函數調用方式详解

现代的编程语言的函数竟然有那麼多的调用方式。这些东西要完全理解还得通过汇编代码才好理解。他们各自有自己的特点
其实这些调用方式的差别在主要在一下几个方面

1.参数处理方式（传递顺序，存取(利用盏还是寄存器)）
2.函数的结尾处理方式（善后处理 如:栈的恢复由谁恢复？函数内恢复/还是调用后恢复）

以下是理论：
__cdecl　由调用者平栈，参数从右到左依次入栈 是C和C＋＋程序的缺省调用方式。每一个调用它的函数都包含清空堆栈的代码，
　　所以产生的可执行文件大小会比调用_stdcall函数的大。函数采用从右到左的压栈方式。VC将函数编译后会在函数名前面加上
　　下划线前缀。是MFC缺省调用约定
__stdcall ，WINAPI，CALLBACK ，PASCAL　由被调用者平栈，参数从右到左依次入栈 ._stdcall是Pascal程序的缺省调用方式，
　　通常用于Win32　Api中，函数采用从右到左的压栈方式，自己在退出时清空堆栈。VC将函数编译后会在函数名前面加上下划
　　线前缀，在函数名后加上"@"和参数的字节数

__fastcall 由被调用者平栈，参数先赋值给寄存器，然后入栈 “ 人 ” 如其名，它的主要特点就是快，因为它是通过寄存器来传送参数的
　　（实际上，它用ECX和EDX传送前两个双字（DWORD）或更小的参数，剩下的参数仍旧自右向左压栈传送，被调用的函数在返回前
　　清理传送参数的内存栈），在函数名修饰约定方面，它和前两者均不同.
　　_fastcall方式的函数采用寄存器传递参数，VC将函数编译后会在函数名前面加上"@"前缀，在函数名后加上"@"和参数的字节数。

__thiscall 由被调用者平栈，参数入栈，this 指针赋给 ecx 寄存器 仅仅应用于 “ C++ ” 成员函数。this指针存放于CX寄存器，参数从右
　　到左压。thiscall不是关键词，因此不能被程序员指定。

__declspec(naked) 这是一个很少见的调用约定，一般程序设计者建议不要使用。编译器不会给这种函数增加初始化和清理代码，
　　更特殊的是，你不能用return返回返回值，只能用插入汇编返回结果。这一般用于实模式驱动程序设计.
以下是实践：

```
int __stdcall test_stdcall(char para1, char para2)
{
    para1 = para2;
    return 0;
}
int __cdecl test_cdecl(char para, )
{
    char    p = '\n';
    va_list marker;
    va_start( marker, para );
    while( p != '\0' )
    {
        p = va_arg( marker, char);
        printf("%c\n", p);
    }
    va_end( marker );
    return 0;
}

int pascal test_pascal(char para1, char para2)
{
    return 0;
}

int __fastcall test_fastcall(char para1, char para2, char para3, char
para4)
{
    para1 = (char)1;
    para2 = (char)2;
    para3 = (char)3;
    para4 = (char)4;
    return 0;
}
__declspec(naked) void __stdcall test_naked(char para1, char
para2)
{
    __asm
    {
        push    ebp
        mov     ebp, esp
        push    eax
        mov     al,byte ptr [ebp + 0Ch]
        xchg    byte ptr [ebp + 8],al
        pop     eax
        pop     ebp
        ret     8
```

```
    }
//    return ;
}


int main(int argc, char* argv[])
{
    test_stdcall( 'a', 'b' );
    test_cdecl( 'c','d','e','f','g' ,'h' ,'\0');
    test_pascal( 'e', 'f' );
    test_fastcall( 'g', 'h', 'i', 'j' );
    test_naked( 'k', 'l');
    return 0;
}
```

