"""

python 基础

1. 运行方式

交互式/文件式

2. 运行过程:

源代码 -- 编译 --> 字节码 -- 解释 --> 机器码

第一次 pyc

导入模块

主模块存放少量代码(不编译)

3. python 快捷键(百度搜索)

4. python 内存管理器

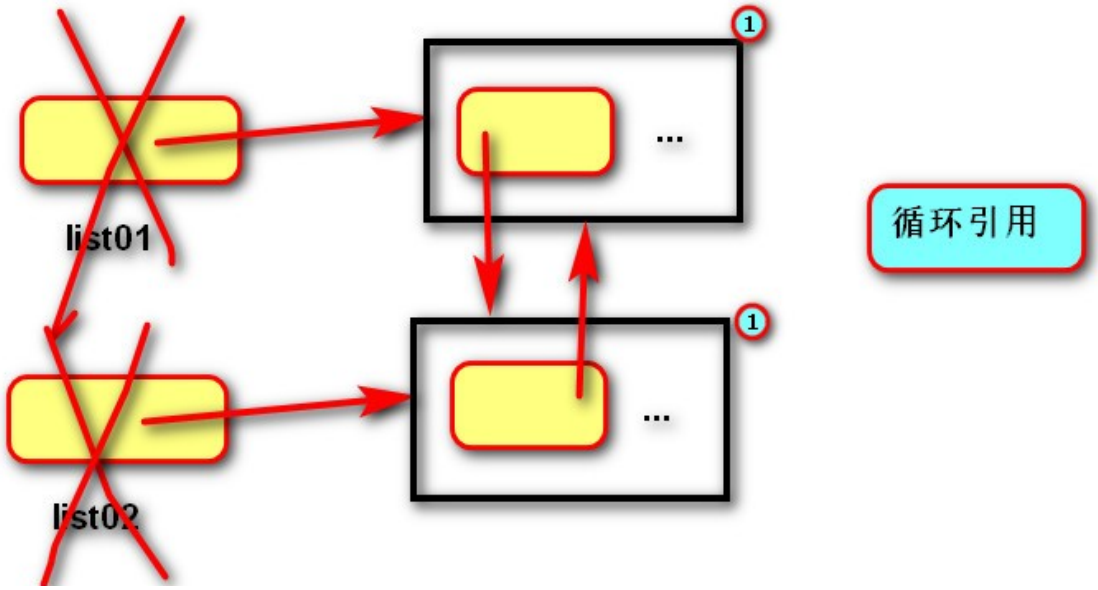
引用计数:每个对象存储被引用的次数,如果数量为 0,则销毁.

a = 对象()

b = a

缺点:不能解决循环引用的问题,意味着浪费内存.

```
list01 = []
list02 = []
list01.append(list02)
list02.append(list01)
del list01,list02
```



标记清除:在内存容量不够时,从栈中开始扫描内存,标记可以访问到的对象.

