# 类中的实例方法，与类本身并没有绑定关系

# 类中的静态方法，与类也没有绑定关系

# 而类中的类方法，是和这个类存在绑定关系的

# 实例中的实例方法，与实例存在绑定关系

# 因为当通过一个实例去访问类中的某方法时，会形成绑定关系，将实例作为第一个参数self传入。

# 类方法与实例也存在绑定关系，所以实例可以直接调用类方法

# 静态方法与实例没有绑定关系

实例调用方法时会自动将自身传到函数中，类调用实例方法时要显示传入实例到函数中

MethodType 会在类内部创建一个链接，指向外部的的可调用对象，在创建实例的同时，这个绑定后的方法也会复制到实例中。MethodType 接受两个参数，第一个是被绑定的可调用对象，第二个是需要绑定到的对象。

MethodType只是一个链接指向外部可调用对象，而不是把外部可调用对象复制到类内部

**例：**

class Student(object):

pass

a=Student()#创建实例

def set\_score(self,score):

self.score=score

from types import MethodType

Student.set\_score = MethodType(set\_score, Student)

a.set\_score = MethodType(set\_score, a)

#只对a对象起作用，其他的对象无法使用set\_score,只有对类进这种操作其他对象才能使用相关的方法

a.set\_score(99)#调用方法

b=Student()

b.set\_score(60)

c=Student()

a.\_\_dict\_\_

b.\_\_dict\_\_

a.score

99

>>> b.score

60

>>> c.score

60

>>> Student.\_\_dict\_

Traceback (most recent call last):

File "<pyshell#164>", line 1, in <module>

Student.\_\_dict\_

AttributeError: type object 'Student' has no attribute '\_\_dict\_'

>>> Student.\_\_dict\_\_

mappingproxy({'\_\_module\_\_': '\_\_main\_\_', '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'Student' objects>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'Student' objects>, '\_\_doc\_\_': None, 'set\_score': <bound method set\_score of <class '\_\_main\_\_.Student'>>, 'score': 60})

>>> a.\_\_dict\_\_

{'set\_score': <bound method set\_score of <\_\_main\_\_.Student object at 0x02B46E30>>, 'score': 99}

>>> b.\_\_dict\_\_

{}

del a.set\_score

>>> a.\_\_dict\_\_

{'score': 99}

>>> b.set\_score(70)

>>> a.score

99

>>> del a.score

>>> a.score

70

**从上面的试验可以看出，当使用methodtype给类绑定方法时即**

**from types import MethodType**

**Student.set\_score = MethodType(set\_score, Student)**

**可以认为methodtype仅仅是在类与被绑定的方法（可认为：而且被绑定的方法复制到某个存储区但并不是复制到类的内部，若复制到内部则这种绑定方法跟直接赋值就没有任何区别了）之间建立一个指向方法的链接，通过这个链接，类可以访问到这个方法，当这个类被实例化以后，那么实例也可以利用链接（同一个链接）来访问这个方法了，当访问时不管是类还是实例，都是仅仅对这个链接到的函数进行调用访问，这个方法会在程序运行的全程有效。**

Student.\_\_dict\_\_

mappingproxy({'\_\_module\_\_': '\_\_main\_\_', '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'Student' objects>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'Student' objects>, '\_\_doc\_\_': None, **'set\_score': <bound method set\_score of <class '\_\_main\_\_.Student'>>, 'score': 60}**)

**通过methodtype给实例再次绑定方法时 ，外部函数会再次被复制到某个存储区（不同于上面类建立的链接函数的位置但不是实例的内部，仅仅可以这样认为而已），methodtype会将实例与这个存储区的函数连接起来，这样就完成了实例与函数的绑定，那么绑定的函数就属于实例了，当调用时实例会优先调用与实例绑定的函数， 虽然类中也有这个函数的链接，但会优先调用实例自身绑定的函数，这有点类似于前面的函数覆盖**

**a.set\_score = MethodType(set\_score, a)**

a.\_\_dict\_\_

**{'set\_score': <bound method set\_score of <\_\_main\_\_.Student object at 0x02B46E30>>, 'score': 99}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**实例的方法的绑定：**

**第一种：直接给类绑定方法，然后从类那里继承而来**

**第二种：from types import MethodType**

1. **set\_score = MethodType(set\_score, a)像这种利用MethodType来绑定方法，只能供实例使用**

**第三种：直接给实例的变量赋值一个函数（重点，易错点）**

**例子：**

**def set\_score(self,score):**

**self.score=score**

**a.set\_score = set\_score**

**a.set\_score(a,99)#调用方法**

这种情况下**，系统没有自动把参数传递给self，**必须自己显示传入，如上面的a对象，要自己单独传入，这种方法利用dir（a）查看会发现与第一种，第二种的方法绑定后，利用dir（a）查看的结果一样，也就是说都绑定成功了，都给对象绑定了相应的方法，就一点不一样就是传self与不传。