C++中的THIS指针和PYTHON中的SELF

在C++中，对象可以通过this指针来访问自己的地址，this指针是所有成员函数的隐含参数。当通过对象调用一个成员函数时，实际上是将这个对象的地址隐式地赋给了这个成员函数的this参数。

在python中，类中的方法的第一个参数永远都是self，这个参数表示调用这个方法的对象本身。通过对象调用方法时，隐式地将这个对象传给了调用方法的第一个参数。

```
#include <iostream>
#include <string>
using namespace std;
class Test{
    protected:
        string name;
    public:
        Test(string name){ this->name = name; };
        void ShowThis(){ cout << "this指针的地址为：" << this <<
endl; };
        void ShowName(){ cout << this->name << endl; };
        Test& ReturnThis(){ return *this; };
};

int main(){
    Test test1("test1");
    Test test2("test2");
    cout << "test1所在的地址为：" << &test1 << endl;
    test1.ShowThis();
    cout << "test2所在的地址为：" << &test2 << endl;
    test2.ShowThis();
    Test& test3 = test1.ReturnThis();
    cout << "test3所在的地址为：" << &test3 << endl;
    test3.ShowThis();
    return 0;
}
```

上面这段程序的输出为：

```
test1所在的地址为：0x7ffee4f653c0
this指针的地址为：0x7ffee4f653c0
test2所在的地址为：0x7ffee4f65380
this指针的地址为：0x7ffee4f65380
test3所在的地址为：0x7ffee4f653c0
this指针的地址为：0x7ffee4f653c0
```

this指针的地址实际上就是这个对象的内存地址。在class定义时要用到对象本身时，此时还不知道变量名，this指针就可以指代类的对象自身。实际上python中类的self参数和this指针的作用基本完全一样：

```
class Test:
    def ShowSelf(self):
        print('self的值为：', self)

test1 = Test()
test2 = Test()

print('test1的地址为：', test1)
test1.ShowSelf()
print('test2的地址为：', test2)
test2.ShowSelf()
```

上面这段程序的输出为：

```
test1的地址为：  <__main__.Test object at 0x7f92447030a0>
self的值为：  <__main__.Test object at 0x7f92447030a0>
test2的地址为：  <__main__.Test object at 0x7f92447035e0>
self的值为：  <__main__.Test object at 0x7f92447035e0>
```

其实调用 `test1.ShowSelf()` 和 `Test.ShowSelf(test1)` 是没有区别的，在类中定义的方法默认为绑定方法，当通过对象调用时自动将该对象传给这个方法。

this指针和self都可以作为成员函数（类方法）的返回值，来返回当前调用这个函数的对象，比如在python中，一个上下文管理器的 `__enter__` 方法的返回值就经常是self..

```
class ContextManager:
    def __init__(self):
        print('__init__方法在运行..')
        pass
    def __enter__(self):
        print('进入了上下文管理器，并把__enter__方法的返回值给了as后的
对象')
        return self
    def __exit__(self, exc_type, exc_value, traceback):
        print('离开上下文管理器')
        pass
    def FunctionA(self):
        print('这个函数可以做些事情..')


with ContextManager() as manager_a:
    manager_a.FunctionA()
```

输出为：

```
__init__方法在运行..
进入了上下文管理器，并把__enter__方法的返回值给了as后的对象
这个函数可以做些事情..
离开上下文管理器
```