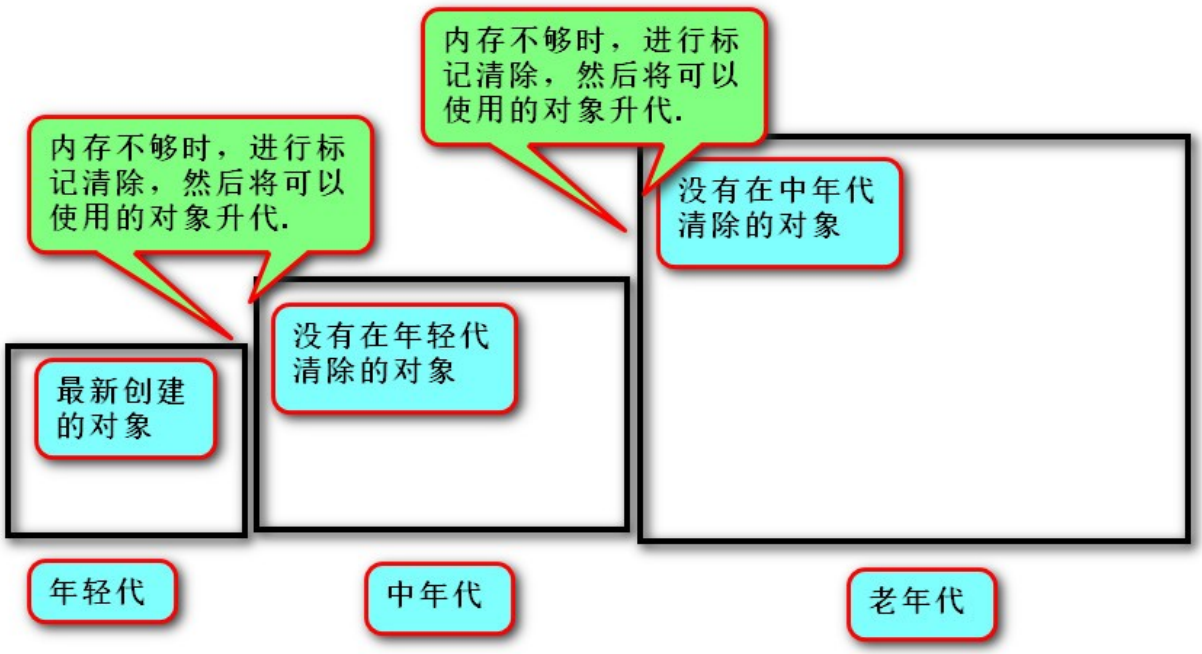
销毁没有标记的对象.

缺点:扫描内存耗时

分代回收:根据回收频次将内存分为多个区域(代),避免标记清除时扫描范围过大.

内存优化:

尽少产生垃圾,对象池(整数/字符串..),手动回收(慎重),



5. 容器

种类:字符串/列表(预留空间)元组(按需分配)/字典(单个元素读写速度快)/集合(去重复/数学运算)

内存图

相互转换:join+split/list()/tuple()/dict([(k,v),(k,v)])/set()

