

官方API

1.name为普通属性，返回Index的名字

2.values/._values为property属性，返回Index的内部数据的视图

3._data为普通属性，返回Index的内部数据

4.shape为property属性，返回内部数据的形状

5._engine为标签映射管理器，它负责管理label和下标之间的映射,通过mapping的get_item可以获得标签的整数下标
最新版的pandas里面移除了这个属性，详情见官方API。取而代之可以用.get_loc()方法

example1--Index的初始化

先将列表数据传入_data，把字符串传入names

In [2]:
```
header1=pd.Index(['国家','种族','战斗力'],name='3年A班')
header2=pd.Index(['国家','国家','种族','战斗力'],name='3年A班')
header3=pd.Index(['国家','种族','战斗力','国家'],name='3年A班')
header1,header2,header3
```

Out[2]:
```
(Index(['国家', '种族', '战斗力'], dtype='object', name='3年A班'),
 Index(['国家', '国家', '种族', '战斗力'], dtype='object', name='3年A班'),
 Index(['国家', '种族', '战斗力', '国家'], dtype='object', name='3年A班'))
```

example2--查看属性

按照上图从上到下依次查看

In [3]:
```
header1.name,header1.values,header1._values,header1.shape,header1._data
```

Out[3]:
```
('3年A班',
 array(['国家', '种族', '战斗力'], dtype=object),
 array(['国家', '种族', '战斗力'], dtype=object),
 (3,),
 array(['国家', '种族', '战斗力'], dtype=object))
```

example3--使用 pandas.Index.get_loc() 获取索引

Index.get_loc(一个标签)
返回所请求标签的索引：整数位置、切片或布尔掩码

整数位置：就是一个整数，比如0，表示在第0个位置
切片：单调的索引切片，比如0到3
布尔掩码：不是连续的切片，中间有间断，比如0，2，4

其中，切片和布尔掩码都是以bool值的array展示

官方API

In [4]:
```
header1.get_loc('国家')  ## 国家不是重复元素，返回国家的下标
```

Out[4]:
```
0
```

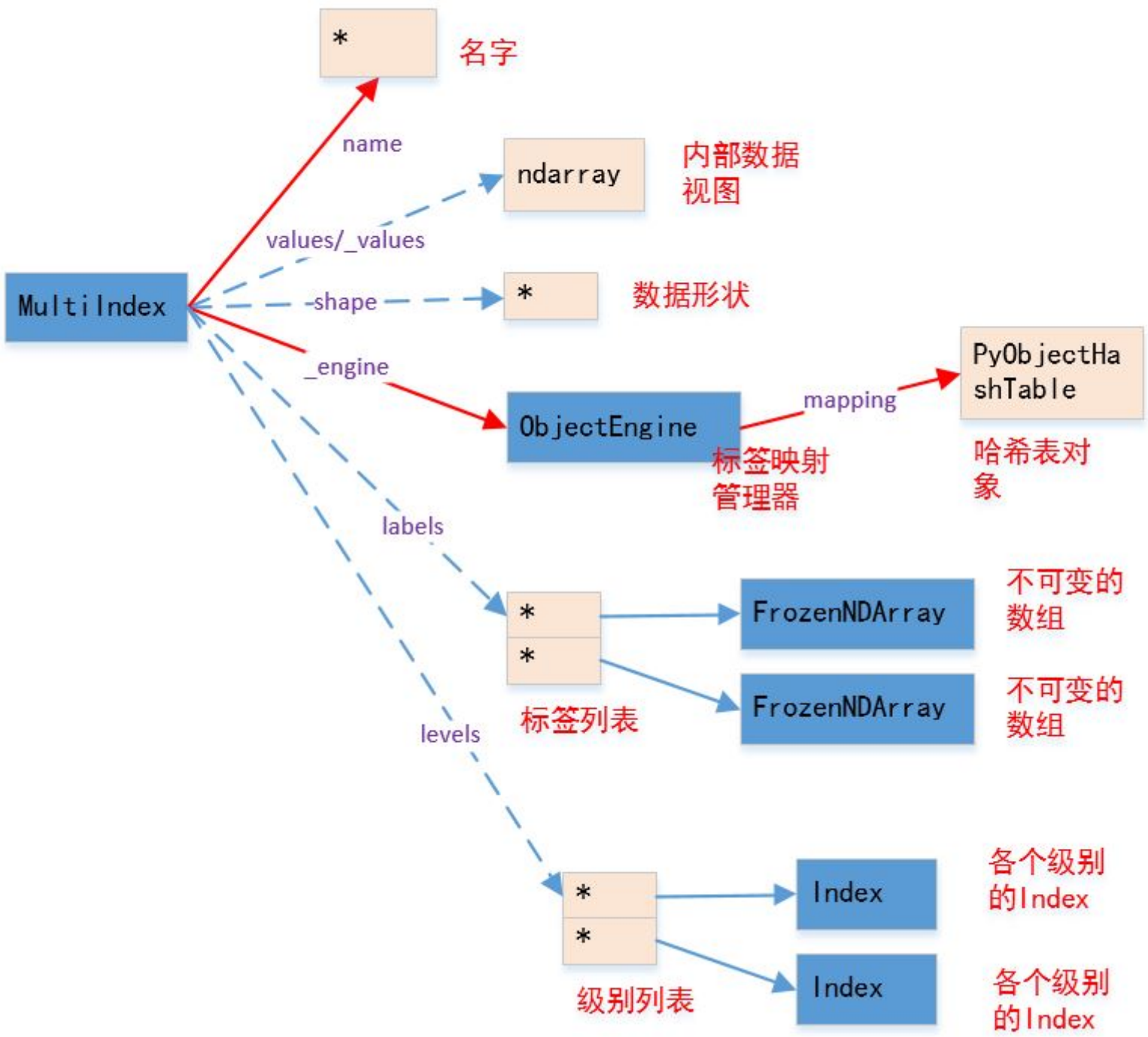
In [5]:
```
header2.get_loc('国家') , header3.get_loc('国家')  ## 国家是重复元素，所以返回的是bool元素的array
```

