**一个例子走近 Python 的 Mixin 类：利用 Python 多继承的魔力**

https://blog.csdn.net/u012814856/article/details/81355935

**一、引言**

这篇博客，是我在看了《Python GUI Programming with Tkinter》第 76 页的小节 Exploiting the power of multiple inheritance 之后，压抑不住自己的激动兴奋之情，认真整理书写分享出来的。所用到的也是这一节作者写出来的代码，希望能给广大网友跟我一样的[恍然大悟](https://www.baidu.com/s?wd=%E6%81%8D%E7%84%B6%E5%A4%A7%E6%82%9F&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)的痛快的感觉。

[言归正传](https://www.baidu.com/s?wd=%E8%A8%80%E5%BD%92%E6%AD%A3%E4%BC%A0&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)，Python 无疑是支持多继承的。我们可以利用 Python 的这种特性，实现一种叫做 Mixin 的类：

Mixin classes contain only a specific set of functionalities that we want to be able to mix in with other classes to compose a new class.(引用于 《Python GUI Programming with Tkinter》P76)

作者的这句话是什么意思呢？我大概翻译一下：

Mixin 类只包含了一组特定的函数集合，而我们将会将其与其他类进行混合，从而生成一个适用于实际需要的新类

看文本介绍太过于苍白，我们来看一个例子是最好的。

**二、看一个 Mixin 类的实例**

这里，我直接先上代码，有兴趣的同学，可以暂停到这里，看看这段代码中的 [subclass](https://www.baidu.com/s?wd=subclass&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd).display() 这行代码，究竟是怎么执行的：

class Displayer():

def display(self, message):

print(message)

class LoggerMixin():

def log(self, message, filename='logfile.txt'):

with open(filename, 'a') as fh:

fh.write(message)

def display(self, message):

super().display(message)

self.log(message)

class MySubClass(LoggerMixin, Displayer):

def log(self, message):

super().log(message, filename='subclasslog.txt')

subclass = MySubClass()

subclass.display("This string will be shown and logged in subclasslog.txt")

代码不多，也就拢共 22 行，22 行的代码里面，定义了 3 个类。其中 MySubClass 多继承了 LoggerMixin 类和 Displayer 类。看似并没有什么异常的代码里面，当你尝试去仔细推敲 subclass.display() 的调用逻辑之后，就变得异常的复杂。

**3 个类的作用**

首先，我们先来依次介绍下这 3 个类的作用：

**1.** Displayer 类   
只有一个函数，display 函数，其简单打印输出传进来的 message 字符串

**2.** LoggerMixin 类   
这个类就是我们引言中提到的 Mixin 类，也就是说，这个类的方法，就是我们想要与 Displayer 类进行混合，然后生成我们想要的新类 MySubClass 的。

这个类有两个方法：

log 方法使用了 with open 语法操作了一个日志文件，日志文件缺省命名为 logfile.txt，然后将得到的 message 参数追加到这个文件中去。

display 方法调用了 super() 的 display 方法，然后调用了 self.log 方法。

这里问题就来了：**我们的 Mixin 类根本就没有继承任何除了 Object 类之外的父类呀**，也就是说，Object 类没有 display 方法，那么 LoggerMixin 类究竟是调用了谁的 display 方法呢？

这个问题相当诡异，我们之后再说。

**3.** MySubClass 类   
这个类就是我们多继承而来的新类，只有一个方法，log 方法简单调用了父类的 log 方法。

**提出问题**

按照作者的原话来说：

Technically, LoggerMixin inherits from Python’s built-in object class, which has no display() method. How, then, can we call super().display() here?

这也是我在上一个标题中提出来的问题，我们的 LoggerMixin 类是怎么调用的 super().display() 方法的呢？

**问题解释**

作者很快就给出了答案：

In a multiple inheritance situation, super() does something a little more complex than just standing in for the superclass. It look up the chain of inheritance using something called the **Method Resolution Order** and determines the nearest class that defines the method we’re calling.

简单翻译下：

在多继承的环境下，super() 有相对来说更加复杂的含义。它会查看你的继承链，使用一种叫做 **Methods Resolution Order（方法解析顺序）** 的方式，来决定调用最近的继承父类的方法。

也就是说，我们的 MySubClass.display() 调用，触发了是这么一系列的行为：

**1.** MySubClass.display() is resolved to LoggerMixin.display().   
MySubClass.display() 方法被解析为 LoggerMixin.display() 方法的调用。这应该还是比较好理解的。因为对于 MySubClass 类来说，在继承链上的两个父类，LoggerMixin 和 Displayer 来说，LoggerMixin 是最近的，因此调用它的 display() 方法。

**2.** LoggerMixin.display() calls super().display(), which is resolved to Displayer.display().   
LoggerMixin.display() 方法调用了 super().display()，这一行代码按照我们刚才的解释，查看 MySubClass 的继承链，是应该调用 Displayer 类的 display() 方法的。这一步是相对来说比较难以理解的。

