# C++类函数

## 引言

类函数的介绍基于《C++之路\_基础\_函数》，对C++的函数再进一步延伸。這里一个函数有一个归属的问题（静态函数除外）。

此文将介绍类函数最主要的两点：普通函数与虚函数、C++多态实现的基础（虚函数表），所以我们这里不考虑C++的多继承和虚继承。

## 函数调用

之前提到的类函数区别于普通函数就是：类函数有一个归属，C++是通过this指针来实现的。

C++将this指针通过函数的参数传入类函数中。

为了介绍方便，我们这里使用两个简单的类CA和CB。CB是CA的一个子类。

**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);**

### 2.1. 类对象创建

对于上述的代码和类，我们首先来分析第一句，类对象的创建。这里用到了操作符operator new和类的构造函数。虽然是简简单单的一句代码，其实是C++中较为复杂的部分。

按照以往的方式，我们按照汇编语句展开，用注释方式来解释：

**CA \* p = new CB();**

**003B172E  push 4 //这里push 4，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**

### 2.2. 类普通函数调用

类对象构建完成之后，开始调用类普通的成员函数：

**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**

### 2.3. 虚函数调用

虚函数的调用：

**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

### 2.4. 内存分析

函数的调用的展开介绍完了，我们看一下的在这个过程的内存：

首先类对象指针p所指向的内存地址是0x00976478

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

看一下前四个字节 ，其实这个就是一个地址，就是虚函数表的地址。

我们看一下虚函数表中存了什么，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

这里的前四个字节0a 10 3b 00转换成地址就是0x003B100A。我们看一下“2.3 虚函数调用”中0x003B1789处的汇编语句（**003B1789 call eax**），再往里面跳转便是代码段（**003B100A jmp CB::fun\_v1 (3B1870h)**），此代码段的地址与虚函数表中存放的第一个虚函数地址吻合。