Out[5]:
```
(array([ True,  True, False, False]), array([ True, False, False,  True]))
```

In [6]:
```
cols = ['国家', '战斗力']
[header1.get_loc(c) for c in cols if c in header1]   ## 获取多个元素的下标
```

Out[6]:
```
[0, 2]
```

二、MultiIndex的数据结构



官方API

1.name为普通属性，返回MultiIndex的名字。同Index

2.values/._values为property属性，返回MultiIndex的内部数据的视图。同Index

3._data为None，这里是与Index不同。

4.shape为property属性，返回内部属性的形状 。同Index

5._engine已弃用,取而代之使用.get_loc()和get_locs()方法，这里不举例说明了，因为这个方法不是给你用的

6.labels为property属性，它返回一个FrozenList（不可变列表），列表中存储每一级的label对应的下标（也就是创建MultiIndex时传入的labels参数），以FrozenNDArray的数据类型。 最新版已移除

7.levels为property属性，它返回一个FrozenList（不可变列表），列表中存储每一级的label（也就是创建MultiIndex时传入的levels参数），以Index的数据类型。

example1--MultiIndex的初始化

- 第1种初始化，按一对一对元组传入

In [7]:
```
tup=[('男','百里守约'),('男','项羽'),('男','猪八戒'),('男','蔡徐坤'),('女','上官婉儿'),('女','小乔'),('女','蔡文姬')]
index_col1=pd.MultiIndex.from_tuples(tup,names=['性别','姓名'])
index_col1
```

Out[7]:
```
MultiIndex([('男', '百里守约'),
            ('男',  '项羽'),
            ('男', '猪八戒'),
            ('男', '蔡徐坤'),
            ('女', '上官婉儿'),
            ('女',  '小乔'),
            ('女', '蔡文姬')],
           names=['性别', '姓名'])
```

In [8]:
```
index_col1.levels,index_col1.levels[0],index_col1.levels[1],index_col1.nlevels
```

Out[8]:
```
(FrozenList([['女', '男'], ['上官婉儿', '小乔', '猪八戒', '百里守约', '蔡徐坤', '蔡文姬', '项羽']]),
 Index(['女', '男'], dtype='object', name='性别'),
 Index(['上官婉儿', '小乔', '猪八戒', '百里守约', '蔡徐坤', '蔡文姬', '项羽'], dtype='object', name='姓名'),
 2)
```

