C++析构函数详解

创建对象时系统会自动调用构造函数进行初始化工作，同样，销毁对象时系统也会自动调用一个函数来进行清理工作，例如释放分配的内存、关闭打开的文件等，这个函数就是析构函数。

析构函数（Destructor）也是一种特殊的成员函数，没有返回值，不需要程序员显式调用（程序员也没法显式调用），而是在销毁对象时自动执行。构造函数的名字和类名相同，而析构函数的名字是在类名前面加一个 ~ 符号。

注意：析构函数没有参数，不能被重载，因此一个类只能有一个析构函数。如果用户没有定义，编译器会自动生成一个默认的析构函数。

上节我们定义了一个 VLA 类来模拟变长数组，它使用一个构造函数为数组分配内存，这些内存在数组被销毁后不会自动释放，所以非常有必要再添加一个析构函数，专门用来释放已经分配的内存。请看下面的完整示例：

```
01.  #include <iostream>
02.  using namespace std;
03.
04.  class VLA{
05.  public:
06.      VLA(int len);  //构造函数
07.      ~VLA();  //析构函数
08.  public:
09.      void input();  //从控制台输入数组元素
10.      void show();  //显示数组元素
11.  private:
12.      int *at(int i);  //获取第i个元素的指针
13.  private:
14.      const int m_len;  //数组长度
15.      int *m_arr; //数组指针
16.      int *m_p;  //指向数组第i个元素的指针
17.  };
18.
19.  VLA::VLA(int len): m_len(len){  //使用初始化列表来给 m_len 赋值
20.      if(len > 0){ m_arr = new int[len];  /*分配内存*/ }
21.      else{ m_arr = NULL; }
22.  }
23.  VLA::~VLA(){
24.      delete[] m_arr;  //释放内存
25.  }
26.  void VLA::input(){
```

```
27.     for(int i=0; m_p=at(i); i++){ cin>>*at(i); }
28. }
29. void VLA::show(){
30.     for(int i=0; m_p=at(i); i++){
31.         if(i == m_len - 1){ cout<<*at(i)<<endl; }
32.         else{ cout<<*at(i)<<", "; }
33.     }
34. }
35. int * VLA::at(int i){
36.     if(!m_arr || i<0 || i>=m_len){ return NULL; }
37.     else{ return m_arr + i; }
38. }
39.
40. int main(){
41.     //创建一个有n个元素的数组（对象）
42.     int n;
43.     cout<<"Input array length: ";
44.     cin>>n;
45.     VLA *parr = new VLA(n);
46.     //输入数组元素
47.     cout<<"Input "<<n<<" numbers: ";
48.     parr -> input();
49.     //输出数组元素
50.     cout<<"Elements: ";
51.     parr -> show();
52.     //删除数组（对象）
53.     delete parr;
54.
55.     return 0;
56. }
```

运行结果：
Input array length: 5
Input 5 numbers: 99 23 45 10 100
Elements: 99, 23, 45, 10, 100

`~VLA()` 就是 VLA 类的析构函数，它的唯一作用就是在删除对象（第 53 行代码）后释放已经分配的内存。

函数名是标识符的一种，原则上标识符的命名中不允许出现 `~` 符号，在析构函数的名字中出现的 `~` 可以认为是一种特殊情况，目的是为了和构造函数的名字加以对比和区分。

注意：at() 函数只在类的内部使用，所以将它声明为 private 属性；m_len 变量不允许修改，所以用 const 进行了限制，这样就只能使用初始化列表来进行赋值。

C++ 中的 new 和 delete 分别用来分配和释放内存，它们与C语言中 malloc()、free() 最大的一个不同之处在于：用 new 分配内存时会调用构造函数，用 delete 释放内存时会调用析构函数。构造函数和析构函数对于类来说是不可或缺的，所以在C++中我们非常鼓励使用 new 和 delete。

析构函数的执行时机

析构函数在对象被销毁时调用，而对象的销毁时机与它所在的内存区域有关。不了解内存分区的读者请阅读《C语言内存精讲》专题。

在所有函数之外创建的对象是全局对象，它和全局变量类似，位于内存分区中的全局数据区，程序在结束执行时会调用这些对象的析构函数。

在函数内部创建的对象是局部对象，它和局部变量类似，位于栈区，函数执行结束时会调用这些对象的析构函数。

new 创建的对象位于堆区，通过 delete 删除时才会调用析构函数；如果没有 delete，析构函数就不会被执行。

下面的例子演示了析构函数的执行。

```
01. #include <iostream>
02. #include <string>
03. using namespace std;
04.
05. class Demo{
06. public:
07.     Demo(string s);
08.     ~Demo();
09. private:
10.     string m_s;
11. };
12. Demo::Demo(string s): m_s(s){ }
13. Demo::~Demo(){ cout<<m_s<<endl; }
14.
15. void func(){
16.     //局部对象
17.     Demo obj1("1");
18. }
19.
```

```
20.  //全局对象
21.  Demo obj2("2");
22.
23.  int main(){
24.      //局部对象
25.      Demo obj3("3");
26.      //new创建的对象
27.      Demo *pobj4 = new Demo("4");
28.      func();
29.      cout<<"main"<<endl;
30.
31.      return 0;
32.  }
```

运行结果:

1

main

3

2