# 深入C++探索---零 开始

## 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。

## 内存存储区域

1. 栈区（stack）

程序运行时由编译器自动分配，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。程序结束时由编译器自动释放。

2、堆区（heap）

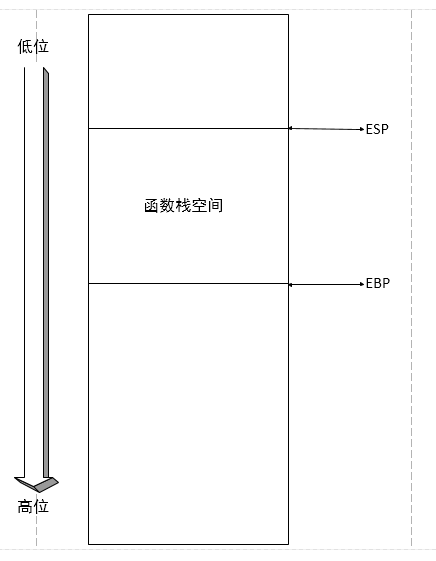
在内存开辟另一块存储区域。一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收 。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表。

1. 全局区（静态区）（static）

—编译器编译时即分配内存。全局变量和静态变量的存储是放在一块的，初始化的 全局变量和静态变量在一块区域， 未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。 - 程序结束后由系统释放

1. 文字常量区 常量字符串就是放在这里的。 程序结束后由系统释放
2. 程序代码区 存放函数体的二进制代码。

## 函数的栈空间



我们从简单到赋值来看函数的栈空间，这一篇的内容展现的栈空间如上图。EBP和ESP限定了函数的栈空间，栈地址是从上往下增长的，所以EBP的地址反而是大于ESP的。

## 三、简单函数

简单函数：void fun(){}

调用代码：fun1();

//

一个简单如fun的函数，这里我们用这个函数作为例子，没有参数，没有多余的代码。在调用时是怎么样的情况呢？

编译器在调用函数的时候，使用call fun address 替换原先的调用代码。

01302CFE call fun1 (13011D1h)

使用VS查看汇编代码，

会发现 13011D1h位置处又如下，使用jmp跳转的 13017C0h

fun1:

013011D1 jmp fun1 (13017C0h)

## 四、汇编代码解释

我们接下来一句一句分析下面的代码：

void fun1()

{

013017C0 push ebp //保存ebp ebp和esp是一对栈指针

//其中ebp是栈底指针，esp是栈顶指针

013017C1 mov ebp,esp //ebp = esp

013017C3 sub esp,0C0h //相当于 esp = esp – 0xC0,在栈上分配0xC0大小的临时空间

013017C9 push ebx //保存ebx

013017CA push esi //保存esi

013017CB push edi //保存edi

013017CC lea edi,[ebp-0C0h] //读入[ebp - 0c0h]的有效地址;

//就是刚分配

013017D2 mov ecx,30h //0x30改ecx，就是下面的循环次数

013017D7 mov eax,0CCCCCCCCh //将0CCCCCCCCh赋给eax

013017DC rep stos dword ptr es:[edi]

// rep便是重复执行，每执行一次ecx便减小1，知道为0停止循环

//stos 将eax的值拷贝到es:[edi]所指向的地址

//最后就是填满了临时存储区

}

013017DE pop edi //返回edi

013017DF pop esi //返回esi

013017E0 pop ebx //返回ebx

013017E1 mov esp,ebp //将此函数的ebp赋给esp，恢复至调用前的状态

013017E3 pop ebp //重新获得调用此函数前的ebp

013017E4 ret //返回

## 五、函数调用基本流程

接下来我们总结一下上面的行为：

一个函数调用会发生的：

1. 保存调用前的状态：保存ebp，ebx，esi，edi；
2. 分配栈临时空间（0xC0大小）；
3. 将临时空间赋上初值0CCCCCCCCh，这个值就是我们对临时变量没有赋值时，就是这个默认值；
4. 用户自己写的代码；
5. 恢复调用前的状态ebp，ebx，esi，edi，esp

## 六、总结

在最近阶段，越来越发现函数的重要性，windows的消息机制，一些常用架构的模块调用，都离不开函数和函数指针。

所以打算再写一个系列从内存和汇编角度着手，从函数开始重新认识一下C++。后面会从函数这个入口，开始介绍复杂的函数，类，和类的函数等。