- 第2种初始化，按两个列表传入

In [9]:
```
list1=['男','男','男','男','女','女','女','女']
list2=['百里守约','项羽','猪八戒','蔡徐坤','上官婉儿','小乔','蔡文姬','上官婉儿']
index_col2=pd.MultiIndex.from_arrays([list1,list2],names=['性别','姓名'])
index_col2
```

Out[9]:
```
MultiIndex([('男', '百里守约'),
            ('男',  '项羽'),
            ('男', '猪八戒'),
            ('男', '蔡徐坤'),
            ('女', '上官婉儿'),
            ('女',  '小乔'),
            ('女', '蔡文姬'),
            ('女', '上官婉儿')],
           names=['性别', '姓名'])
```

- 第3种初始化，两个列表笛卡尔积（构成的一对一对元组）传入

In [10]:
```
list3=['语文成绩','数学成绩']
list4=['百里守约','项羽','猪八戒','蔡徐坤','上官婉儿','小乔','蔡文姬']
index_col3=pd.MultiIndex.from_product([list3,list4],names=['学科','姓名'])
index_col3
```

Out[10]:
```
MultiIndex([('语文成绩', '百里守约'),
            ('语文成绩',  '项羽'),
            ('语文成绩', '猪八戒'),
            ('语文成绩', '蔡徐坤'),
            ('语文成绩', '上官婉儿'),
            ('语文成绩',  '小乔'),
            ('语文成绩', '蔡文姬'),
            ('数学成绩', '百里守约'),
            ('数学成绩',  '项羽'),
            ('数学成绩', '猪八戒'),
            ('数学成绩', '蔡徐坤'),
            ('数学成绩', '上官婉儿'),
            ('数学成绩',  '小乔'),
            ('数学成绩', '蔡文姬')],
           names=['学科', '姓名'])
```

example2--pandas.MultiIndex.get_locs（（参数1，参数2））

参数1：对应第一列，可以是一个标签，一个切片，一个bool类型的mask列表

标签：'蔡徐坤'

slice切片：np.s_['项羽','小乔']或者('项羽', '小乔')

mask列表：[True,False,False,True]

注1：当单个参数不是元组时，填写两个参数外面的那个圆括号可以省略，但是如果单个参数已经是元组了，
比如slice切片，这时候外面那个圆括号不能省略，否者就会被认为是三元组，也就是三个参数了

注2：这里与官方API写的稍有不同,因为两个levels的Index顺序是被打乱的，直接按照官方API操作会出错

In [11]:
```
index_col2     ##这是下面操作的对象
```

Out[11]:
```
MultiIndex([('男', '百里守约'),
            ('男',   '项羽'),
            ('男',  '猪八戒'),
            ('男',  '蔡徐坤'),
            ('女', '上官婉儿'),
            ('女',   '小乔'),
            ('女',  '蔡文姬'),
            ('女', '上官婉儿')],
           names=['性别', '姓名'])
```

- 1.两个参数都是单个标签

In [12]:
```
index_col2.get_locs( ('男', '项羽'))  ## 获得了矩阵的列标1，供矩阵使用
```

Out[12]:
```
array([1], dtype=int64)
```

- 2.第二个参数为slice切片：

In [13]:
```
slice(None),np.s_['项羽','小乔']
```

Out[13]:
```
(slice(None, None, None), ('项羽', '小乔'))
```

In [14]:
```
index_col2.get_locs( ('女', slice(None)))  ## 获得了矩阵的列标4，5，6，供矩阵使用
```

Out[14]:
```
array([4, 5, 6, 7], dtype=int64)
```

In [15]:
```
index_col2.get_locs( ('男', np.s_['项羽','小乔']) )
```

Out[15]:
```
array([1], dtype=int64)
```

- 3.第二个参数是掩码列表

In [16]:
```
index_col2.get_locs( ('男', [True,False,True,False,True,False,True,False] ) )
```

