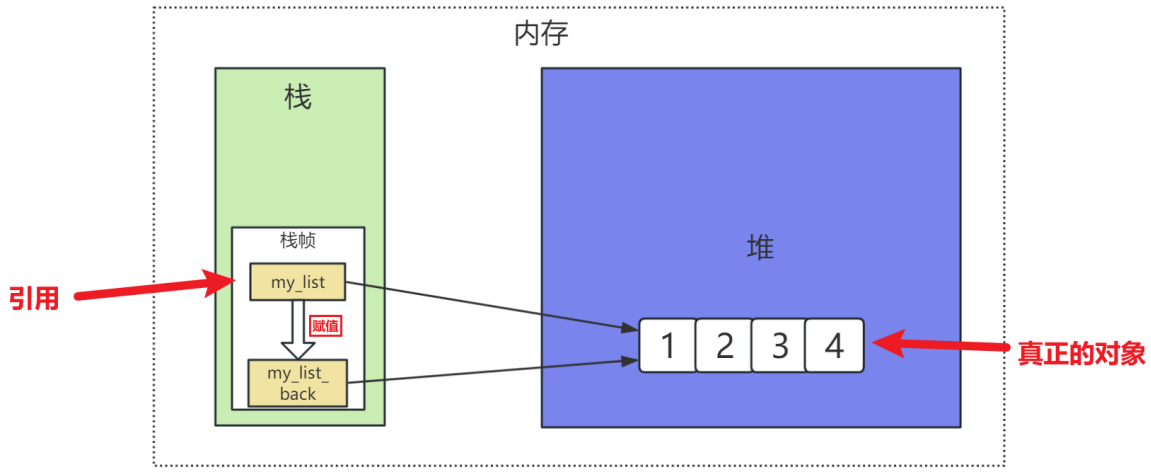
什么是拷贝？

在 Python 中，直接用 `=` 赋值**不是拷贝**，而是给原数据起“别名”（共享同一块内存）。修改一个变量，会直接影响另一个变量（对于可变数据类型），这在处理复杂数据（如嵌套列表、字典）时容易出问题。

举个例子：

```
my_list = [1,2,3,4]
my_list_copy = my_list
```



等号（=） 这种操作是赋值操作，在赋值的时候，其实是赋予的引用保存的地址值

代码示例：

```
# 原数据：嵌套列表（表层列表+内层列表）
original = [1, 2, [3, 4]]

# 直接赋值（不是拷贝）
alias = original

# 修改alias的表层元素
alias[0] = 100

# 修改alias的嵌套元素
alias[2][0] = 300

# 查看原数据：被影响了！
print("原数据original: ", original)   # 输出：[100, 2, [300, 4]]
print("别名alias: ", alias)           # 输出：[100, 2, [300, 4]]
```