汇编代码如下

```
int main(int argc, char* argv[])
{
00411350  push        ebp
00411351  mov         ebp,esp
00411353  sub         esp,0C0h
00411359  push        ebx
0041135A  push        esi
0041135B  push        edi
0041135C  lea         edi,[ebp-0C0h]
00411362  mov         ecx,30h
00411367  mov         eax,0CCCCCCCCh
0041136C  rep stos    dword ptr es:[edi]
    test_stdcall( 'a', 'b' );
0041136E  push        62h
00411370  push        61h
00411372  call        _test_stdcall@8
    test_cdecl( 'c','d','e','f','g' ,'h' ,'\0');
00411377  push        0
00411379  push        68h
0041137B  push        67h
0041137D  push        66h
0041137F  push        65h
00411381  push        64h
00411383  push        63h
00411385  call        _test_cdecl
0041138A  add         esp,1Ch ;恢复_test_cdecl参数压入前的堆栈指令
是: add esp,n*4 n=参数的数量
    test_fastcall( 'g', 'h', 'i', 'j' );
0041138D  push        6Ah
0041138F  push        69h
00411391  mov         dl,68h
```

```
00411393  mov         cl,67h
00411395  call        test_fastcall
   test_naked( 'k', 'l');
0041139A  push        6Ch
0041139C  push        6Bh
0041139E  call        _test_naked
   return 0;
004113A3  xor         eax,eax
}

int __stdcall test_stdcall(char para1, char para2)
{
004111F0  push        ebp
004111F1  mov         ebp,esp
004111F3  sub         esp,0C0h
004111F9  push        ebx
004111FA  push        esi
004111FB  push        edi
004111FC  lea         edi,[ebp-0C0h]
00411202  mov         ecx,30h
00411207  mov         eax,0CCCCCCCCh
0041120C  rep stos    dword ptr es:[edi] ;初始edi
   para1 = para2;
0041120E  mov         al,byte ptr [para2] ;mov al,byte ptr[ebp+c]
00411211  mov         byte ptr [para1],al ;mov byte ptr[ebp+8],al
   return 0;
00411214  xor         eax,eax
00411216  pop         edi
00411217  pop         esi
00411218  pop         ebx
00411219  mov         esp,ebp
0041121B  pop         ebp
0041121C ret      8 ;恢复到压入函数参数前堆栈,由于有两个参数所以
ret 8 相当于 pop eip 然后esp+8
}
int __cdecl test_cdecl(char para,... )
{
00411230  push        ebp
00411231  mov         ebp,esp
00411233  sub         esp,0D8h
0041123C  lea         edi,[ebp-0D8h]
00411242  mov         ecx,36h
00411247  mov         eax,0CCCCCCCCh
0041124C  rep stos    dword ptr es:[edi]
   char    p = '\n';
0041124E  mov         byte ptr [p],0Ah
```

```
    va_list marker;
    va_start( marker, para );
00411252  lea         eax,[ebp+0Ch]
00411255  mov         dword ptr [marker],eax
    while( p != '\0' )
00411258  movsx       eax,byte ptr [p]
0041125C  test        eax,eax
0041125E  je          test_cdecl+60h (411290h)
    {
        p = va_arg( marker, char);
00411260  mov         eax,dword ptr [marker]
00411263  add         eax,4
00411266  mov         dword ptr [marker],eax
00411269  mov         ecx,dword ptr [marker]
0041126C  mov         dl,byte ptr [ecx-4]
0041126F  mov         byte ptr [p],dl
        printf("%c\n", p);
00411272  movsx       eax,byte ptr [p]
00411276  mov         esi,esp
00411278  push        eax
00411279  push        offset string "%c\n" (41401Ch)
0041127E  call        dword ptr [__imp__printf (416180h)]
00411284  add         esp,8
0041128E  jmp         test_cdecl+28h (411258h)
    }
    va_end( marker );
00411290  mov         dword ptr [marker],0
    return 0;
00411297  xor         eax,eax
004112A9  mov         esp,ebp
004112AB  pop         ebp
004112AC  ret
}

int __fastcall test_fastcall(char para1, char para2, char para3, char
para4)
{
004112D0  push        ebp
004112D1  mov         ebp,esp
004112D3  sub         esp,0D8h
004112DD  lea         edi,[ebp-0D8h]
004112E3  mov         ecx,36h
004112E8  mov         eax,0CCCCCCCCh
004112ED  rep stos    dword ptr es:[edi]
004112EF  pop         ecx
004112F0  mov         byte ptr [ebp-14h],dl
```

```
004112F3  mov         byte ptr [ebp-8],cl
    para1 = (char)1;
004112F6  mov         byte ptr [para1],1
    para2 = (char)2;
004112FA  mov         byte ptr [para2],2
    para3 = (char)3;
004112FE  mov         byte ptr [para3],3
    para4 = (char)4;
00411302  mov         byte ptr [para4],4
    return 0;
00411306  xor         eax,eax
0041130B  mov         esp,ebp
0041130D  pop         ebp
0041130E  ret         8  ;由于使用了ecx ,edx 传递参数 本来4个参数只使用两push 所以这里是 ret 4*2
}


__declspec(naked) void __stdcall test_naked(char para1, char para2)
{
00411330  push        ebp     ;这里编译器没加入任何初始化和清栈的指令,你代码如何写它就复制过来
00411331  mov         ebp,esp
00411333  push        eax
00411334  mov         al,byte ptr [para2]
00411337  xchg        al,byte ptr [para1]
0041133A  pop         eax
0041133B  pop         ebp
0041133C  ret         8
}
```