lidan

effective C++ 条款 49：了解new-handler的行为

当operator new无法满足某一内存分配需求时，会抛出异常。再抛出异常以反映一个未获满足的内存需求之前，它会先调用客户指定的错误处理函数，new-handler。为了指定这个“用以处理内存不足”的函数，客户必须调用set-new-handler，那是声明于<new>的一个标准函数库函数：

```
namespace std{
    typedef void (*new_handler)();
    new_handler set_new_handler(new_handler p) throw();
}
```

thow ( ) 是一份异常明细，表示函数不会抛出任何异常。

set_new_handler参数是一个指针指向operator new无法分配足够内存时该被调用的函数。返回指向set_new_handler被调用前正在执行的那个new_handler函数。

```
void outOfMem()
{
    std::cout << "Unable to satisfy request for memory\n";
    std::abort();
}
int main()
{
    std::set_new_handler(outOfMem);
    int* pBigDataArray = new int[1000000000000L];
}
```

一个设计良好的new_handler必须做以下事情：

1.让更多内存可使用。造成operator new内的下一次内存分配动作可能成功。一个做法是，程序开始执行就分配一大块内存，在new-handler第一次被调用，将它们释还给程序使用。

< 2012年2月 >

日	一	二	三	四	五	六
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	1	2	3
4	5	6	7	8	9	10

常用链接

我的随笔 我的评论
我的参与 最新评论
我的标签 更多链接

随笔分类

1. c/c++(72)
2. java(2)

2.安装另一个new-handler。如果这个new-handler无法取得更多可用内存，或许它知道另外有个new-handler有此能力。目前这个就可以安装另外那个以替换自己（只需调用set_new_handler）。下次当operator new调用那个new-handler，调用的将是最新安装的那个。（这个旋律的变奏之一就是让new-handler修改自己的行为，于是当他下次被调用就会做些不同的事。为达此目的，做法之一就是令new-handler修改“会影响new-handler行为”的static数据、namespace数据、或global数据。）

3.卸除new-handler。就是将null指针传给set_new_handler。一旦没有安装任何new_handler，operator new会在内存分配不成功时抛出异常。

4.抛出bad_alloc（或派生自bad_alloc）的异常。这样的异常不会被operator new捕获，因此会传到内存索求处。

5.不反回。通常调用abort或exit。

有时你想以不同方式处理内存分配失败情况，你希望视被分配物属于哪个class而定：

c++并不支持class专属之new-handler，其实也不需要。你可以自己实现出这种行为。只需令每一个class提供自己的set_new_handler和operator new即可。其中set_new_handler使客户指定class专属的new-handler。

现在，假设你打算处理Widget class 的内存分配失败情况。

```
class Widget{
public:
    static std::new_handler set_new_handler(std::new_handler p) throw();
    static void* operator new(std::size_t size) throw(std::bad_alloc);
private:
    static std::new_handler currentHandler;
};
```

static成员必须在class定义之外定义（除非它们是const而且是整数型）：

```
std::new_handler Widget::currentHandler = 0;
std::new_handler Widget::set_new_handler(std::new_handler p) throw()
{
```


3. linux(12)
4. VC(6)
5. web(2)
6. 笔试面试(2)
7. 方法论(1)
8. 工具(10)
9. 设计模式(3)
10. 数据结构和算法(2)
11. 数据库(1)
12. 网络(7)

随笔档案

1. 2012年8月 (1)
2. 2012年7月 (3)
3. 2012年6月 (1)
4. 2012年5月 (14)
5. 2012年4月 (9)
6. 2012年3月 (3)
7. 2012年2月 (29)
8. 2012年1月 (25)
9. 2011年11月 (7)
10. 2011年10月 (1)
11. 2011年9月 (3)
12. 2011年8月 (15)
13. 2011年7月 (18)
14. 2011年6月 (2)
15. 2011年5月 (6)

最新评论

1. 1. Re:【转】win7建立FTP
2. C:\Windows\System32\inetsrv\inetinfo.exe，没有怎么破
3. --会飞的海

```
    std::new_handler oldHandler = currentHandler;
    currentHandler = p;
    return oldHandler;
}
```

Widget的operator new做以下事情：

1.调用标准set_new_handler，告知Widget的错误处理函数，这回将Widget的new_handler安装为global new-handler。

2.调用global operator new，执行实际的内存分配。如果分配失败，global operator new会调用Widget的new_handler，才刚被安装为global new-handler。如果global operator new最终无法分配足够内存，会抛出一个bad_alloc异常。此情况下Widget的operator new必须恢复原本的global new_handler，然后再传播该异常。为确保原本的new_handler总是能够被重新安装回去，Widget将global new_handler视为资源并遵守条款13忠告，运用资源管理对象防止资源泄漏。