通用操作:+ * 比较 in 索引/切片

获取所有元素

6. 函数

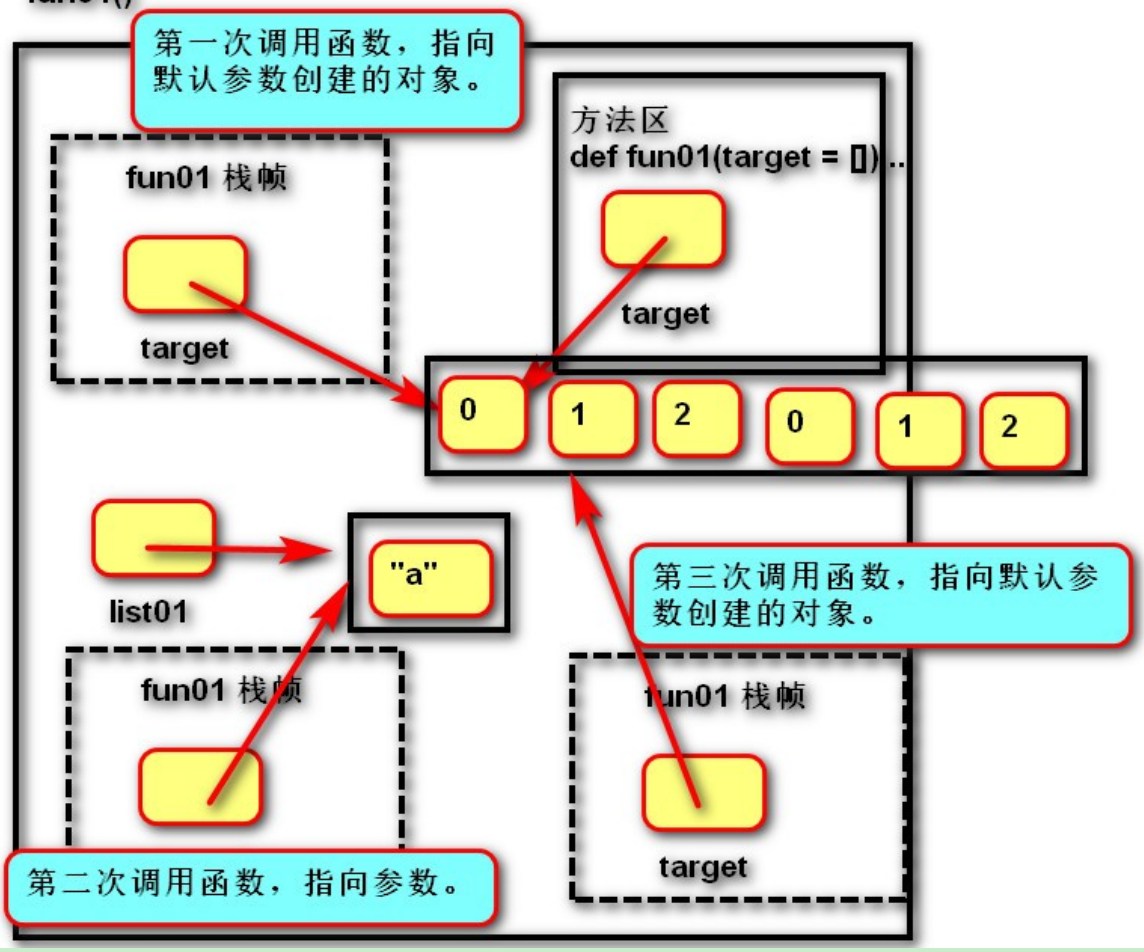
设计:单一

```
        参数:
            实参
                位置(1,2,3)
                    序列实参(*(1,2,3))
                关键字(a=1,b=2)
                    (**{"a":1,"b":2})
            形参
                默认(a,b=0)
                位置(a,b)
                    星号元组形参(*args)
                关键字(*args,a) (*,a,b)
                    双星号字典形参(**kwargs)
            万能参数(*args,**kwargs)
"""
#  4. python 内存管理器
list01 = []
list02 = []
list01.append(list02)
list02.append(list01)
del list01, list02  # 循环引用
# 5. 容器
list03 = ["a", "b", "c"]
# a--b--c
# list --> str
print("--".join(list03))
# str --> list
print("a--b--c".split("--"))
list01 = [1, 2]
print(id(list01))
list01 += [3]
```

```python
print(id(list01)) # 列表+=后,与之前是同一个对象 [可变]
tuple01 = (1, 2)
print(id(tuple01))
tuple01 += (3,)
print(id(tuple01)) # 元组+=后,创建新对象 [不可变]
dict01 = {"a": 1}
# dict01 += {"b":2}
# dict01.update({"b":2})# 字典累加前与后,是同一个对象 [可变]
# ** 将字典中的元素进行拆分
dict02 = {**dict01, **{"b": 2}} # 字典合并前与后,是新对象
print(dict02)
# *用于合并多余的元素
a, *b = 1, 2, 3, 4, 5
print(a, b)
# 索引(定位单个)
# 切片(定位多个)
list01 = [1, 2, 3, 4, 5]
# -- 读取(创建新列表  浅拷贝)
list02 = list01[1:4]
# 切片 = 可迭代对象
# 遍历可迭代对象,将每个元素存入切片位置
list01[1:4:2] = "ab"
print(list01)
for item in list01:
    print(item)
# for item in list01[::-1]:
#   print(item)
# 0 1 2 3 4
for i in range(len(list01) - 1, -1, -1):
```

```
    print(list01[i])
# 参数
# 将函数存入方法取时,创建默认参数对象(空列表)
# 只要实参不提供数据,则使用默认的同一个对象.
# 结论:默认参数不要使用可变对象
def fun01(target=[]):
    for i in range(3):
        target.append(i)
    print(target)
# list01 = ["a"]
fun01()  # [0, 1, 2]
# fun01(list01)# ['a', 0, 1, 2]
fun01()  # [0, 1, 2, 0, 1, 2] 默认参数只要不传入数据,都会使用加载模块时创建的对象.
```

```
def fun01(target = []):
    for i in range(3):
        target.append(i)
    print(target)
fun01()# [0, 1, 2]
list01 = ["a"]
fun01(list01)# ['a', 0, 1, 2]
fun01()
```



