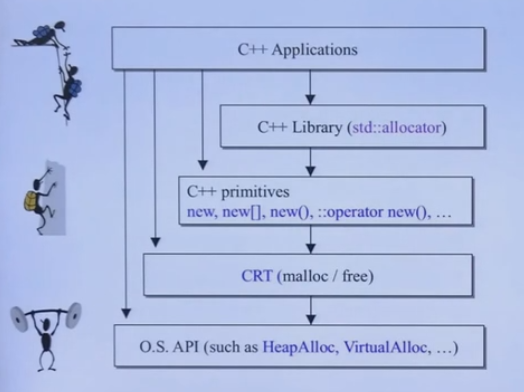
注意区分：

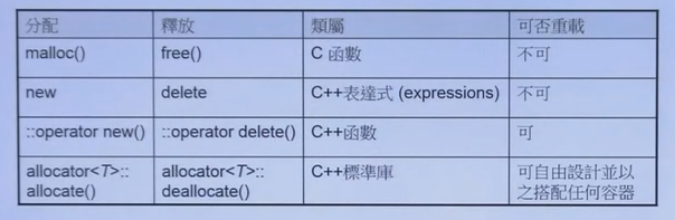
1. new和delete expression是在应用程序层面的
2. ::operator new是全局的操作符，应用程序使用new分配内存的时候，会调用每个类::operator new，然后再转到全局的::operator new，最终全局的opreator new调用malloc。全局的operator new可以被重载，但是一般不重载它，因为所有分配内存的函数都会调用到它。
3. 内存管理的层次

内存管理的层次可以分为五个层次，c++ application（应用层）、c++ library（c++标准库，例如std::allocator）、c++ pimitives（c++基本单元，例如new new[] new() operator::new ::operator::new）、CRT（malloc free）、OS API。

