C 对象布局及多态之虚成员函数调用 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/133/2021_2022_C___E5_AF_B9_E8_B1_A1_E5_c97_133690.htm 在构造函数中调用虚成员函数，虽然这是个不很常用的技术，但研究一下可以加深对虚函数机制及对象构造过程的理解。这个问题也和一般直观上的认识有所差异。先看看下面的两个类定义。struct C180{ C180() { foo(). this->foo(). } virtual foo() { cout }}.struct C190 : public C180{ C190() {} virtual foo() { cout }}. 父类中有一个虚函数，并且父类在它的构造函数中调用了这个虚函数，调用时它采用了两种方法一种是直接调用，一种是通过this指针调用。同时子类又重写了这个虚函数。 我们可以来预测一下如果构造一个C190的对象会发生什么情况。 我们知道，在构造一个对象时，首先会按对象的大小得到一块内存(在heap上或在stack上)，然后会把指向这块内存的指针做为this指针来调用类的构造函数，对这块内存进行初始化。如果对象有父类就会先调用父类的构造函数(并依次递归)，如果有多个父类(多重继承)会依次对父类的构造函数进行调用，并会适当的调整this指针的位置。在调用完所有的父类的构造函数后，再执行自己的代码。 照上面的分析构造C190时也会调用C180的构造函数，这时在C180构造函数中的第一个foo调用为静态绑定，会调用到C180::foo()函数。第二个foo调用是通过指针调用的，这时多态行为会发生，应该调用的是C190::foo()函数。执行如下代码：C190 obj.obj.foo(). 结果为： 和我们的分析大相径庭。前2行是构造C190时的输出，后1行是我们用静态绑定方式调用的C190::foo()函数。第2行的输出说明多态行为并

没有象预期的那样发生。而且比较输出的最后一列，发现在调用C180的构造函数时对象对应的虚表和构造后对象对应的虚表不是同一个。其实这正是奥秘的所在。 为此我查了一下C 标准规范。在12.7.3条中有明确的规定。这是一种特例，在这种情况下，即在构造子类时调用父类的构造函数，而父类的构造函数中又调用了虚成员函数，这个虚成员函数即使被子类重写，也不允许发生多态的行为。即，这时必须要调用父类的虚函数，而不子类重写后的虚函数。 我想这样做的原因是因为在调用父类的构造函数时，对象中属于子类部分的成员变量是肯定还没有初始化的，因为子类构造函数中的代码还没有被执行。如果这时允许多态的行为，即通过父类的构造函数调用到了子类的虚函数，而这个虚函数要访问属于子类的数据成员时就有可能出错。 