## 函数详解

C++中的构造函数，可以说是C++类中最为复杂的一部分。接下来，将详细介绍构造函数。我们对第二章中的两个函数修改。

我们把构造函数与虚函数放在一起解释，是因为有关于虚函数表。

**class CA**

**{**

**protected:**

**int m\_x0;**

**public:**

**CA(){m\_x0 = 0;}**

**virtual ~CA(){}**

**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 1;}**

**};**

**class CB : public CA**

**{**

**public:**

**CB(){m\_x = 1;}**

**~CB(){}**

**int m\_x;**

**virtual int fun\_v1(int x,int y)**

**{**

**int z = m\_x0 + x;**

**z += m\_x;**

**return z;**

**}**

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

**};**

对两个类添加成员变量，并加上构造函数和析构函数。

### 3.1. 构造函数

我们首先来看一下构造函数，构造函数应该是类中最复杂的函数了，他做了当前类以及父类的初始化工作；

我们来看一下它的汇编代码（去除头和尾部）

**CB(){m\_x = 1;}**

**...**

**013618EF  pop ecx   //获取this指针，这里有些不符，如果是按照之前调用之前，this指针存放在了ecx，这里应该是可以直接用的**

**013618F0  mov dword ptr [ebp-8],ecx //分配栈控件存放this指针**

**013618F3  mov ecx,dword ptr [this]  //分获取this指向的内容**

**013618F6  call CA::CA (28100Ah)**

**CA::CA()//CA的构造函数也在这里展开了**

**{**

**0136195F   pop ecx   //获取this指针，**

**01361960  mov dword ptr [ebp-8],ecx   //分配栈控件存放this指针**

**01361963  mov eax,dword ptr [this]   //分获取this指向的内容**

**01361966 mov dword ptr [eax],offset CA::`vftable' (1366754h)     //将CA的虚函数表的地址存放在地址为eax的地方，也就是this指向的地方**

**0136196C  mov eax,dword ptr [this] //将this指向的值付给eax，也就是将虚函数表的地址存放在eax**

**0136196F  mov dword ptr [eax+4],0 //将虚函数表地址的下四个字节赋0，也是 m\_x0 = 0;**

**01361976  mov eax,dword ptr [this] //将this指针指向的地址赋给eax，作为返回值**

**}**

**013618FB  mov eax,dword ptr [this]  //将this指向的值付给eax，也就是将虚函数表的地址存放在eax**

**013618FE mov dword ptr [eax],offset CB::`vftable' (1366740h)   //将CB的虚函数表的地址存放在地址为eax的地方，也就是this指向的地方**

**//其实这里将虚函数表的地址覆盖了，从用CA的虚函数表改为用CB的虚函数表**

**01361904  mov eax,dword ptr [this]  //将this指向的值付给eax，也就是将虚函数表的地址存放在eax**

**01361907  mov dword ptr [eax+8],1  //将虚函数表地址的第二个四个字节赋1，也就是 m\_x = 1;**

**0136190E  mov eax,dword ptr [this]  //将this指针指向的地址赋给eax，作为返回值**

...

### 3.2. 内存分析

我们再从内存角度看一下这个过程，从ebp-8为 0x0036F94C，ebp-8就是存放了this指针

**0x0036F94C** a0 66 22 00 cc cc cc cc 34 fa 36 00 fb 18 36 01 34 fb 36 00 00 00

从上述内存的前四个字节，获取this指针0x002266a0，这个地址所在的内存中存放的内容：

**0x002266A0** cd cd cd cd cd cd cd cd cd cd cd cd fd fd fd fd ab ab

代码段（**01361966 mov dword ptr [eax],offset CA::`vftable' (1366754h)**）执行之后，地址**0x002266A0**处的内存放生了变化：

**0x002266A0** 54 67 36 01 cd cd cd cd cd cd cd cd fd fd fd fd ab ab

再来看一下这段内存前四个字节（54 67 36 01）转换成地址就是0x0x01366754，这个地址就是CA虚函数表的地址。

代码段（**013618FE mov dword ptr [eax],offset CB::`vftable' (1366740h)**）这执行之后的变化，再观察**0x002266A0**处的内存：

**0x002266A0** 40 67 36 01 00 00 00 00 cd cd cd cd fd fd fd fd ab ab

这里虚函数表的地址变成了0x01366740，这个地址就是CB的虚函数表，this指向的地址变成了这样：

**0x002266A0** 40 67 36 01 00 00 00 00 01 00 00 00 fd fd fd fd ab ab

对于虚函数表指向的地址的内存，这里就不展开了，里面存放的就是类每一个虚函数的地址。还有**0x002266A0** 40 67 36 01 00 00 00 00 01 00 00 00 fd fd fd fd ab ab，高亮部分等到多继承时候再详细解释。