Out[16]:
```
array([0, 2], dtype=int64)
```

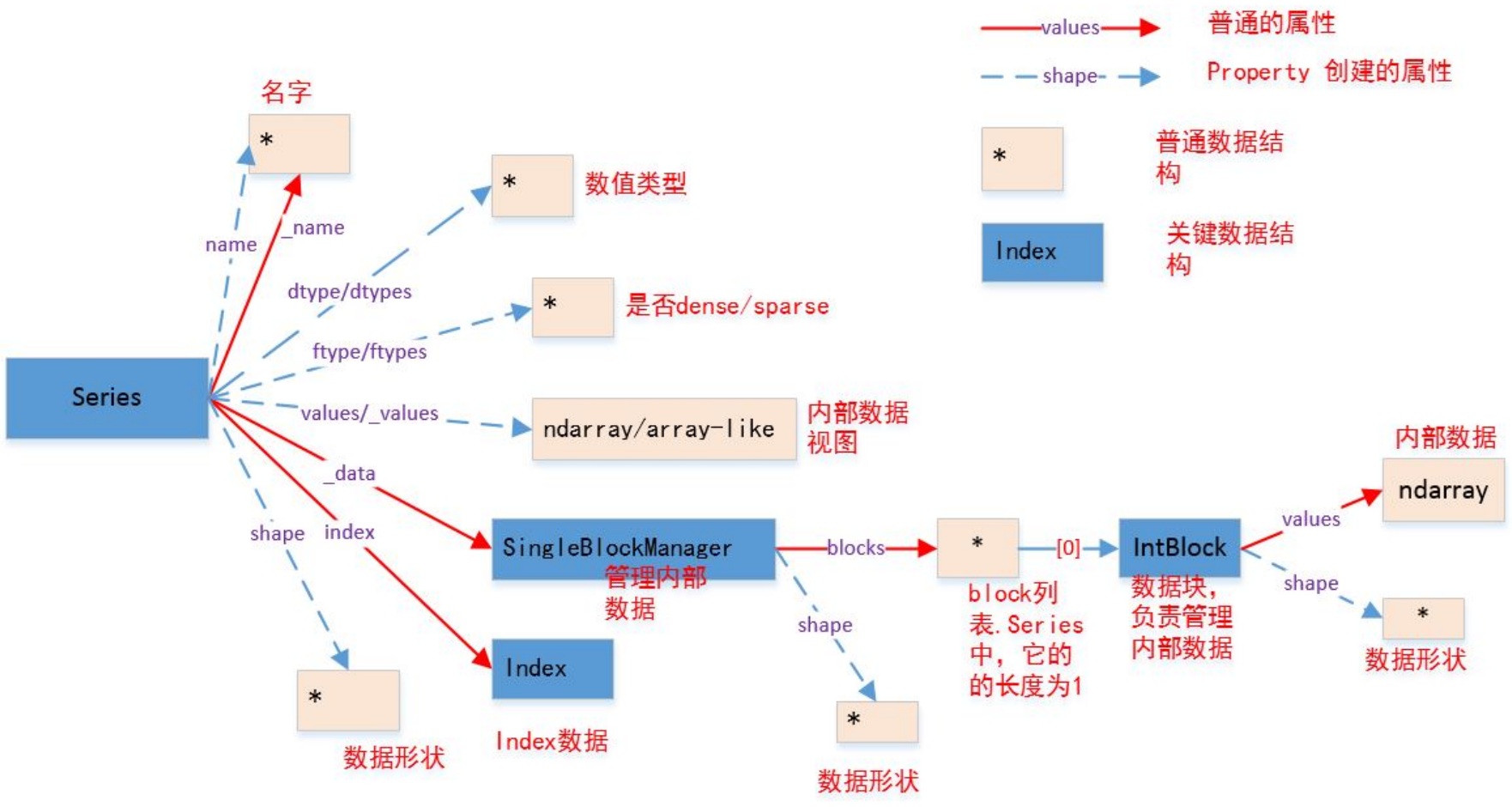
小结：

- 创建Index和MultiIndex：
  传一个list或者两个list

- Index.get_loc(参数)
  参数只能是一个标签
  返回的是整数下标array或者bool值的array

- MultiIncex.get_locs（参数1，参数2）
  单个参数可以是：一个标签，一个元组表示的切片，一个bool列表
  返回的是整数下标的array