"""

17:15

面向对象

概述

面向过程:干　关心步骤(过程)

面向对象:找　关心谁(对象)?干嘛

经典案例:购物车 code/day09/day08_exercise/shopping

购物车 code/day15/shopping_oo

三大特征

封装:分

具体影响效果(负责处理具体功能)

技能释放器(负责根据需求创建具体影响效果)

继承:隔

影响效果(隔离释放器与具体效果的变化)

多态:执

具体影响效果重写影响效果

根据需求创建具体影响效果对象

六大原则

开闭:增加新效果,不影响其他代码.

单一:每个功能都只负责一个变化点

依赖倒置:释放器调用影响效果(父),不调用具体影响效果(子)

组合复用:释放器存储影响效果变量

里氏替换:...

迪米特:谁都不影响谁

经典案例:技能系统 code/day14/day13_exercise/exercise01

关系

泛化(继承)

组合(变量)

技能系统 code / day14 / day13_exercise / exercise01

class 影响效果

...

class 消耗法力(影响效果)

...

class ...(影响效果)

...

class 技能释放器

```
    准备技能()
        读取配置文件,根据技能名称,获取影响效果名称
        创建具体影响效果对象 eval("....")
    生成技能()
        调用影响效果,执行具体影响效果.
变量=技能释放器("降龙十八掌")
变量.生成技能()
"""
class Student:
    def __init__(self, name=""):
        self.name = name
s01 = Student("老大")
list01 = [
    s01,
    Student("老二"),
    Student("老三"),
]
def fun01(list_target):
    #修改列表第一个元素
    list_target[0] = Student("大哥")
    #修改列表第二个元素指向的对象
    list_target[1].name = "二哥"
    #修改栈帧中的局部变量
    list_target = Student("三弟")
fun01(list01)
print(s01.name)  #"老大"
print(list01[0].name)  #大哥
print(list01[1].name)  #二哥
print(list01[2].name)  #老三
```

"""

python 高级

1. 模块和包

导包的路径从项目根目录开始计算.

是否成功导入?取决于导包的路径与 sys.path 中的路径拼接后是否可以正确定位模块.

2. 异常处理

异常现象:报错后返回给调用者,不会向下执行.

处理目的:让异常变为正常(自上而下向后执行)

3. 迭代

```
class 可迭代对象:
    def __iter__(self):
        return 迭代器()
class 迭代器:
    def __next__(self):
        if 没有元素了:
            raise StopIteration
        return 数据
iterator = 可迭代对象.__iter__()
            iter(可迭代对象)
while True:
    try:
        item = iterator.__next__()
    except StopIteration:
        break
```

能被 for 的条件:对象具有__iter__函数

迭代设计思想:使用者只需通过一种方式，便可获取元素。

4.生成器

特点：执行一次，计算一次，返回一次

```
class 生成器:
```

```
            def __iter__(self):
                return self
            def __next__(self):
                if 没有元素了:
                    raise StopIteration
                return 数据
        生成器函数：通过yield语句将函数分割为多个__next__方法.
            def 名称():
                ...
                yield 数据
                ...
        生成器表达式：列表推导式的语法
        函数式编程
            函数作为参数
            函数作为返回值
                装饰器...
"""
def fun01():
    for i in range(5):
        data = yield i
        if data == "qtx":
            yield "q t x来啦"
iterator = fun01() #创建生成器对象
#__next__的返回值是yield后面的数据
print(iterator.__next__()) #获取一个数据
print(iterator.__next__()) #获取一个数据
#send的参数赋值给yield前面的变量
print(iterator.send("qtx"))
print(iterator.__next__()) #获取一个数据
```

```
print(iterator.send(""))# 获取一个数据
```