问题根源： `original` 和 `alias` 指向同一块内存，修改任何一个都会同步影响另一个。

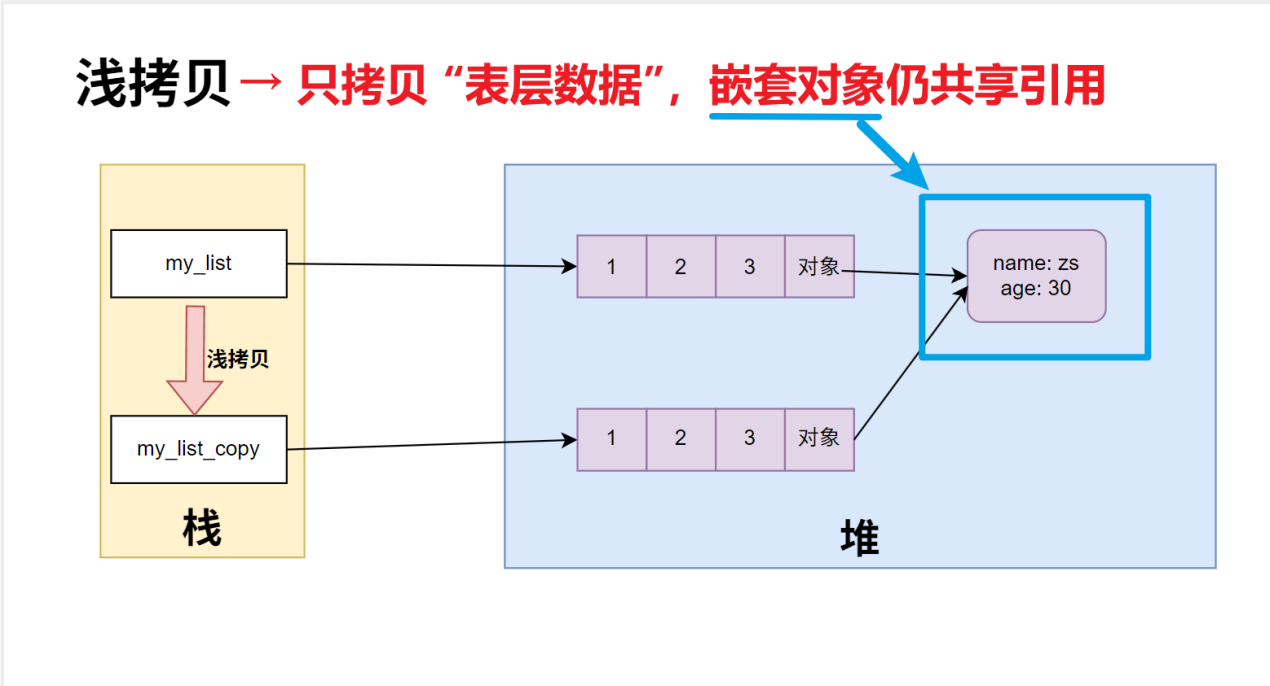
解决方案： 用“拷贝”创建独立的副本，避免相互干扰 —— 这就需要区分深拷贝和浅拷贝。

