C 内存管理基础之new&0delete PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/274/2021_2022_C___E5_86_85_E5_AD_98_E7_c67_274977.htm 内存管理的基础是要知道怎么获得以及释放内存，如你所知，在C/C 中就是调用new和0delete操作。 1. 分清operator new和new operator 全局函数operator new通常这样声明： void * operator new(size_t size). 返回值类型是void*，表示其返回的是一个未经处理（raw）的指针，指向未初始化的内存。参数size_t确定分配多少内存。你能增加额外的参数重载函数operator new，但是第一个参数类型必须是size_t。头文件中有一个很好的重载的例子，那就是placement new，它看上去象这样： void * operator new(size_t, void *location) { return location. } 这初看上去有些陌生，但它却是new操作符的一种常见重载方法，使用一个额外的变量buffer，当new操作符隐含调用operator new函数时，把这个变量传递给它。被调用的operator new函数除了持有强制的参数size_t外，还必须接受void*指针参数，指向构造对象占用的内存空间。未被使用的（但是强制的）参数size_t没有参数名字，以防止编译器警告说它未被使用。在使用placement new的情况下，调用者已经获得了指向内存的指针，因为调用者知道对象应该放在哪里。placement new需要做的就是返回传递给它的指针。 我们更经常使用的new是new操作符(new operator)，而非操作符new(operator new)，如当你使用new操作符构建一个对象的时候，实际上做了两件事情，一是调用operator new函数获取内存，二是调用对象的构造函数，如：string *ps = new string("Hello, world!"). 它完成与下面代码相

似的功能： void *memory = operator new(sizeof(string)). // 为String对象得到未经处理的内存 call string::string("Hello, world!") on *memory. // 调用构造函数初始化内存中的对象 string *ps = static_cast(memory). // ps指针指向新的对象 注意第二步中构造函数的调用只能由编译器完成，用户是不允许这样操作的，也就是说如果你想建立一个堆对象就必须用new操作符，不能直接像上面一样调用构造函数来初始化堆对象。new操作符(new operator)是编译器内置的，其行为被语言固定下来，不受用户控制。但是它们所调用的内存分配函数也就是操作符new(operator new)则可以根据需要进行重载。试着回顾new操作符(new operator)与操作符new(operator new)的关系，如果你想在堆上建立一个对象，应该用new操作符。它既分配内存又为对象调用构造函数。如果你仅仅想分配内存，就应该调用operator new函数，它不会调用构造函数。如果你想定制自己独有的内存分配过程，你应该重载全局的operator new函数，然后使用new操作符，new操作符会调用你定制的operator new。如果你想在一块已经获得指针的内存里建立一个对象，应该使用placement new。 最后需要记住的一点是，0delete和new一样具有以上的特性，只是需要注意的一点是delte操作符中是首先调用对象的析构函数，然后再调用operator 0delete函数的。 2. 针对数组的new[]和0delete[]操作建立数组时new操作符(new[])的行为与单个对象建立(new)有少许不同： 第一是内存不再调用用operator new函数进行分配，代替以operator new[]函数(常称作array new)。它与operator new一样能被重载，允许定制数组的内存分配，就象定制单个对象内存分配一样。 第二个不同是new[]操作时调用构造函数

的数量。对于new[]而言，在数组里的每一个对象的构造函数都必须被调用。 0delete[]操作符的语义基本上和new[]相同，他们的实现类似这样： void * operator new[](size_t size) { cout int *g =(int *) malloc(sizeof(size)). return g. } void operator 0delete[](void* p) { cout free(p). } 3. operator new和0delete函数的实现 operator new实际上总是以标准的C malloc()完成，虽然并没有规定非得这么做不可。同样，operator 0delete也总是以标准得C free()来实现，不考虑异常处理的话他们类似下面的样子： extern void* operator new(size_t size) { if(size == 0) size = 1. // 这里保证像 new T[0] 这样得语句也是可行的 void *last_alloc. while(!(last_alloc = malloc(size))) { if(_new_handler) (*_new_handler)(). else return 0. } return last_alloc. } extern void operator 0delete(void *ptr) { if(ptr) // 从这里可以看出，删除一个空指针是安全的 free((char*)ptr). } 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com