三、Series的数据结构



官方API

1._name为普通属性，返回Seris的名字；.name为property属性，返回的也是Seris名字

2.dtype/.dtypes为property属性，返回Series的数据类型。

3.ftype/ftypes为property属性，返回一个字符串，说明Series是否稀疏数据。（二者返回的字符串的值相同，但不是同一个字符串对象）

4.values/._values为property属性，返回Series的内部数据的视图

5.index为普通属性，返回Series的索引

6.shape为property属性，返回Series的数据的形状

7._data为普通属性，它返回的是一个SingleBlockManager对象，该对象负责管理内部数据。
SingleBlockManager的.shape属性为property属性，返回内部数据的形状
SingleBlockManager的.blocks属性为普通属性，返回一个列表，该列表只有一个元素，该元素为一个IntBlock对象（或者其他的xxxBlock对象），代表了内部数据。
IntBlock的.values属性为普通属性，它返回内部数据：一个ndarray。
IntBlock的.shape属性为property属性，它返回内部数据的形状

example1--单级索引的Series初始化

class pandas.Series(data=None, index=None, dtype=None, name=None, copy=False,fastpath=False)

参数：

1.data：它可以是一个字典、array-like、标量。表示Series包含的数据，如果是序列/数组，则它必须是一维的
如果是字典，则字典的键指定了label。如果你同时使用了index，则以index为准。
如果是标量，则结果为：该标量扩充为index长度相同的列表。

2.index：一个array-like或者一个Index对象。它指定了label。其值必须唯一而且hashable，且长度与data一致。
如果data是一个字典，则index将会使用该字典的key（此时index不起作用）。如果未提供，则使用np.arange(n)。

3.name：一个字符串，为Series的名字。

4.dtype：指定数据类型。如果为None，则数据类型被自动推断

5.copy：一个布尔值。如果为True，则拷贝输入数据data

In [17]:
```
# （1）由字典创建，字典的key就是index，values就是values
dic = {'a':1 ,'b':2 , 'c':3, '4':4, '5':5}
s = pd.Series(dic)
s
```

Out[17]:
```
a    1
b    2
c    3
4    4
5    5
dtype: int64
```

In [18]:
```
# （2）由数组创建（一维数组）
arr = np.random.randn(5)
```

```
s = pd.Series(arr)
arr,s
```

Out[18]:
```
(array([ 0.1980849 , -0.55649407,  0.00830882,  1.05311009,  0.12881228]),
 0    0.198085
 1   -0.556494
 2    0.008309
 3    1.053110
 4    0.128812
 dtype: float64)
```

In [19]:
```
s=pd.Series(arr, index = ['a','b','c','d','e'],dtype = np.object)
s
```

```
C:\Users\Amadeus\AppData\Local\Temp\ipykernel_27936\2992262385.py:1: DeprecationWarning: `np.object` is a deprecated alias for the builtin `object`. To silence this warning, use `object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  s=pd.Series(arr, index = ['a','b','c','d','e'],dtype = np.object)
```

Out[19]:
```
a    0.198085
b   -0.556494
c    0.008309
d     1.05311
e    0.128812
dtype: object
```

