C 对象布局及多态实现之动态和强制转换 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/467/2021_2022_C___E5_AF_B9_E8_B1_A1_E5_c67_467256.htm 从这部分开始我们除了利用内存的信息打印来进行探索外，更多的会通过跟踪和观察编译器产生的汇编代码来理解编译器对这些语言特性的实现方式。汇编方面知识的讨论超出了本文的范围，我只对和我们讨论相关的汇编代码进行解析。理解本文要讨论的知识并不需要有很完整的汇编知识，但必须了解起码的概念。 下面我们看看引入虚继承后的影响。为了有所对比我们首先看看普通成员函数的调用情况。 执行如下代码，它包括了对象的普通成员函数调用，类的静态成员函数调用、通过指针调用普通成员函数： C010 obj. PRINT_OBJ_ADR(obj) obj.foo(). C012::sfoo(). C010 * pt = &amp.obj. pt-> foo(). 结果如下： obj ’s address is : 0012F843 这是obj对象的内存地址。 首先我们看看对象的普通成员函数调用，obj.foo().，对应的汇编代码为： 00422E09 lea ecx,[ebp FFFFF967h] 00422E0F call 0041E289 第1行把对象的地址存入ecx寄存器，执行完这行指令后，我们要以看到ecx中的值为0x0012F843，就是前面打印出的值。如果函数需要传递参数，我们还会在前面看到一些push指令。在第2行我们可以看到call的是一个直接的地址，这也就是静态绑定。即函数的调用地址在编译时已经被编译器决议。 跟踪进去我们要以看到是一条跳转指令，继续执行可以看到真正的函数代码部分，如下(注：为了讨论方便我在第行前面加了一个行号)： 01 00425FE0 push ebp 02 00425FE1 mov ebp,esp 03 00425FE3 sub esp,0CCh 04 00425FE9 push ebx 05 00425FEA push

esi 06 00425FEB push edi 07 00425FEC push ecx 08 00425FED lea edi,[ebp FFFFFF34h] 09 00425FF3 mov ecx,33h 10 00425FF8 mov eax,0CCCCCCCCh 11 00425FFD rep stos dword ptr [edi] 12 00425FFF pop ecx 13 00426000 mov dword ptr [ebp-8],ecx 14 00426003 mov eax,dword ptr [ebp-8] 15 00426006 mov byte ptr [eax],2 16 00426009 pop edi 17 0042600A pop esi 18 0042600B pop ebx 19 0042600C mov esp,ebp 20 0042600E pop ebp 21 0042600F ret 我们看看第7行，把ecx寄存器入栈，后面4行初始化了函数的堆栈中的保存局部变量的部分。第12行弹出ecx值，到这里时ecx的值保持为在函数调用前存入的对象内存地址，第13行就是保存this指针的值，作为一个局部变量。这样我们就知道了VC7.1不是象传递普通函数那样通过压栈来传递this 指针，而是通过ecx寄存器来传递。第14、15行利用这个this指针给对象的成员变量进行了赋值 再看看静态成员函数调用的汇编代码： 00422E14 call 0041DD84 非常直接，因为它不需要处理this指针，跟踪到函数的汇编代码，可以看到同样不需要处理this指针。具体的代码这里就不列出来了。 再看看通过指针调用普通成员函数pt-> foo()., 产生的汇编代码如下： 00422E25 mov ecx,dword ptr [ebp FFFFF958h] 00422E2B call 0041E289 和通过对象调用普通成员函数的代码差不多。不过存对象地址到ecx寄存器地，是通过解引用pt指针来找到对象地址的。