C++ Tricks 2.2 I386 平台的内存布局

从 farseerfc.wordpress.com 导入

2.2 I386平台的内存布局

众所周知，I386是32位体系结构。因此对于绝大多数I386平台的C++编译器而言，sizeof(int)=sizeof(long)=sizeof(void*)=4。当然C++标准对此没有任何保证，我们也不应该试图编写依赖于此的代码。

32位指针的可寻址空间为4GB。为充分利用这么大的寻址空间，也是为了支持其它更先进的技术比如多任务技术或者动态链接库技术，WinNT使用虚拟内存技术，给与每个应用程序全部4GB的内存空间。4GB的地址被一分为二，前2GB供应用程序自己使用，后2GB由系统内核分配和管理。这2GB的内存地址，通常被划分成3种内存区使用：

1 代码及静态数据区

由代码加载器从动态链接库镜像(通常是exe或dll文件)加载，通常定位到镜像文件中指定的基址开始的内存区。如果基址所在内存已被占用，动态连接器会将代码或数据重定向到其它可用地址。

在C++中，静态数据包括：名字空间(namespace)和全局(global)对象、函数的static对象、类的static数据成员。这些静态数据由编译器分配地址(但可能被重定向)，由静态连接器写入代码文件(通常是exe或dll)的静态数据区段。所以标准说，这些静态数据在编译期就已经具有地址。

2 栈(Stack)

栈是最常用的动态数据存储区，所有函数的non-static对象和函数参数都在程序运行期在栈上分配内存。在数据结构中，术语“栈(Stack)”意指先进后出(FILO，First In Last Out)，与“队列(Queue)”所指的FIFO相对。相对于基于堆的对象分配技术，默认使用栈的对象分配有两点优势：

一、栈的FILO与人的思维方式相同

现实生活中有许多事例都使用FILO的方式，比如人们必须先提起话筒再拨打号码，而后挂断电话之后再放下话筒。使用FILO的栈，可以保证事物的销毁顺序以其诞生顺序相反的顺序进行，不会产生在挂断电话之前就放下话筒的尴尬。

二、栈的分配管理仅需要两个额外指针：栈顶(esp)和栈底(ebp)指针

从实现的技术层面而言，栈的管理比其它动态分配技术要简单很多。I386平台上的动态栈管理，仅需要栈顶和栈底两个指针。这两个指针的存储显然不能放置于栈中，置于静态数据区又有损效率。I386平台为管理动态栈专门预留了两个通用寄存器变量esp与ebp，分别代表栈顶(esp,Extended Stack Pointer)与栈底(Extended Bottom Pointer)指针。其中的extended代表它们是32位指针，以区分16位的sp和bp寄存器。

栈是动态存储区的特点，表明它的内存占用将随着程序

的运行而变化。I386平台上WinNT将应用程序的栈置于程序空间，向下增长。程序初始化时，由操作系统将esp指向系统分配的栈空间的顶部。当程序需要在栈上分配变量时，就将esp减去变量所需字节数，这被称作“压栈(Push)”；随后又要销毁变量时，就将esp加上变量所需字节数，这被称作“弹出(Pop)”。esp与ebp两者之间所夹的空间，就是当前函数正在使用的栈空间。由于栈向下增长，esp(栈顶)的值总是小于ebp(栈底)的值，新分配的变量地址总是小于旧变量的地址。

3 堆(Heap)和自由存储区

栈中的变量对于分配与释放的顺序有特定要求，这在一定程度上限制了栈的适用范围。面向对象(OO，Object Oriented)的程序设计思想也要求能自由地控制变量的分配与销毁。由此，现代操作系统都提供了被称作“堆(Heap)”的自由存储区，以允许由程序员控制的对象创建和销毁过程。C标准库函数malloc和free则是对操作系统提供的堆操作的封装。C++提供的自由存储区运算符new和delete则通常是malloc和free的又一层封装。

操作系统经由malloc和free控制对堆的访问。堆的存储管理技术各不相同，简单的使用双链表管理，复杂的可以比拟一个完整的文件系统。

由于额外的管理需求，使用系统提供的通用分配器在堆上分配和销毁变量的代价，无论从空间角度还是效率角度而言，都比在栈上分配对象要高昂很多。对于sizeof上百的大型对象，这样的高昂代价还是可以接受的，但是对于sizeof只有个位数的小对象，这样的代价通常是一个数量级的差距。正因为这个原因，STL不使用new和delete，转而使用分配子(alllocor)分配对象。