深拷贝和浅拷贝是什么？

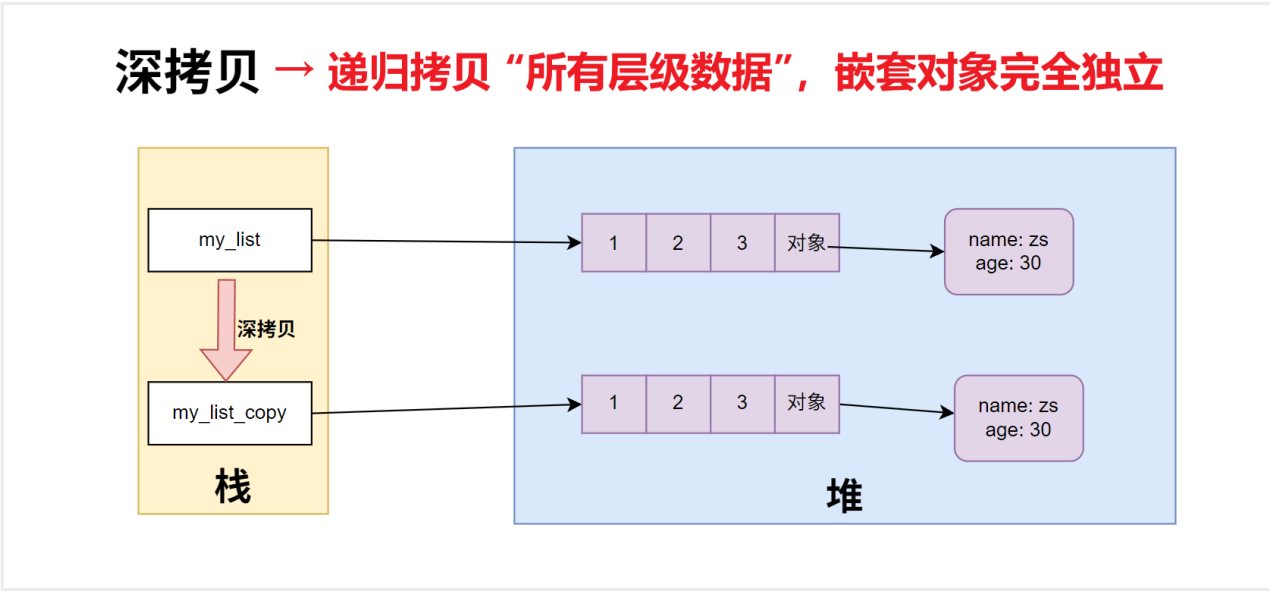
浅拷贝和深拷贝的本质差异，在于**是否拷贝“嵌套对象”的内部数据**：

对比维度	浅拷贝（Shallow Copy）	深拷贝（Deep Copy）
拷贝范围	只拷贝“表层数据”，嵌套对象仍共享引用	递归拷贝“所有层级数据”，嵌套对象完全独立
数据独立性	表层数据独立，嵌套数据仍关联	所有层级数据都独立，与原数据无关联
适用场景	数据无嵌套（如单层列表、字典）	数据有嵌套（如列表套列表、字典套字典）
效率	速度快，占用内存少	速度慢，占用内存多（需拷贝所有层级）

浅拷贝图解



深拷贝图解



浅拷贝

浅拷贝会创建表层数据的副本，但嵌套对象（如内层列表、子字典）依然是原数据的引用，修改嵌套对象会同步影响原数据。

浅拷贝的 3 种实现方式

```
import copy  # 深拷贝需用copy模块，浅拷贝也可通过模块实现

# 原数据（嵌套列表）
original = [1, 2, [3, 4]]

# 方式1：列表切片（[:]，仅适用于列表）
shallow1 = original[:]

# 方式2：list.copy()方法（适用于列表）
shallow2 = original.copy()

# 方式3：copy模块的copy()函数（通用，适用于列表、字典等）
shallow3 = copy.copy(original)
```

重点掌握方式3

浅拷贝-半独立

举个例子：

```
import copy

original = [1, 2, [3, 4]]

shallow = copy.copy(original)  # 浅拷贝

# 1. 修改表层数据：浅拷贝的表层独立，不影响原数据
shallow[0] = 100

print("修改表层后-原数据：", original)  # 输出：[1, 2, [3, 4]]（原数据表层不变）
print("修改表层后-浅拷贝：", shallow)    # 输出：[100, 2, [3, 4]]（拷贝表层变）

# 2. 修改嵌套数据：浅拷贝的嵌套仍共享，影响原数据
shallow[2][0] = 300

print("修改嵌套后-原数据：", original)  # 输出：[1, 2, [300, 4]]（原数据嵌套变）
print("修改嵌套后-浅拷贝：", shallow)  # 输出：[100, 2, [300, 4]]（拷贝嵌套变）
```

结论：浅拷贝只解决“表层数据独立”，嵌套数据依然“牵一发而动全身”。

深拷贝

深拷贝会递归遍历所有层级的数据，包括嵌套对象，创建完全独立的副本 —— 修改任何层级的数据，都不会影响原数据。

1. 深拷贝的实现

深拷贝通过`copy`模块的`deepcopy()`函数实现：

```
import copy

original = [1, 2, [3, 4]]

deep = copy.deepcopy(original)  # 深拷贝
```

2. 深拷贝-完全独立

```
import copy

original = [1, 2, [3, 4]]

deep = copy.deepcopy(original)  # 深拷贝

# 1. 修改表层数据：深拷贝独立，不影响原数据
deep[0] = 100
print("修改表层后-原数据：", original)  # 输出：[1, 2, [3, 4]]（不变）
print("修改表层后-深拷贝：", deep)      # 输出：[100, 2, [3, 4]]（变）

# 2. 修改嵌套数据：深拷贝独立，不影响原数据
deep[2][0] = 300
print("修改嵌套后-原数据：", original)  # 输出：[1, 2, [3, 4]]（不变！）
print("修改嵌套后-深拷贝：", deep)      # 输出：[100, 2, [300, 4]]（变）
```

结论：深拷贝是“彻底的拷贝”，原数据和拷贝数据完全独立，无任何关联。

注意事项

浅拷贝对不可变类型和可变类型的copy不同。

- copy.copy()对于可变类型，会进行浅拷贝
- copy.copy()对于不可变类型，不会拷贝数据，仅仅是拷贝引用并指向对象

```
import copy

data = [1,2,3,4]          # 列表可变
# data = (1,2,3,4)        # 元组不可变
copy_data = copy.copy(data)

# 当浅拷贝列表的时候，两个对象的地址值不一样
# 当浅拷贝元组的时候，两个对象的地址值一样
print(id(my_tuple))
print(id(my_tuple_back))
```