一层一层都是逐层封装的，对于上层的application来说，我们可以直接调用顶层的c++ library，也可以调用底下几层的API。



1. c++分配内存的基本工具（函数）

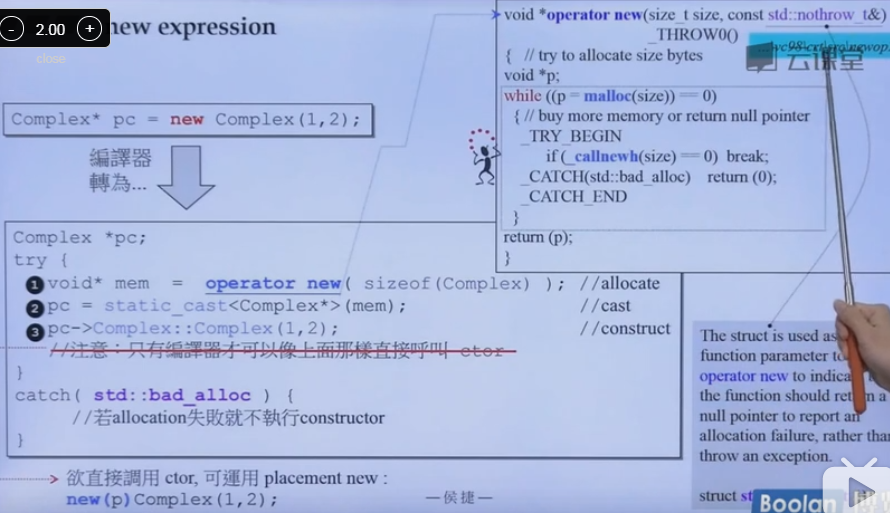


1、C++ new expression（new表达式）的调用流程

1. 调用operator new（操作符）分配内存，这个operator是可以被重载的

* 调用malloc，成功就返回
* 如果不成功，会进入\_callnewh函数里面，可以释放一些内存，然后再重新调用malloc，直到分配内存成虫

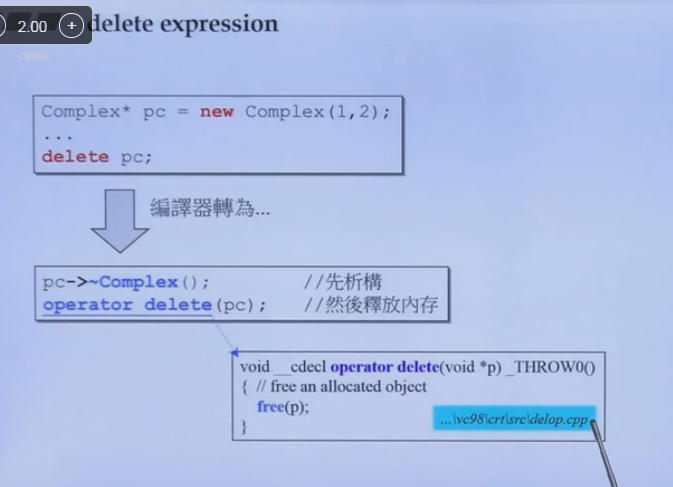
1. 类型转换
2. 调用类的构造函数



2、C++ delete expression调用流程

1. 调用析构函数
2. 调用operator delete释放内存

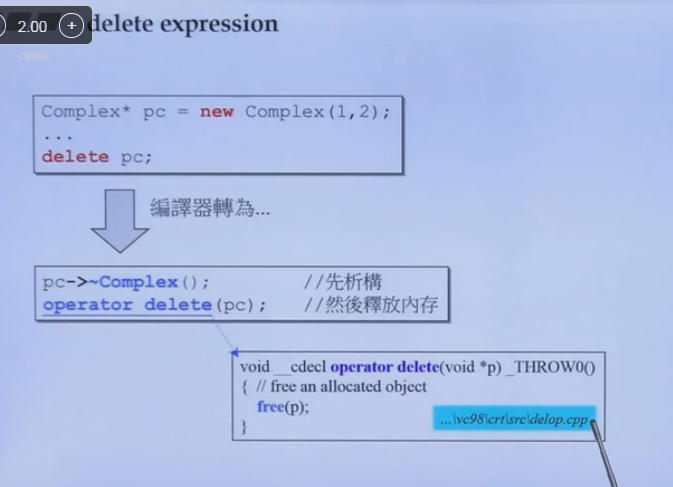
