**一.c++中的set\_new\_handler和new\_handler**

一个栗子：

#include <iostream>

using namespace std;

int main(void)

{

int \* p = new int[1000000000];

if (0 == p)

{

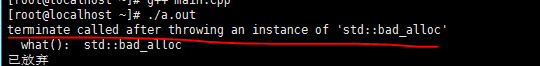
cout<<"bad NEW!!!"<<endl;

}

return 0;

}

结果：



咋一看，写的没问题啊，new失败的时候会通过if判断！ 其实，这里的 if ( 0 == p ) 完全是没意义的,C++ 里，如果 new 分配内存失败，默认是**抛出异常**的。所以new成功与否都不会走进if！

正确写法：

try

{

int \* p = new int[1000000000];

}

catch ( const bad\_alloc& e )

{

cout<<"bad NEW!!!"<<endl;

return -1;

}

Operator new的源码：

void \*\_\_CRTDECL operator new(size\_t size) \_THROW1(\_STD bad\_alloc)

{

void \*p;

while ((p = malloc(size)) == 0)

if (\_callnewh(size) == 0)

{

static const std::bad\_alloc nomem;

\_RAISE(nomem);

}

return (p);

}

调用malloc失败后，会调用\_callnewh函数，如果该函数的返回值为0，则抛出异常。所以new分配内存失败后，会抛出异常。

可通过set\_new\_handler设置函数句柄，当new失败时，做出相应的处理！

**二．为什么要重载operator new/delete**

通常我们为了一些目的而使用new和delete的内存分配系统，但是在特殊情况下，它并不能够满足需要，原因如下：

1）频繁的new/delete造成太多的内存碎片，以至于没有大块的连续内存导师下一次new失败的情况；

2）频繁的new/delete变成了速度的瓶颈；

3）某些项目需要对内存作统一的处理(规避内存泄漏，段错误等情况)；

注意事项：

1）这里说的重载只是改变了原有的内存分配方法，编译器将用重载的new代替默认版本去分配内存；

2）如果new的对象有构造函数，编译器会负责调用构造函数；

**三．如何重载**

重载的new必须有一个size\_t参数，这个参数由编译器产生并传递给operator new，它是要分配内存的对象的长度，然后返回一个指向等于这个长度（或者大于这个长度）的对象的指针；如果没有找到存储单元，new-handler会抛出异常！当new成功是，编译器会负责调用对于的构造函数，这个不在我们的控制范围了，在这里不做探讨！

1）重载全局operator new和delete

2）对于一个类重载new和delete

为一个类重载new和delete的时候，尽管不必显式的使用static，但是实际上仍是在创建static成员函数。它的语法也和重载任何其它运算符一样。当编译器看到使用new创建自己定义的类的对象时，它选择成员版本的operator new()而不是全局版本的new()，但是全局版本的new和delete仍为所有其他类型对象使用（除非它们也有自己的new和delete），这个和全局变量、局部变量的意思是一样的，应该很好懂吧。

https://www.cnblogs.com/BaiYiShaoNian/p/4681367.html