In [20]:
```
# （3）由标量创建
s = pd.Series(10, index = range(4))
s
```

Out[20]:
```
0    10
1    10
2    10
3    10
dtype: int64
```

example2--两级索引的Series初始化

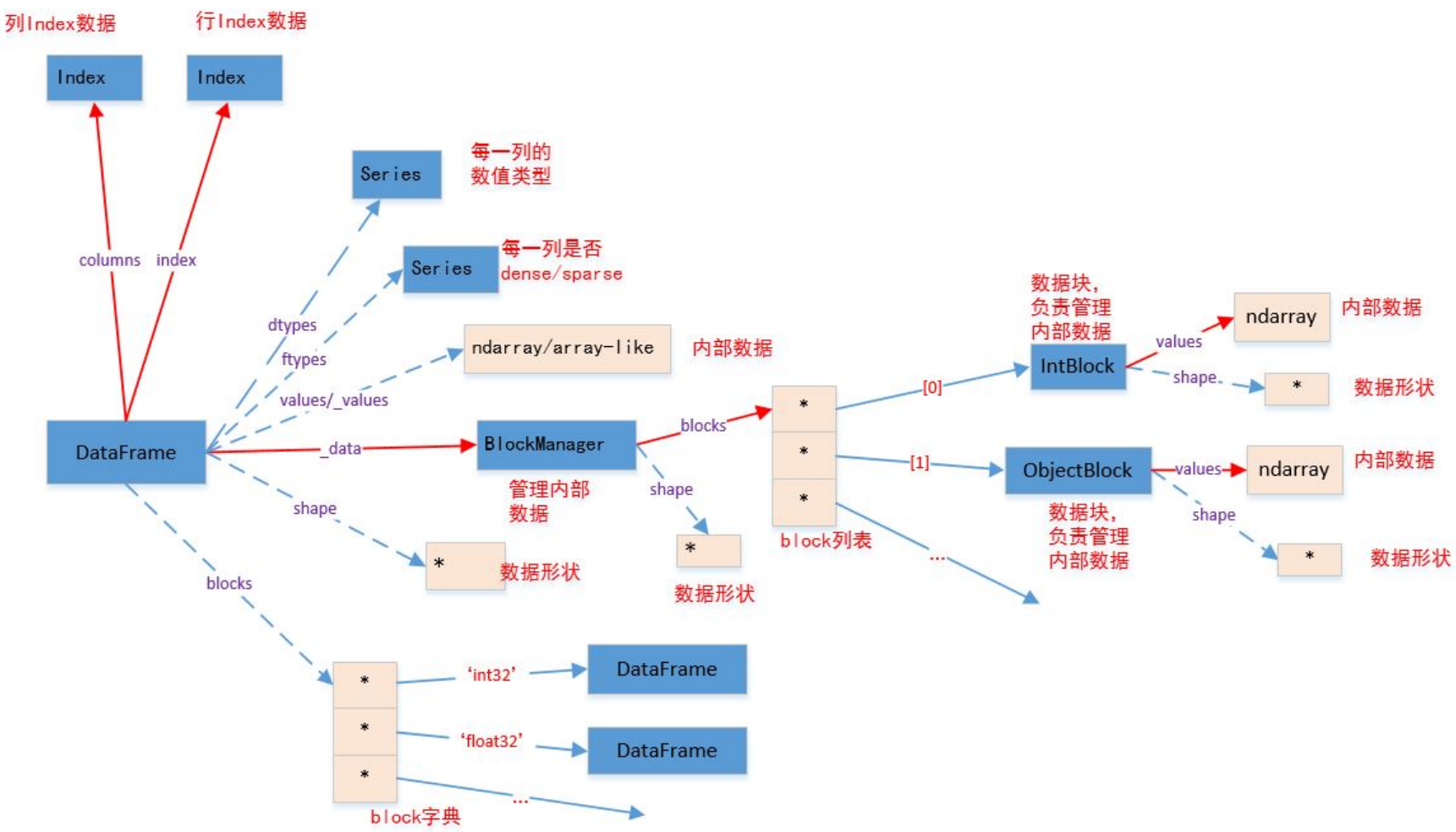
In [21]:
```
# （1）data使用一维的array，index使用两个list
sel=pd.Series(data=np.random.randn(4),index=[list("aabb"),[1,2,1,2]])
sel
```

Out[21]:
```
a  1   -2.258218
   2    0.616072
b  1    0.463946
   2    0.327015
dtype: float64
```

In [22]:
```
# （2）data是一维的list，index使用MultiIndex
se2=pd.Series(data=[88,56,94,44],index=pd.MultiIndex.from_arrays([list("aabb"),[1,2,1,2]]))
se2
```

Out[22]:
```
a  1    88
   2    56
b  1    94
   2    44
dtype: int64
```

一、DataFrame的数据结构



官方API

1.index/columns属性都为普通属性，它们返回的都是一个Index对象，参考Series。

2.dtypes属性为property属性，给出了每列的数值类型。它返回的是一个Series。并且没有.dtype属性，这一点与Series不同。

3.ftypes属性为property属性，给出了每列是否为sparse/dense的。它返回的是一个Series。并且没有.ftype属性，这一点与Series不同。

4.values/._values/.shape属性都为property属性，参考Series。

5._data属性为普通属性，它返回的是一个BlockManager对象，该对象负责管理内部数据。该对象的.block属性（普通属性）返回一个列表，该列表里面有多个元素。 DataFrame尽量用一个数组保存类型相同的列。
每个元素都为一个xxBlock对象。如IntBlock/FloatBlock...
一个xxBlock 可能存储多个列的数据（这些列的数据都是同一个类型）
xxBlock对象的.values属性（普通属性）就是存储的某个列（或者某些类型相同的列）的内部数据，一个ndarray
xxBlock对象的.shape属性（property属性）就是存储的某个列（或者某些类型相同的列）的内部数据的形状