调用free释放内存



3、C++ delete expression调用流程

1. 调用析构函数
2. 调用operator delete释放内存

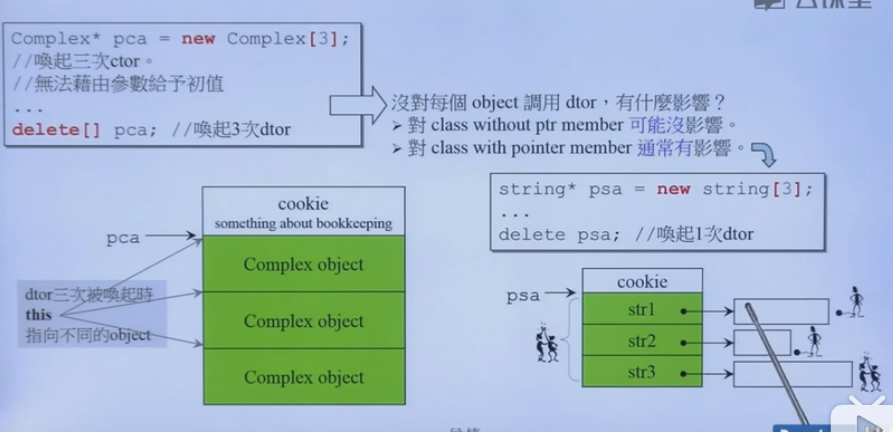
调用free释放内存



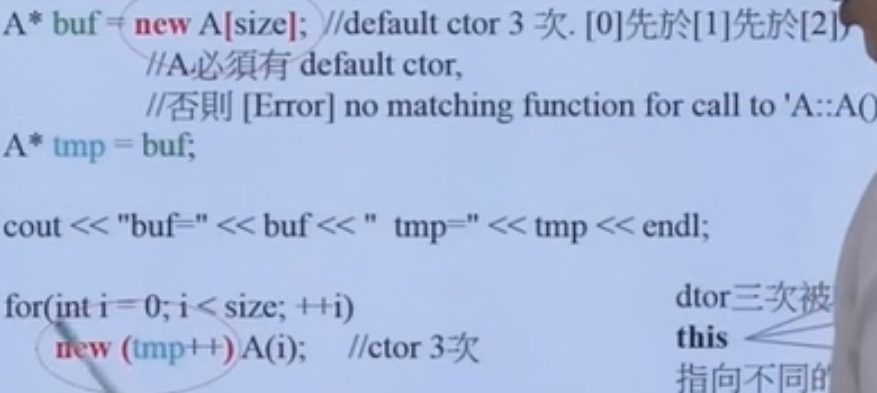
1. 调用new expression申请数组内存（简析何时会发生内存泄漏）
2. 调用delete [] ptr在释放对象指针对应的内存（整个数组的大小），还会调用每个数组里面对象对应的析构函数，所以如果析构函数里面有释放内存的话，调用delete ptr就会发生内存泄漏。

内存泄漏发生在：如果对象里面本身也有分配内存，并且分配的内存在析构函数进行释放。那么调用delete ptr相比于delete [] ptr就会发生内存泄漏。因为delete ptr本身只调用一次析构函数，然后再去释放对象对应内存。

