写一个好的C++程序，我们要懂得好多东西，比如说最基本的面向对象编程思想，C++的封装、继承、多态机制，设计模式等，还有一个很重要的内容便是性能优化，像C/C++这种接近底层的语言，追求的就是性能，与之相关的一项内容便是内存管理，内存分配要合理，禁止破坏内存，不能有内存泄漏，操作不好的话，程序可能会越运行越慢，要不就是莫名其妙的crash，于是令人头疼的bug排查就这样开始了。

1、内存分配方式

在C++中，内存分为5个区：堆、栈、自由存储区、全局/静态存储区和常量存储区。

**堆**：由new分配的内存块，分配成功后需要我们在适当的时候通过delete释放内存，为了避免内存泄漏，new和delete要成对使用。

**栈**：执行函数时，函数内的局部变量（也叫自动变量）在栈上创建存储单元，函数执行结束时这些存储单元被自动释放，正因如此，函数返回局部变量的指针或引用是错误的。

**自由存储区**：由malloc等函数分配的内存块，与堆类似，但内存释放是用free来完成的，malloc/free与new/delete的区别是前者属于库函数，而后者是操作符，除了分配/释放内存外，还会调用类的构造/析构函数。

**全局/静态存储区**：全局变量和静态变量被分配到同一块存储区，生命周期和作用域与局部变量不同。

**常量存储区**：顾名思义，内存中存放的是常量，不允许修改。

2、堆与栈的区别

堆、栈是经常提及的两个内存块，主要区别体现在以下几个方面。

**管理方式**：栈由编译器自动管理，栈上的变量离开作用域之后内存被自动回收，而堆由我们自己管理，动态创建与释放内存，管理起来较为复杂。

**生长方向**：从整体内存空间来看，栈位于高地址，向着内存地址减小的方向生长，堆位于低地址，向着内存地址增加的方向生长。

**空间大小**：栈空间较小，堆空间则很大。

**碎片问题**：栈的数据结构形式是后进先出，内存空间是一块连续的区域，不会有内存碎片问题，而堆则不同，频繁的new/delete会造成内存空间的不连续，从而产生堆碎片，影响程序性能。

**分配方式**：栈有两种分配方式，静态分配和动态分配，静态分配由编译器完成，如局部变量的内存分配和释放，动态分配通过调用alloca函数实现，但内存释放是由编译器完成的。堆内存分配则都是动态分配，需要我们自己分配内存并进行内存释放。

**分配效率**：栈的分配使用的是机器系统提供的数据结构，在计算机底层有专门的寄存器存放栈的地址，压栈出栈都有专门的指令执行，效率较高。而堆的分配是由C/C++函数库提供的，有一定的分配算法，为了分配成功，会去搜索可用的足够大小的空间，还有可能引发用户态和内核态的切换，效率较低。

3、正确使用内存

从上面介绍的几个方面可以看出，栈高效，堆灵活，具体如何使用需要根据实际情况而定，不过使用不当的话很有可能引发系统错误，下面简单总结一下常见的情况。

错误1：内存分配未成功，却使用了这个分配结果。

正确1: 内存分配是有可能失败的，在使用之前一定要检查分配结果是否不为NULL。

错误2：内存分配成功了，但是没有进行初始化就使用它。

正确2：内存分配成功后，虽然一些简单数据类型的变量为0，但复杂结构的变量就不确定了，所以一定要进行初始化。

错误3：内存越界。

正确3：内存越界常发生在数组、容器的使用中，使用前一定要确认其索引/下标有效。

错误4：没有释放已分配内存，造成内存泄漏。

正确4：分配了内存之后忘记释放内存是可能发生的，造成内存泄漏，如果内存泄漏严重，就会造成内存不足或内存耗尽，所以malloc/free、new/delete一定要配对使用。

错误5：函数返回了指向局部变量的指针或引用。

正确5：局部变量使用的是栈内存，函数返回时局部变量的内存被自动释放，这时再通过指针或引用来使用这个变量的话就出错了，所以返回指向局部变量的指针或引用是禁止的，如果返回值为指针或引用的话，可以返回静态变量或全局变量，或者返回堆上的变量。

错误6：误用野指针。

正确6：用free或delete释放了内存后，还继续使用这块内存，这是有问题的，在内存释放后一定要把指针设置为NULL，避免野指针。

4、关于double free

当C++类中定义了指针成员变量，使用的却是默认拷贝构造函数时，是有可能发生常见的double free问题的，具体可参考：

<http://blog.csdn.net/ieearth/article/details/49332709>

5、内存检测

Valgrind是一个内存检测工具，可以用来检测内存非法访问、内存泄漏等问题，具体可参考：

<http://blog.csdn.net/ieearth/article/details/48377493>