6.blocks属性为property属性。该属性返回一个字典，该字典的键为不同的数值类型，该字典的值为该数值类型的数值组成的DataFrame

example1--DataFrame初始化

方法一：分别指定data（数据），index（行索引），columnx（列索引）即可

data：二维list或者array
index：list，array或者Index，MultiIndex
columns：list，array或者Index，MultiIndex

In [23]:
```
df1=pd.DataFrame(np.random.randint(0,10,[4,2]),index=['A','B','C','D'],columns=['a','b'])
df1
```

Out[23]:

	a	b
A	3	1
B	5	8
C	2	7
D	5	0

In [24]:
```
header1.name='属性'
header1._data[0]='体力值'     #如果直接通过index的索引是无法修改其值的，这里通过_data属性直接找到内存修改
df2=pd.DataFrame(data=np.random.randint(1,100,[7,3]),index=index_col1,columns=header1)
df2
```

```
Canceled future for execute_request message before replies were done
The Kernel crashed while executing code in the the current cell or a previous cell. Please review the code in the cell(s) to identify a possible cause of the failure. Click <a href='https://aka.ms/vscodeJupyterKernelCrash'>here</a> for more info. View Jupyter <a href='command:jupyter.viewOutput'>log</a> for further details.
```

方法二：从字典到DataFrame

下面使用的是DataFrame的初始化方法，默认字典的key是列索引，值是一列数据
DataFrame.from_dict方法具有更丰富的功能，这里不进行介绍了，知道就行，一般用不上

In [ ]:
```
dict={'a':[1,2,3,4],'b':[5,6,7,8]}
df1=pd.DataFrame(dict)
df2=pd.DataFrame(dict,index=['A','B','C','D'])

## df3=pd.DataFrame.from_dict(dict)      ##默认字典的key为columns，但奇怪的是这里无法指定行索引
## df4=pd.DataFrame.from_dict(dict,orient='index',columns=['A','B','C','D'])  ##设置字典的key为index，columns额外指定
df1,df2
```

Out[ ]:
```
(   a  b
 0  1  5
 1  2  6
 2  3  7
 3  4  8,
    a  b
 A  1  5
 B  2  6
 C  3  7
 D  4  8)
```

方法三：从Series到DataFrame
默认会把Series的索引作为DataFrame的行索引，列索引可以手动指定，不指定会默认设置为数字索引

In [ ]:
```
se1=pd.Series(data=[1,2,3,4],index=['a','b','c','d'])
se2=pd.Series(data=np.random.randn(4),index=[list("aabb"),[1,2,1,2]])
df1=pd.DataFrame(se1,columns=['QQ'])
df2=pd.DataFrame(se2,columns=['QQ'])
df1,df2
```

Out[ ]:
```
(   QQ
 a   1
 b   2
 c   3
 d   4,
             QQ
 a 1 -0.183538
   2  0.412109
 b 1  0.547671
   2 -0.295981)
```

example2--直接访问三个属性的内存管理对象（通过这三个对象，可以分别访问它们数据的内存）

In [ ]:
```
df1._data,df1._data.blocks[0]
```

Out[ ]:
```
(BlockManager
 Items: Index(['体力值', '种族', '战斗力'], dtype='object', name='属性')
 Axis 1: MultiIndex([('男', '百里守约'),
             ('男',   '项羽'),
             ('男',  '猪八戒'),
             ('男',  '蔡徐坤'),
             ('女', '上官婉儿'),
             ('女',   '小乔'),
             ('女',  '蔡文姬')],
            names=['性别', '姓名'])
 IntBlock: slice(0, 3, 1), 3 x 7, dtype: int32,
 IntBlock: slice(0, 3, 1), 3 x 7, dtype: int32)
```

In [ ]:
```
df1.index,df1.index.levels[0]
```

Out[ ]:
```
(MultiIndex([('男', '百里守约'),
             ('男',   '项羽'),
             ('男',  '猪八戒'),
             ('男',  '蔡徐坤'),
             ('女', '上官婉儿'),
             ('女',   '小乔'),
             ('女',  '蔡文姬')],
            names=['性别', '姓名']),
 Index(['女', '男'], dtype='object', name='性别'))
```

In [ ]:
```
df1.columns,df1.columns._data
```

Out[ ]:
```
(Index(['体力值', '种族', '战斗力'], dtype='object', name='属性'),
 array(['体力值', '种族', '战斗力'], dtype=object))
```