3.如果global operator new能够分配足够一个Widget对象所用的内存，Widget的operator new会返回一个指针，指向分配所得。Widget 析构函数会管理global new_handler，它会自动将widget's operator new被调用前的那个global new_handler恢复回来。

```
class NewHandlerHolder{
public:
    explicit NewHandlerHolder(std::new_handler nh)
        :handler(nh){}                                  //取得目前的
new_handler
    ~NewHandlerHolder()
    {
        std::set_new_handler(handler);                  //释放它
    }
private:
    std::new_handler handler;                           //记录下来
    NewHandlerHolder(const NewHandlerHolder&);          //阻止copying
    NewHandlerHolder& operator=(const NewHandlerHolder&);
};
```

这就使得Widget's operator new的实现相当简单：

4. 2. Re:g++参数介绍
5. 赞
6. --www_elesos_com站长
7. 3. Re:【转】win7建立FTP
8. 很详细，帮助很大，有个小问题，C:\Windows\System32\inetsrv\inetinfo.exe，这个程序没有找到，外网访问不了
9. --淡蓝深海

阅读排行榜

1. g++参数介绍(16101)
2. 【转】win7建立FTP(12576)
3. 【转】Eclipse的HTML编辑器(9161)
4. [转]Ubuntu 用VSFTP搭建FTP服务…
5. [转]基于MFC的ActiveX控件开发(4550)

评论排行榜

1. 【转】win7建立FTP(4)
2. g++参数介绍(2)
3. effective C++ 条款 8：别让异常逃离…
4. 【转】linux 下的UDP client/server …
5. 【转】Eclipse的HTML编辑器(1)

推荐排行榜

1. g++参数介绍(3)
2. 【转】win7建立FTP(3)
3. [转]基于MFC的ActiveX控件开发(1)
4. effective C++ 条款 34：区分接口继…

```
void* Widget::operator new(std::size_t size) throw(std::bad_alloc)
{
    NewHandlerHolder h(std::set_new_handler(currentHandler)); //安装 Widget的new_handler
    return ::operator new(size);                              //分配内存或抛出异常。
}                                                             //恢复global new_handler
```

Widget的客户应该这样使用其new_handling：

```
void outOfMem();
Widget::set_new_handler(outOfMem);
Widget* pw1 = new Widget;                  //如果内存分配失败，调用outOfMem
std::string* ps = new std::string;         //如果内存分配失败，调用global new_handling函数
Widget::set_new_handler(0);
Widget* pw2 = new Widget;                  //如果内存分配失败，立刻抛出异常，new_handling为null
```

实现这一方案的代码并不因class的不同而不同，因此在它处加以复用是个合理的构想。一个简单的做法是建立一个mixin风格的base class，这种base class允许derived classes继承单一特定能力。然后将这个base class转换为template，如此一来每个derived class将获得实体互异的class data复件。

```
template<typename T>
class NewHandlerSupport{
public:
    static std::new_handler set_new_handler(std::new_handler p) throw();
    static void* operator new(std::size_t size) throw(std::bad_alloc);
private:
    static std::new_handler currentHandler;
};

template<typename T>
std::new_handler    NewHandlerSupport<T>::set_new_handler(std::new_handler p) throw()
```

```
{
    std::new_handler oldHandler = currentHandler;
    currentHandler = p;
    return oldHandler;
}

template<typename T>
void* NewHandlerSupport<T>::operator new(std::size_t size)
{
    NewHandlerHolder h(std::set_new_handler(currentHandler));
    return ::operator new(size);
}
```

以下将每个currentHandler初始化为null

```
template<typename T>
std::new_handler NewHandlerSupport<T>::currentHandler = 0;
```

为Widget添加set_new_handler支持能力就轻而易举了：只要Widget继承自NewHandlerSupport<Widget>就好。

```
class Widget: public NewHandlerSupport<Widget>{
    …               //和先前一样，但不必声明set_new_handler或operator new
};
```

实际上使用template T只是希望， 继承自NewHandlerSupport的每个class，拥有实体互异的NewHandlerSupport复件（更确切的说是其static成员变量currentHandler）。类型参数只是用来区分不同的derived class。template机制会自动为每一个T（NewHandlerSupport赖以具现化的根据）生成一份currentHandler。

Widget继承自一个模板化的base class，而后者又以Widget作为类型参数：“怪异的循环模板模式”（curiously recurring template pattern；crtp）。像是do it for me.

1993年之前，operator new在无法分配足够内存时返回null。新一代的operator new 抛出bad_alloc异常。

```
class Widget{...};
Widget* pw1 = new Widget;
if (pw1 == 0)...                    //这个测试一定失败，因为如果分配失败，抛出bad_alloc
Widget* pw2 = new(std::nothrow) Widget;
if (pw2 == 0)...                    //这个测试可能成功，如果分配Widget失败，返回0
```

nothrow　new是个颇为局限的东西，因为它只适用于内存分配，后续的构造函数还是可能抛出异常。如果构造函数又new一些东西，没人强迫它再次适用nothrow new。

分类: c/c++

lidan
关注 - 0
粉丝 - 14
+加关注

0 0

(请您对文章做出评价)

« 上一篇：effective C++ 条款 48：认识template元编程
» 下一篇：effective C++ 条款 50：了解new和delete的合理替换时机

posted @ 2012-02-18 22:27 lidan 阅读(307) 评论(0) 编辑 收藏

刷新评论 刷新页面 返回顶部