Mixin 是多继承的一种形式，其来自多个父类的行为和属性可以组合在一起。

<https://www.cnblogs.com/aademeng/articles/7262520.html>

<http://www.php.cn/python-tutorials-88720.html>

<https://www.cnblogs.com/eric-nirnava/p/mixin.html>

[python用法——Mixin](https://www.cnblogs.com/eric-nirnava/p/mixin.html)

在读werkzeug和flask的源码中，经常能遇到类名中有mixin这个东西。这个东西的用法让我想到了java中的接口名有able的用法。今天我就来看了看这个mixin是什么东西。  
学习了python的都知道，python是支持多继承的，但是支不支持动态继承性质呢？在程序运行过程中，重定义类的继承，python是支持这种动态继承性质的。这也就是python中的mixin，在定义类过程中**改变类的继承顺序**，继承类。**当某个模块不能修改时，通过mixin方式可以动态添加该类的方法，动态改变类的原有继承体系**。

**Mixin**

Mixin编程是一种开发模式，是一种**将多个类中的功能单元的进行组合的利用的方式**，这听起来就像是有类的继承机制就可以实现，然而这与传统的类继承有所不同。通常**mixin并不作为任何类的基类**，也不关心与什么类一起使用，而是**在运行时动态的同其他零散的类一起组合使用**。  
使用mixin机制有如下好处：可以在不修改任何源代码的情况下，对已有类进行扩展；可以保证组件的划分；可以根据需要，使用已有的功能进行组合，来实现“新”类；很好的避免了类继承的局限性，因为新的业务需要可能就需要创建新的子类。  
在举例子之前，我们需要先熟悉几个关键字：

1. \_\_bases\_\_: 返回一个元组，该元组元素是类的基类，如：

class a:  
pass

print(a.\_\_bases\_\_)  
(<class 'object'>,)  
可见a类继承了object类

1. \_\_mro\_\_: This attribute is a tuple of classes that are considered when looking for base classes during method resolution.我的理解是 返回一个元组，该元组元素是该类的继承的类，当查询继承的时候会按从左到右的顺序。以刚刚的a类为例：

print(a.\_\_mro\_\_)  
(<class '**main**.a'>, <class 'object'>)

**例子**

在了解了以上的知识后，我来举个例子

import types

def mixin(pyClass, pyMixinClass, key=0):

if key:

pyClass.\_\_bases\_\_ = (pyMixinClass,) + pyClass.\_\_bases\_\_

elif pyMixinClass not in pyClass.\_\_bases\_\_:

pyClass.\_\_bases\_\_ += (pyMixinClass,)

else:

pass

class test1:

def test(self):

print('In the test1 class!')

class testMixin:

def test(self):

print('In the testMixin class!')

class test2(test1, testMixin):

def test(self):

print('In the test2 class!')

class test0(test1):

pass

if \_\_name\_\_ == '\_\_main\_\_':

print(test0.\_\_mro\_\_) #继承了test1，object

test\_0 = test0()

test\_0.test() #调用test1的方法

mixin(test0, testMixin, 1) #优先继承testMixin类

test\_\_0 = test0()

test\_\_0.test() #由于优先继承了testMixin类，所以调用testMixin类的方法

print(test0.\_\_mro\_\_)

print(test2.\_\_mro\_\_)

mixin(test2, testMixin)

print(test2.\_\_mro\_\_)

输出结果：

(<class '\_\_main\_\_.test0'>, <class '\_\_main\_\_.test1'>, <class 'object'>)

In the test1 class!

In the testMixin class!

(<class '\_\_main\_\_.test0'>, <class '\_\_main\_\_.testMixin'>, <class '\_\_main\_\_.test1'>, <class 'object'>)

(<class '\_\_main\_\_.test2'>, <class '\_\_main\_\_.test1'>, <class '\_\_main\_\_.testMixin'>, <class 'object'>)

(<class '\_\_main\_\_.test2'>, <class '\_\_main\_\_.test1'>, <class '\_\_main\_\_.testMixin'>, <class 'object'>)

注意本例的mixin方法，它的作用就是动态地给某个类(pyClass)通过继承的方式添加方法。  
要注意：pyClass.\_\_bases\_\_ = (pyMixinClass,) + pyClass.\_\_bases\_\_和pyClass.\_\_bases\_\_ += (pyMixinClass,) 二者的顺序是不一样的。 还是用例子说明：

a = (1,)

a = a + (2,)

print(a)

#输出：(1,2)

a = (3,) + a

print(a)

#输出：(3,1,2)

\_\_bases\_\_返回的元组的元素顺序不一样会导致继承的顺序不一样，类会优先继承元组左边的类