\_\_init\_\_与\_\_new\_\_这两个魔法方法组成了Python类对象的构造器，在Python类实例化时，其实最先调用的不是\_\_init\_\_而是\_\_new\_\_。\_\_new\_\_是负责实例化对象的，而\_\_init\_\_是初始化操作。\_\_del\_\_是析构器，当Python对象的所有引用都不存在了（被del了），就会自动触发\_\_del\_\_执行。

[复制代码](javascript:void(0);)

class CapStr(str):

def \_\_new\_\_(cls, string): #此时string = 'i love you' cls是CapStr这个类对象

string = string.upper()

return str.\_\_new\_\_(cls, string) #返回cls类带string参数的实例化的对象,但是这里为什么不直接return string ？看下面的代码段

def \_\_del\_\_(self):

print('\_\_del\_\_析构方法被调用了！')

>>> a = CapStr('i love you')  
>>> b = a  
>>> c = b  
>>> del c  
>>> del a  
>>> del b  
\_\_del\_\_析构方法被调用了！

[复制代码](javascript:void(0);)

由于CapStr类继承的是str这种不可修改的类型，所以，就不能在\_\_init\_\_方法中对str类型的数据进行自身的改变，这时候就要重写\_\_new\_\_来实现对不可变类型数据的修改。

1.a = CapStr('i love you')  
2.首先执行使用'i love you'这个参数来执行CapStr类的\_\_new\_\_方法，这个\_\_new\_\_方法会返回CapStr类的一个实例（通常情况下是使用 super(CapStr, cls).\_\_new\_\_(cls, ... ...) 这样的方式），  
3.然后利用这个实例来调用类的\_\_init\_\_方法，上一步里面\_\_new\_\_产生的实例也就是\_\_init\_\_里面的的self  
所以，\_\_init\_\_ 和 \_\_new\_\_ 最主要的区别在于：  
1.\_\_init\_\_ 通常用于初始化一个新实例，控制这个初始化的过程，比如添加一些属性， 做一些额外的操作，发生在类实例被创建完以后，它是实例级别的方法。  
2.\_\_new\_\_ 通常用于控制生成一个新实例的过程，它是类级别的方法。

[复制代码](javascript:void(0);)

class CapStr(str):

def \_\_new\_\_(cls, string):

string = string.upper()

return string  
  
a = CapStr('mmy')

>>> type(a)  
<class 'str'>

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

class CapStr(str):

def \_\_new\_\_(cls, string):

string = string.upper()

return str.\_\_new\_\_(cls, string)

>>> a = CapStr('mmy')  
>>> type(a)  
<class '\_\_main\_\_.CapStr'>

[复制代码](javascript:void(0);)

可以看出，二者返回的对象是不一样的。

注：str.\_\_new\_\_(cls, string)的实际意思是str(string)，使用工厂函数str()将参数强制转换成str类型，然后再变成str的子类型CapStr