# 深入C++探索--- 四 从函数到类

## 零、声明

1. 这个系列《深入C++探索》，以汇编和内存角度对C++的一些基本的运行机制进行讨论；
2. 这个系列中出现的汇编代码，都是来自于VS2010的Debug环境下生成的；
3. 如果觉得文章对你有帮助而需要转载，也请阁下能够注明出处；
4. 如果觉得博文对问题的讨论有误，也可以给博主留言。

## 类

### 1.1.C++代码

代码：

class CA

{

public:

int fun(int x,int y){return 1;}

virtual int fun\_v1(int x,int y){return x+y;}

virtual int fun\_v2(int x,int y){return x+y;}

};

class CB : public CA

{

virtual int fun\_v1(int x,int y){return 2;}

virtual int fun\_v2(int x,int y){return 3;}

};

//使用代码

CA \* p = new CB();

p->fun(1,2);

p->fun\_v1(1,2);

### 1.2.汇编代码

我们首先来看调用代码的汇编：

CA \* p = new CB();

003B172E push 4 //这里push 4，在之前提到函数调用参数时提到

//其实这个4是一个参数，需要传入operator new(size\_t size)

//CB的大小就是4，其实就是sizeof(CB)

003B1730 call operator new (3B11D6h) //调用operator new(size\_t size);

//分配4字节的内存

003B1735 add esp,4 //平衡当前函数的栈指针esp;

// 这里的4和push 4意义不同

//这里的4表示是要在栈上分配4个字节来存放new出来的内存地 也即是指针

003B1738 mov dword ptr [ebp-0D4h],eax //将分配的内存地址存放在栈上分配的4字节中

003B173E cmp dword ptr [ebp-0D4h],0 //比较指针p是否为0

003B1745 je wmain+4Ah (3B175Ah) //如果指针p为0就跳转，不执行构造函数

003B1747 mov ecx,dword ptr [ebp-0D4h] //将指针p赋给ecx

//对于每一个类的非静态函数都会传入this指针，就是这么传入的

003B174D call CB::CB (3B1195h) //调用CB的构造函数，会重新分配适合

003B1752 mov dword ptr [ebp-0DCh],eax //将返回结果存放在 ebp-0DCh位置

003B1758 jmp wmain+54h (3B1764h) //跳转到 0B51764h处代码

003B175A mov dword ptr [ebp-0DCh],0 //将 ebp-0DCh位置处赋0 这个语句不会执行到

003B1764 mov eax,dword ptr [ebp-0DCh] //将新的内存地址分配给eax

003B176A mov dword ptr [p],eax //也即是将构造函数返回的地址重新赋给指针p，这一步操作的详细，将在之后的构造函数中详细展开

构造完成，来看类对象的普通函数调用

p->fun(1,2);

003B176D push 2 //压入参数

003B176F push 1 //压入参数

003B1771 mov ecx,dword ptr [p]//传入this指针p

003B1774 call CA::fun (3B11B8h) //调用fun函数

来看类对象的虚函数调用

p->fun\_v1(1,2);

003B1779 mov esi,esp //

003B177B push 2 //压入参数

003B177D push 1 //压入参数

003B177F mov eax,dword ptr [p] //将p所指向的值付给eax

003B1782 mov edx,dword ptr [eax] //将地址等于eax处的值付给edx，也就是p指向内存地址存放的虚函数表的指针

003B1784 mov ecx,dword ptr [p] //传入this指针p

003B1787 mov eax,dword ptr [edx] // 其实就是将虚函数表中第一个函数fun\_v1的地址

//如果调用的是p->fun\_v2(1,2),那么这里应该是 mov eax,dword ptr [edx+4]，移动到第二个函数槽

003B1789 call eax //调用fun\_v1

003B178B cmp esi,esp //校验esp

这里就是简单的类对象的生成，以及函数的调用和虚函数的调用

我们看一下内存

p所指向的内存地址

0x00976478 40 57 3b 00 cd cd cd cd fd fd fd fd ab ab ab ab ab ab ab ab ee fe ee fe 00 00 00 00 00 00 00 00 f0 e5 d9

看一下第一个四个字节 ，其实这个就是一个地址（虚函数表的地址）

我们看一下虚函数表中的存了什么， 这里的第一个四个字节 ，也是就是一个地址，即第一个虚函数的地址

0x003B5740 0a 10 3b 00 35 12 3b 00 90 65 3b 00 18 11 3b 00 30 12 3b 00 a8 65

我们看一下汇编语句，在往里面跳转

01391789 call eax //调用fun\_v1 再往里面跳转003B100A jmp CB::fun\_v1 (3B1870h)

函数的地址与虚函数表中存放的地址吻合。

以上代码在VS2010 Debug编译环境下生成

2.函数被调用者修改堆栈

3.函数名自动加前导的下划线，后面紧跟一个@符号，其后紧跟着参数的尺寸，这一个命名的也就限死了不能用可变数量的参数

相对于的\_\_cdcel较为简单

fun1\_stdcall(1,2);

int \_\_stdcall fun1\_stdcall(int x,int y)

{

return x+y;

}

详细代码：

调用：

fun1\_stdcall(1,2);

001C168E push 2 //压入参数，从右往左

001C1690 push 1 //压入参数，从右往左

001C1692 call fun1\_stdcall (1C108Ch)

实现：

int \_\_stdcall fun1\_stdcall(int x,int y)

{

...

return x+y;

001515CE mov eax,dword ptr [x]

001515D1 add eax,dword ptr [y]

}

...

001515DA ret 8 //这个就是stdcall需要函数本身维持栈的平衡

//因为压入了两个int所以这里需要移动8来平衡。

这里前面有介绍过，最有一点留意的是ret 8。在win32应用程序里,宏APIENTRY，WINAPI，都表示\_stdcall。

## 三、总结

其他的调用方式这里不再做展开讨论了。