**原问题**

Python中既然可以直接通过父类名调用父类方法为什么还会存在super函数？   
比如   
class Child(Parent):   
    def \_\_init**(self):   
    Parent.\_\_init**(self)   
这种方式与super(Child, self).**init**()有区别么？

**回答**   
针对你的问题，答案是可以，并没有区别。但是这题下的回答我感觉都不够好。

要谈论 super，首先我们应该无视 "super" 这个名字带给我们的干扰。

**不要一说到 super 就想到父类！super 指的是 MRO 中的下一个类！**   
**不要一说到 super 就想到父类！super 指的是 MRO 中的下一个类！**   
**不要一说到 super 就想到父类！super 指的是 MRO 中的下一个类！**

一说到 super 就想到父类这是初学者很容易犯的一个错误，也是我当年犯的错误。 忘记了这件事之后，再去看这篇文章：[Python’s super() considered super!](https://rhettinger.wordpress.com/2011/05/26/super-considered-super/) 这是 Raymond Hettinger 写的一篇文章，也是全世界公认的对 super 讲解最透彻的一篇文章，凡是讨论 super 都一定会提到它（当然还有一篇 Python's Super Considered Harmful）。

如果不想看长篇大论就去看[这个答案](http://stackoverflow.com/questions/15896265/python-super-inheritance-and-arguments-needed/15896594#15896594)，super 其实干的是这件事：

def super(cls, inst):

mro = inst.\_\_class\_\_.mro()

return mro[mro.index(cls) + 1]

两个参数 cls 和 inst 分别做了两件事：   
1. inst 负责生成 MRO 的 list   
2. 通过 cls 定位当前 MRO 中的 index, 并返回 mro[index + 1]   
这两件事才是 super 的实质，一定要记住！   
MRO 全称 Method Resolution Order，它代表了类继承的顺序。后面详细说。

举个例子

class Root(object):

def \_\_init\_\_(self):

print("this is Root")

class B(Root):

def \_\_init\_\_(self):

print("enter B")

# print(self) # this will print <\_\_main\_\_.D object at 0x...>

super(B, self).\_\_init\_\_()

print("leave B")

class C(Root):

def \_\_init\_\_(self):

print("enter C")

super(C, self).\_\_init\_\_()

print("leave C")

class D(B, C):

pass

d = D()

print(d.\_\_class\_\_.\_\_mro\_\_)

输出

enter B

enter C

this is Root

leave C

leave B

(<class '\_\_main\_\_.D'>, <class '\_\_main\_\_.B'>, <class '\_\_main\_\_.C'>, <class '\_\_main\_\_.Root'>, <type 'object'>)

知道了 super 和父类其实没有实质关联之后，我们就不难理解为什么 enter B 下一句是 enter C 而不是 this is Root（如果认为 super 代表“调用父类的方法”，会想当然的认为下一句应该是this is Root）。流程如下，在 B 的 \_\_init\_\_ 函数中：

super(B, self).\_\_init\_\_()

首先，我们获取 self.\_\_class\_\_.\_\_mro\_\_，注意这里的 self 是 D 的 instance 而不是 B 的

(<class '\_\_main\_\_.D'>, <class '\_\_main\_\_.B'>, <class '\_\_main\_\_.C'>, <class '\_\_main\_\_.Root'>, <type 'object'>)

然后，通过 B 来定位 MRO 中的 index，并找到下一个。显然 B 的下一个是 C。于是，我们调用 C 的 \_\_init\_\_，打出 enter C。

顺便说一句为什么 B 的 \_\_init\_\_ 会被调用：因为 D 没有定义 \_\_init\_\_，所以会在 MRO 中找下一个类，去查看它有没有定义 \_\_init\_\_，也就是去调用 B 的 \_\_init\_\_。

其实这一切逻辑还是很清晰的，关键是理解 super 到底做了什么。

于是，MRO 中类的顺序到底是怎么排的呢？Python’s super() considered super!中已经有很好的解释，我翻译一下：   
**在 MRO 中，基类永远出现在派生类后面，如果有多个基类，基类的相对顺序保持不变。**   
关于 MRO 的官方文档参见：[The Python 2.3 Method Resolution Order](https://www.python.org/download/releases/2.3/mro/)，有一些关于 MRO 顺序的理论上的解释。

最后的最后，提醒大家. 什么 super 啊，MRO 啊，都是针对 new-style class。如果不是 new-style class，就老老实实用父类的类名去调用函数吧。