让我们这么来理解它，当 LoggerMixin.display() 中调用了 super().display() 的时候，它会尝试去寻找属于当前类的继承链。而这个当前类是什么类呢？不是 LoggerMixin 类，而是 MySubClass 类。MySubClass 类的继承连是 LoggerMixin，然后 Displayer。所以，我们就找到了 Displayer 的 display() 方法。

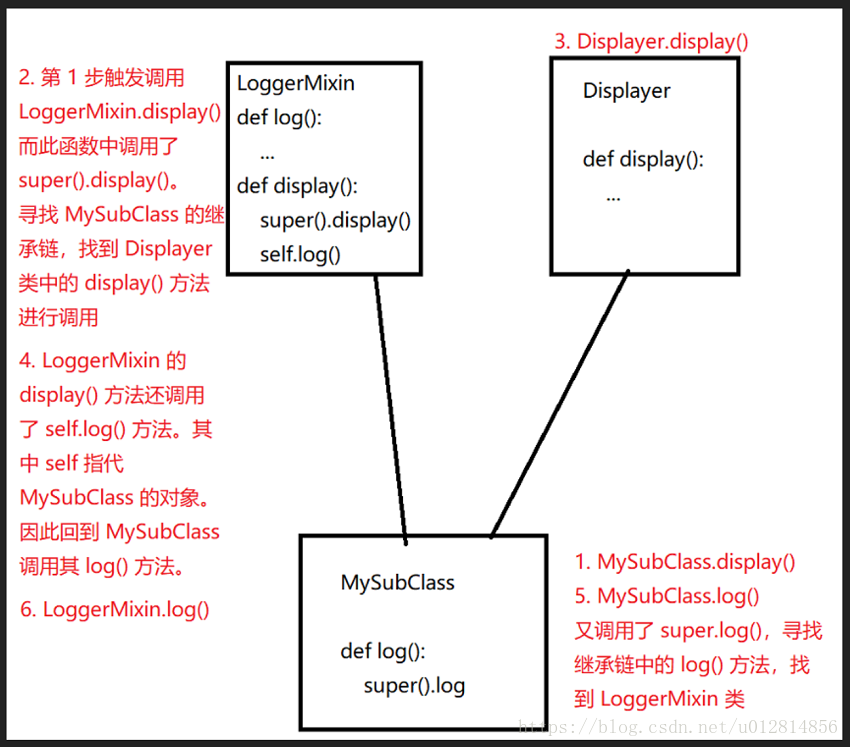
**3.** It alse calls self.log(). Since self, in this case, is a MySubClass instance, it resolves to MySubClass.log(). MySubClass.log() calls super().log(), which is resolved back to LoggerMixin.log().   
别忘了，我们的 LoggerMixin 类还调用了 self.log() 方法。这个看似好像要直接调用 LoggerMixin 的 log 方法，其实不然。

LoggerMixin 的 display() 方法在当前语境中的 self，其实是 MySubClass 类的对象，因此对于 MySubClass 类的对象，想要调用 log 方法，是直接调用自己类中的 log 方法，也就是 MySubClass.log() 方法，而不是 LoggerMixin.log() 方法的。

而又因为 MySubClass.log() 方法调用了 super().log() 方法，这才根据继承链寻找最近的父类，才找到了 LoggerMixin 类中的 log() 方法进行调用。

**画图总结**

为了方便大家理解，我画了一个简略的图方便理解：   
一句简单的 subclass.display() 的背后，究竟发生了什么？



**三、总结**

看上面的解释或许还是觉得[云里雾里](https://www.baidu.com/s?wd=%E4%BA%91%E9%87%8C%E9%9B%BE%E9%87%8C&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)，其实作者给出了更加简单的理解方式：

If this seems confusing, just remember that self.method() will look for method() in the current class first, then follow the list of inherited classes from left to right until the method is found. The super().method() will do the same, except that it skips the current class.

简单翻译下：

如果上述的解释太过于难以理解，我们可以简单记住，self.method() 将会先在当前类中查看 method() 方法，如果没有，就在继承链中进行查找，查找顺序就是你继承的顺序从左到右，直到 method() 方法被找到。super().method() 与 self.method() 是差不多的，只是 super().method() 需要跳过当前类而已。

这也就是我们的 Mixin 类牵扯出来的多继承函数调用问题的展现。

话说回来，为什么要叫做 Mixin 类呢？

Note that LoggerMixin is not usable on its own: it only works when combined with a class that has a display() method. This is why it’s a mixin class because it’s meant to be mixed in to enhance other classes.

也就是说，我们的 LoggerMixin 类是无法单独使用的，它必须要和一个拥有 display() 函数定义的类一起混合使用。这也就是为什么它被称作是 Mixin 类的原因，它总是需要与其他类混合来加强其他类。

至于这种编写代码模式的作用，还是有很大作用的。可以大大简化和方便我们的代码的开发过程。

阐述到这里，是不是也跟我一样有了恍然大悟的快感了呢？