当然，我们写代码的时候使用new array[n]，应该统一用delete [] array进行释放。

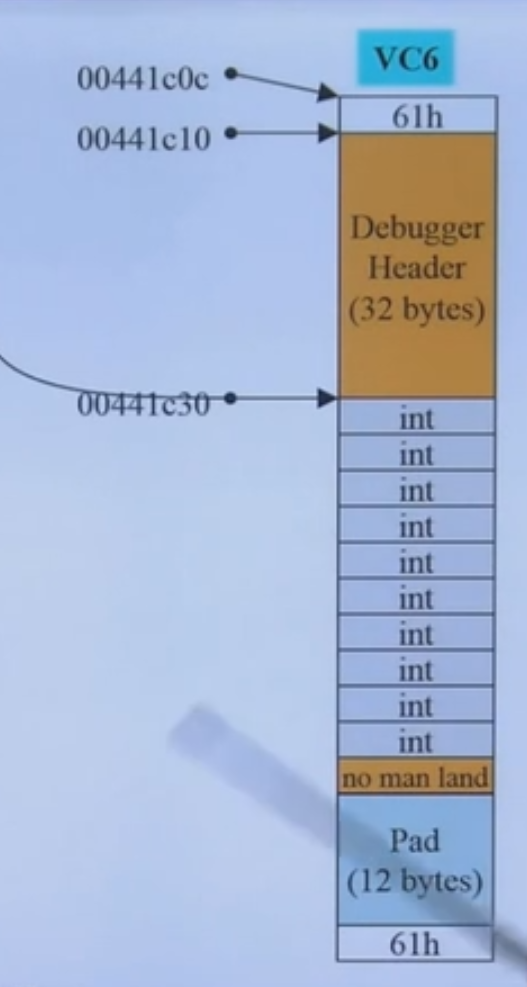


5、Placement new的用法

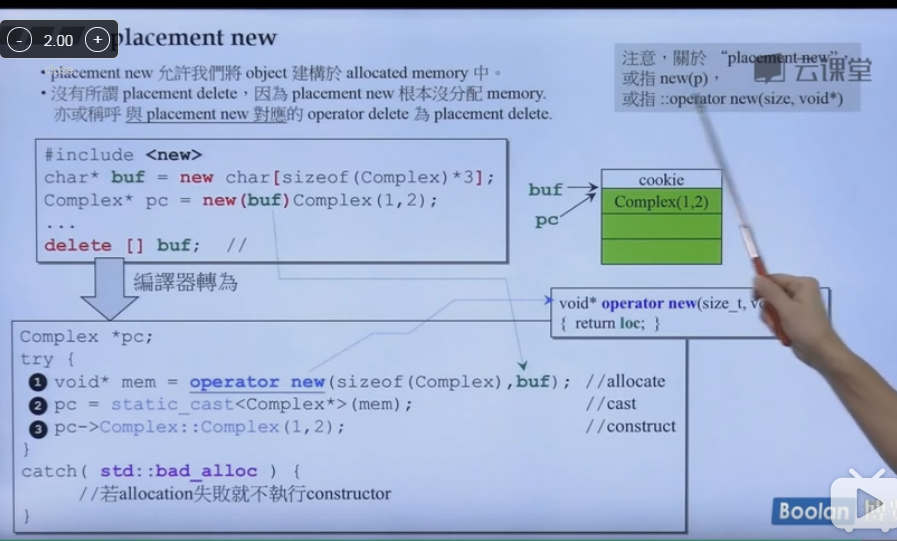


1. c++中malloc的内存分配图

分为以下几个部分，本身分配的内存、Debugger header（标注申请的内存大小）、文件头、pad data（是为了保证整块内存是16bytes的倍数）

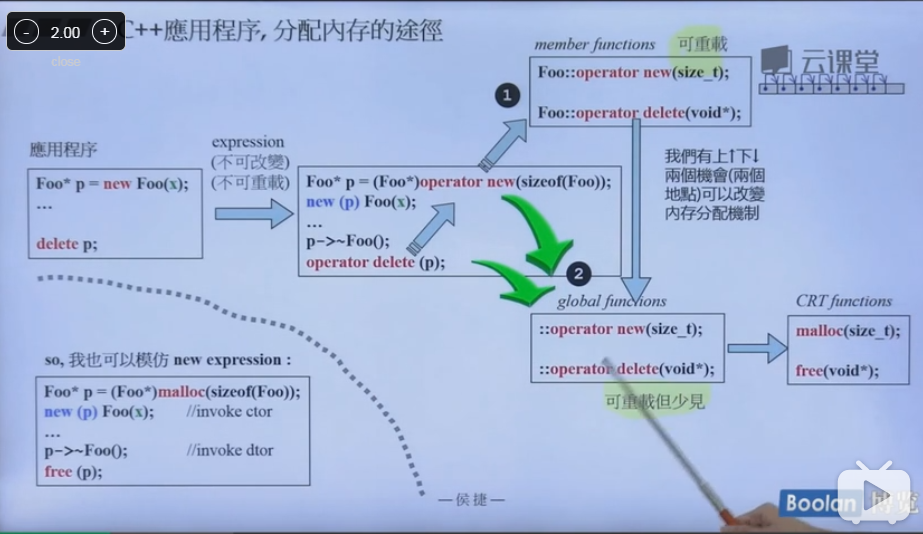


7、Placement new底层调用流程

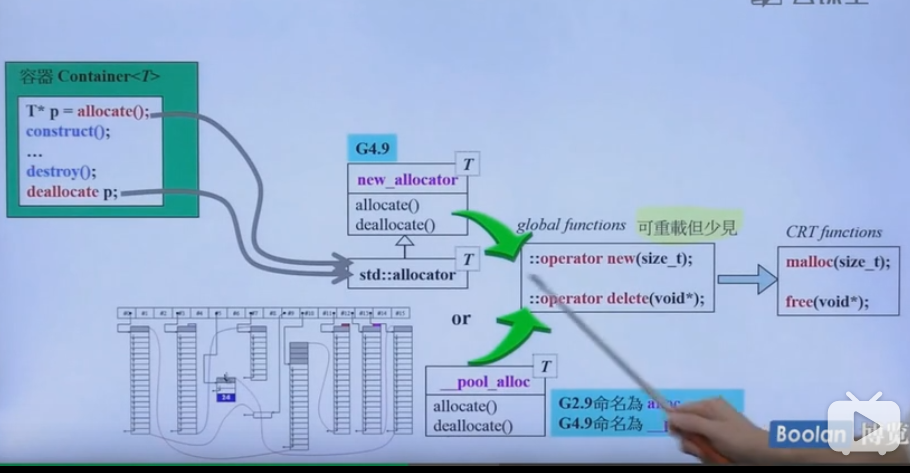


