### Декораторы. Поддержка множества экземпляров

Декоратор это обёртка для класса или функции, которая позволяет обслуживать создание экземпляров классов или вызов функций и оборачивать их дополнительной логикой. Декоратор может быть определён как класс или как функция. И если определить декоратор класса как класс, то возникает одна проблема: создание каждого нового экземпляра затирает предыдущий. Для того, чтобы выяснить причину этого, я немного адаптировал пример из книжки Лутца "Изучаем Python". Вот что происходит, если декоратор определить как функцию:

>>> **def** decorator(cls):

**print** ('in func; cls='+str(cls))

**class** Wrapper:

**def** \_\_init\_\_(self, \*args):

**print** ('in class init')

self.wrapped=cls(\*args)

**print** ('self='+str(self)+' type(self)='+str(type(self))+'**\n**self.wrapped='+str(self.wrapped)+' type(self.wrapped)='+str(type(self.wrapped))+'**\n**cls='+str(cls)+' type(cls)='+str(type(cls)))

**def** \_\_getattr\_\_(self,name):

**return** getattr(self.wrapped,name)

**print** (str(Wrapper)+'**\t**'+str(type(Wrapper)))

**return** Wrapper

>>> @decorator

**class** C:

**def** \_\_init\_\_(self, \*args):

self.attr=args

**in** func; cls=<class '\_\_main\_\_.C'>

<class '\_\_main\_\_.Wrapper'> <class 'type'>

>>> C, type(C)

(<class '\_\_main\_\_.Wrapper'>, <class 'type'>)

>>> x=C('asdf')

**in** **class** init

self=<\_\_main\_\_.Wrapper object at 0x010C9B90> type(self)=<class '\_\_main\_\_.Wrapper'>

self.wrapped=<\_\_main\_\_.C object at 0x010C9CD0> type(self.wrapped)=<class '\_\_main\_\_.C'>

cls=<class '\_\_main\_\_.C'> type(cls)=<class 'type'>

>>> y=C('asdf')

**in** **class** init

self=<\_\_main\_\_.Wrapper object at 0x010C7DB0> type(self)=<class '\_\_main\_\_.Wrapper'>

self.wrapped=<\_\_main\_\_.C object at 0x010C93D0> type(self.wrapped)=<class '\_\_main\_\_.C'>

cls=<class '\_\_main\_\_.C'> type(cls)=<class 'type'>

>>> x,type(x)

(<\_\_main\_\_.Wrapper object at 0x010C9B90>, <class '\_\_main\_\_.Wrapper'>)

>>> y,type(y)

(<\_\_main\_\_.Wrapper object at 0x010C7DB0>, <class '\_\_main\_\_.Wrapper'>)

Как мы видим, на этапе использования декоратора создается только объект класса, но не экземпляр класса Wrapper, оригинальный же класс сохраняется в локальной области видимости функции в процессе компиляции (не вызова) объекта класса. А вот уже при создании экзепляров декорированного класса С, по сути экземпляров класса Wrapper, создаются разные экземпляры и объектов С и оборачивающего его Wrapper, как можно видеть по выводимым адресам. Что же происходит, если определить декоратор как класс?

>>> **class** Decorator:

**def** \_\_init\_\_(self, C):

**print** ('Decorator init')

self.C=C

**print** ('self='+str(self)+' type(self)='+str(type(self))+'**\n**self.C='+str(self.C)+' type(self.C)='+str(type(self.C))+'**\n**C='+str(C)+' type(C)='+str(type(C)))

**def** \_\_call\_\_(self,\*args):

**print** ('Decorator call')

self.wrapped=self.C(\*args)

**print** ('self='+str(self)+' type(self)='+str(type(self))+'**\n**self.wrapped='+str(self.wrapped)+' type(self.wrapped)='+str(type(self.wrapped))+'**\n**cls='+str(self.C)+' type(self.C)='+str(type(self.C)))

**return** self

**def** \_\_getattr\_\_(self, attrname):

**return** getattr(self.wrapped,attrname)

>>> @Decorator

**class** C: **pass**

Decorator init

self=<\_\_main\_\_.Decorator object at 0x010A9FB0> type(self)=<class '\_\_main\_\_.Decorator'>

self.C=<class '\_\_main\_\_.C'> type(self.C)=<class 'type'>

C=<class '\_\_main\_\_.C'> type(C)=<class 'type'>

>>> x=C()

Decorator call

self=<\_\_main\_\_.Decorator object at 0x010A9FB0> type(self)=<class '\_\_main\_\_.Decorator'>

self.wrapped=<\_\_main\_\_.C object at 0x00C70830> type(self.wrapped)=<class '\_\_main\_\_.C'>

cls=<class '\_\_main\_\_.C'> type(self.C)=<class 'type'>

>>> y=C()

Decorator call

self=<\_\_main\_\_.Decorator object at 0x010A9FB0> type(self)=<class '\_\_main\_\_.Decorator'>

self.wrapped=<\_\_main\_\_.C object at 0x010A9FF0> type(self.wrapped)=<class '\_\_main\_\_.C'>

cls=<class '\_\_main\_\_.C'> type(self.C)=<class 'type'>

>>> x,type(x),y,type(y)

(<\_\_main\_\_.Decorator object at 0x010A9FB0>, <class '\_\_main\_\_.Decorator'>, <\_\_main\_\_.Decorator object at 0x010A9FB0>, <class '\_\_main\_\_.Decorator'>)

Тут мы видим, что на этапе декорирования создаётся [b]экземпляр[/b] класса Decorator! И при создании экземпляров декорируемого класса будет вызываться метод \_\_call\_\_, которому каждый раз будет передаваться данный экземпляр класса Decorator и будет затираться его атрибут wrapped. Но важно отметить, что если мы применим этот декоратор к другому классу это не затрет наш предыдущий декоратор. Да это и понятно, так как в данном случае будет просто создан ещё один экземпляр класса Decorator:

>>> @Decorator

**class** C: **pass**

Decorator init

self=<\_\_main\_\_.Decorator object at 0x010A9FB0> type(self)=<class '\_\_main\_\_.Decorator'>

self.C=<class '\_\_main\_\_.C'> type(self.C)=<class 'type'>

C=<class '\_\_main\_\_.C'> type(C)=<class 'type'>

>>> x=C()

Decorator call

self=<\_\_main\_\_.Decorator object at 0x010A9FB0> type(self)=<class '\_\_main\_\_.Decorator'>

self.wrapped=<\_\_main\_\_.C object at 0x00C70830> type(self.wrapped)=<class '\_\_main\_\_.C'>

cls=<class '\_\_main\_\_.C'> type(self.C)=<class 'type'>

>>> @Decorator

**class** D: **pass**

Decorator init

self=<\_\_main\_\_.Decorator object at 0x010B1030> type(self)=<class '\_\_main\_\_.Decorator'>

self.C=<class '\_\_main\_\_.D'> type(self.C)=<class 'type'>

cls=<class '\_\_main\_\_.D'> type(C)=<class 'type'>

>>> y=D()

Decorator call

self=<\_\_main\_\_.Decorator object at 0x010B1030> type(self)=<class '\_\_main\_\_.Decorator'>

self.wrapped=<\_\_main\_\_.D object at 0x00C70830> type(self.wrapped)=<class '\_\_main\_\_.D'>

cls=<class '\_\_main\_\_.D'> type(self.C)=<class 'type'>

>>> y,type(y)

(<\_\_main\_\_.Decorator object at 0x010B1030>, <class '\_\_main\_\_.Decorator'>)