### 3.3. 虚函数

**//虚函数**

**virtual int fun\_v1(int x,int y)**

**{**

**...**

**003B1ADF pop ecx    //获取this指针，**

**003B1AE0 mov dword ptr [ebp-8],ecx //存放this指针在栈上**

**int z = m\_x0 + x;**

**003B1AE3 mov eax,dword ptr [this] //存放this指向的地址存放在eax**

**003B1AE6 mov ecx,dword ptr [eax+4]  //取this指向地址的后四个字节，其实就是去m\_x0的值**

**003B1AE9 add ecx,dword ptr [x]  //和传入的x相加**

**003B1AEC mov dword ptr [z],ecx //将m\_x0 + x保存在z中，这里z是在函数的栈空间，这里就不展开了**

**z += m\_x;**

**003B1AEF mov eax,dword ptr [this]  // 存放this指向的地址存放在eax**

**003B1AF2 mov ecx,dword ptr [z]   //将z的值存放在ecx中**

**003B1AF5 add ecx,dword ptr [eax+8] // 取this指向地址的后第二个四字节，其实就是取m\_x，然后和ecx中的值相加**

**003B1AF8 mov dword ptr [z],ecx //将值存放在z中**

**return z;**

**003B1AFB mov eax,dword ptr [z] //将z中的值放入eax中，作为返回值**

**}**

### 3.4. 析构函数

**//delete p**

**delete p;**

**009C180C mov eax,dword ptr [ebp-14h] //获取p的值**

**009C180F mov dword ptr [ebp-0E0h],eax //分配临时栈控件存放this**

**009C1815 mov ecx,dword ptr [ebp-0E0h] //又将栈空间的this存放在ecx中**

**009C181B mov dword ptr [ebp-0ECh],ecx //有一次分配临时栈空间存放ecx中的值**

**009C1821 cmp dword ptr [ebp-0ECh],0 //判断this指向的地方是否为0**

**009C1828 je wmain+0FFh (9C184Fh) //如果为0就直接跳过下面代码，不执行函数**

**009C182A mov esi,esp //保存esp**

**009C182C push 1 //将1压入,delete 压入1 ；delete[] 压入3？**

**//这个参数是什么意义,是一个flag**

**// vector destructor iterator还是不同的destructor**

**009C182E mov edx,dword ptr [ebp-0ECh] //将存放在栈空间的this值赋值给edx**

**009C1834 mov eax,dword ptr [edx]  //将地址为edx存放值相同的所存放的值赋给eax，就是将虚函数表赋给eax**

**009C1836 mov ecx,dword ptr [ebp-0ECh] 又将this指针放入ecx中**

**009C183C mov edx,dword ptr [eax] //将eax中的值赋给edx，就是虚函数表的地址**

**009C183E call edx //调用edx**

**//其实就是 CB::`scalar deleting destructor'**

**{**

**...**

**009C1B6F pop ecx**

**009C1B70 mov dword ptr [ebp-8],ecx //1放入ebp-8的栈空间中**

**009C1B73 mov ecx,dword ptr [this] //传入this**

**009C1B76 call CB::~CB (9C115Eh) //调用 CB::~CB ()**

**//CB::~CB()**

**{**

**...**

**009C1BDF pop ecx**

**009C1BE0 mov dword ptr [ebp-8],ecx**

**009C1BE3 mov eax,dword ptr [this]**

**009C1BE6 mov dword ptr [eax],offset CB::`vftable' (9C6740h) //将CB的需函数表地址赋给this指向的地方**

**//和构造函数相反**

**009C1BEC mov ecx,dword ptr [this] //传入this**

**009C1BEF call CA::~CA (9C1163h)  //调用 CA::~CA()**

**//CA::~CA()**

**{**

**...**

**009C1A9F pop ecx**

**009C1AA0 mov dword ptr [ebp-8],ecx**

**009C1AA3 mov eax,dword ptr [this]**

**009C1AA6 mov dword ptr [eax],offset CA::`vftable' (9C6754h)  //将CA的需函数表地址赋给this指向的地方**

**//和构造函数相反**

**...**

**}**

**}**

**009C1B7B mov eax,dword ptr [ebp+8] //获取**

**009C1B7E and eax,1  //将eax中的值与1 与操作 ,就是在 009C182C push 1 ，传入的flag**

**//这里与下一句配合，为了让标志位ZF(Zero Flag) = 1,这里如果eax和1与操作后为0就让下一句跳转**

**009C1B81 je CB::`scalar deleting destructor'+3Fh (9C1B8Fh) //如果eax与1为0  就跳过下面的操作，不执行delete操作**

**009C1B83 mov eax,dword ptr [this] //传入this指针**

**009C1B86 push eax //压入this指针**

**009C1B87 call operator delete (9C10AAh) //调用operator delete**

**009C1B8C add esp,4 //平衡栈**

**009C1B8F mov eax,dword ptr [this] //返回值**

**...**

**}**

**009C1840 cmp esi,esp**

**009C1842 call @ILT+415(\_\_RTC\_CheckEsp) (9C11A4h)**

**009C1847 mov dword ptr [ebp-10Ch],eax**

**009C184D jmp wmain+109h (9C1859h)**

**009C184F mov dword ptr [ebp-10Ch],0**

## 小结

这里介绍类函数，其实是以分析为主来介绍类函数，包括类的虚函数表以及this指针。

这里可以配置着《深入C++对象模型》一起学习。