8、C++应用程序使用new分配内存的流程

1. 一般重载类::operator new
2. 应用程序调用的new expression不可以重载
3. ::operator new（全局）可以重载，但一般不重载，因为他影响太多东西了



1. 容器分配内存的流程

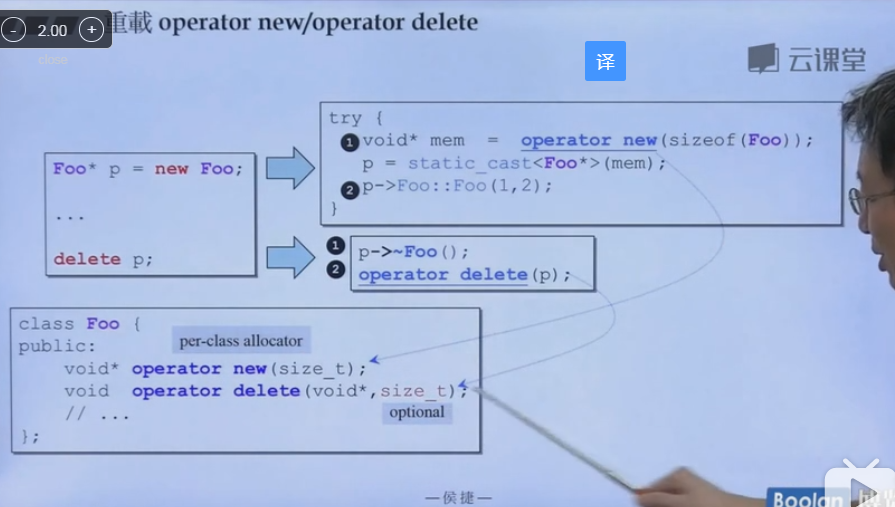


二、c++内存重载

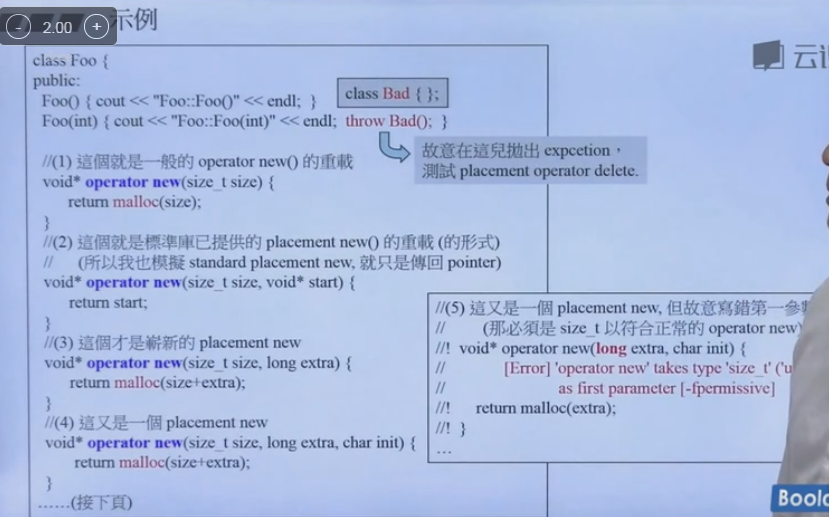
1、重载全局的 ::operator new



2、类里面重载operator new和operator delete



3、重载placement new，两个参数



4、Basic\_string重载operator new（调用分配器分配内存）

