C 内存对象大会战计算机等级考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/646/2021_2022_C___E5_86_85_E5_AD_98_E5_c97_646147.htm 如果一个人自称为程序高手，却对内存一无所知，那么我可以告诉你，他一定在吹牛。用C或C 写程序，需要更多地关注内存，这不仅仅是因为内存的分配是否合理直接影响着程序的效率和性能，更为主要的是，当我们操作内存的时候一不小心就会出现问题，而且很多时候，这些问题都是不易发觉的，比如内存泄漏，比如悬挂指针。笔者今天在这里并不是要讨论如何避免这些问题，而是想从另外一个角度来认识C 内存对象。 我们知道，C 将内存划分为三个逻辑区域：堆、栈和静态存储区。既然如此，我称位于它们之中的对象分别为堆对象，栈对象以及静态对象。那么这些不同的内存对象有什么区别了？堆对象和栈对象各有什么优劣了？如何禁止创建堆对象或栈对象了？这些便是今天的主题。 一．基本概念 先来看看栈。栈，一般用于存放局部变量或对象，如我们在函数定义中用类似下面语句声明的对象： Type stack_object . stack_object便是一个栈对象，它的生命期是从定义点开始，当所在函数返回时，生命结束。 另外，几乎所有的临时对象都是栈对象。比如，下面的函数定义： Type fun（Type object） . 这个函数至少产生两个临时对象，首先，参数是按值传递的，所以会调用拷贝构造函数生成一个临时对象object_copy1 ，在函数内部使用的不是使用的不是object，而是object_copy1，自然，object_copy1是一个栈对象，它在函数返回时被释放；还有这个函数是值返回的，在函数返回时，如果我们不考虑返回值优化（NRV

），那么也会产生一个临时对象object_copy2，这个临时对象会在函数返回后一段时间内被释放。比如某个函数中有如下代码： Type tt ,result . //生成两个栈对象 tt = fun（tt） . //函数返回时，生成的是一个临时对象object_copy2 上面的第二个语句的执行情况是这样的，首先函数fun返回时生成一个临时对象object_copy2 ，然后再调用赋值运算符执行 tt = object_copy2 . //调用赋值运算符 看到了吗？编译器在我们毫无知觉的情况下，为我们生成了这么多临时对象，而生成这些临时对象的时间和空间的开销可能是很大的，所以，你也许明白了，为什么对于“ 大 ”对象最好用const引用传递代替按值进行函数参数传递了。 接下来，看看堆。堆，又叫自由存储区，它是在程序执行的过程中动态分配的，所以它最大的特性就是动态性。在C 中，所有堆对象的创建和销毁都要由程序员负责，所以，如果处理不好，就会发生内存问题。如果分配了堆对象，却忘记了释放，就会产生内存泄漏；而如果已释放了对象，却没有将相应的指针置为NULL，该指针就是所谓的“ 悬挂指针 ”，再度使用此指针时，就会出现非法访问，严重时就导致程序崩溃。 那么，C 中是怎样分配堆对象的？唯一的方法就是用new（当然，用类malloc指令也可获得C式堆内存），只要使用new，就会在堆中分配一块内存，并且返回指向该堆对象的指针。 再来看看静态存储区。所有的静态对象、全局对象都于静态存储区分配。关于全局对象，是在main()函数执行前就分配好了的。其实，在 main()函数中的显示代码执行之前，会调用一个由编译器生成的_main()函数，而_main()函数会进行所有全局对象的的构造及初始化工作。而在main()函数结束之前，会调用由编译器生成的exit函

数，来释放所有的全局对象。比如下面的代码： void main（void）{ …… // 显式代码 } 实际上，被转化成这样： void main（void）{ _main（）. //隐式代码，由编译器产生，用以构造所有全局对象 …… // 显式代码 …… exit（） . // 隐式代码，由编译器产生，用以释放所有全局对象 } 所以，知道了这个之后，便可以由此引出一些技巧，如，假设我们要在main()函数执行之前做某些准备工作，那么我们可以将这些准备工作写到一个自定义的全局对象的构造函数中，这样，在main()函数的显式代码执行之前，这个全局对象的构造函数会被调用，执行预期的动作，这样就达到了我们的目的。刚才讲的是静态存储区中的全局对象，那么，局部静态对象了？局部静态对象通常也是在函数中定义的，就像栈对象一样，只不过，其前面多了个static关键字。局部静态对象的生命期是从其所在函数第一次被调用，更确切地说，是当第一次执行到该静态对象的声明代码时，产生该静态局部对象，直到整个程序结束时，才销毁该对象。 还有一种静态对象，那就是它作为class的静态成员。考虑这种情况时，就牵涉了一些较复杂的问题。 第一个问题是class的静态成员对象的生命期，class的静态成员对象随着第一个class object的产生而产生，在整个程序结束时消亡。也就是有这样的情况存在，在程序中我们定义了一个class，该类中有一个静态对象作为成员，但是在程序执行过程中，如果我们没有创建任何一个该class object，那么也就不会产生该class所包含的那个静态对象。还有，如果创建了多个class object，那么所有这些object都共享那个静态对象成员。 第二个问题是，当出现下列情况时： class Base { public: static Type s_object . } class Derived1 :

public Base / / 公共继承 { … …// other data } class Derived2 : public Base / / 公共继承 { … …// other data } 