

C++从入门到入土之类和对象II

Vect. 已于 2025-03-20 21:17:48 修改

一、六大默认成员函数

默认成员函数是**用户没有显式实现**，编译器自动生成的成员函数。

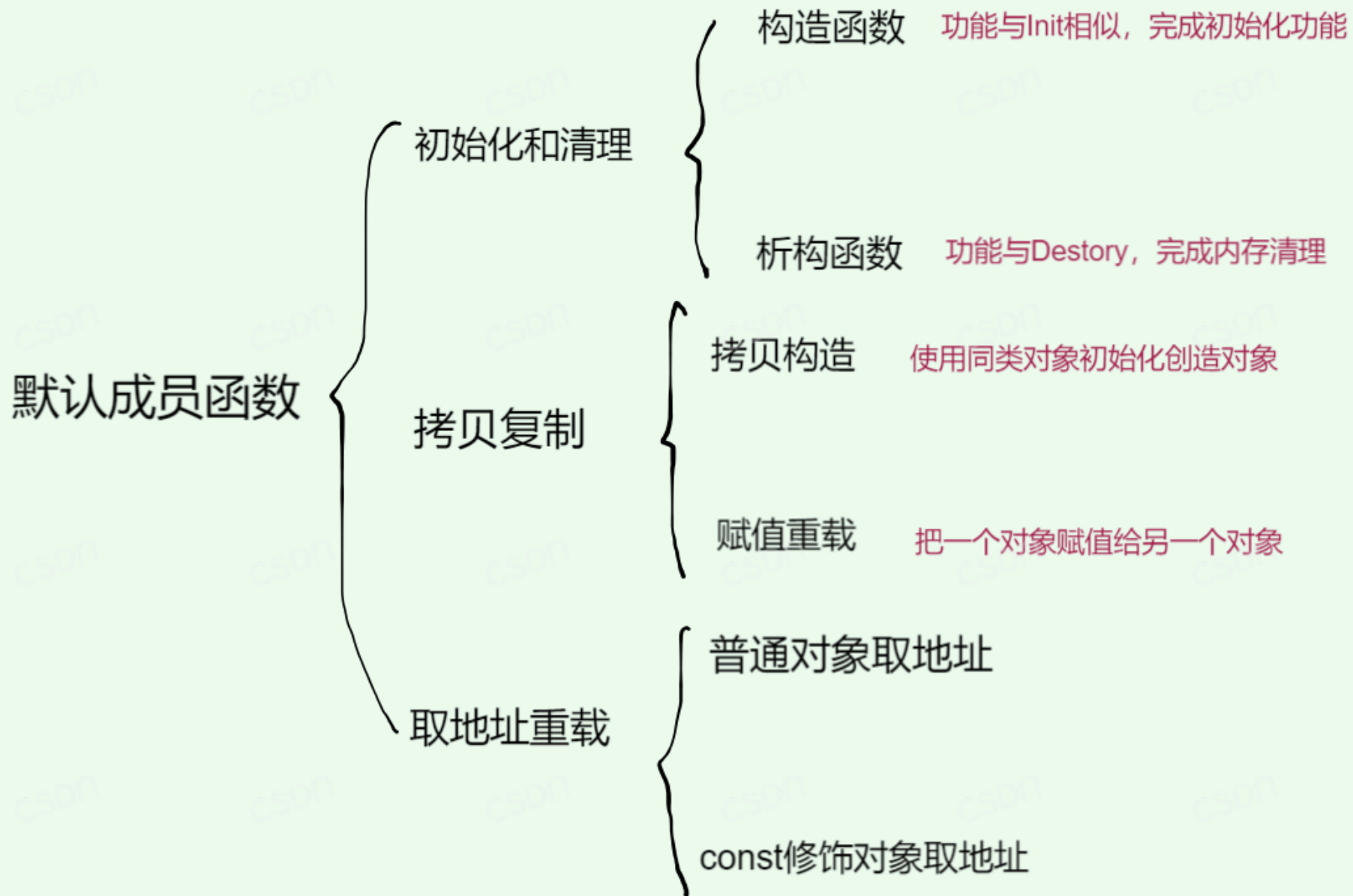
一个类，我们在不写的情况下，编译器会默认生成六个默认成员函数

内容来源：csdn.net

作者昵称：Vect.

原文链接：<https://blog.csdn.net/ting1fengyu1/article/details/146404067>

作者主页：<https://blog.csdn.net/ting1fengyu1>



本文详细介绍构造函数和析构函数

二、构造函数

内容来源: csdn.net

作者昵称: Vect.

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/146404067>

作者主页: <https://blog.csdn.net/ting1fengyu1>

构造函数虽名为构造函数，但是这个函数并不开辟空间创建对象（经常使用的局部对象是栈帧创建的）

构造函数的功能是：在对象实例化时初始化对象，类似于我们以前写的 `Init()` 函数

C++引入**构造函数**，我们也就可以代替 `Init()` 函数了

构造函数的特点：

- 函数值与类名相同

```
1 class Info{
2 public:
3 // 构造函数
4     Info(){
5         //...
6     }
7 private:
8 //...
9 };
```

AI写代码

- **无返回值**（啥都不需要给，void也不用）
- **对象实例化时（创建对象）**系统会自动调用对应的构造函数
- **支持重载**，可以根据参数不同定义多个构造函数
- 如果没有**显式定义**构造函数，C++的编译器会自动生成一个**无参的默认构造函数**，一旦显式定义就不会生成

构造函数的类型

1. 默认构造函数

默认构造函数分为**无参构造函数**、**编译器自动生成的构造函数**、**全缺省构造函数**

内容来源：csdn.net

作者昵称：Vect

原文链接：<https://blog.csdn.net/ting1fengyu1/article/details/146404067>

作者主页：<https://blog.csdn.net/ting1fengyu1>

这三个函数有且只能存在一个:

```
138 class Info {
139 public:
140     // 无参的默认构造函数
141     Info() {
142         _name = "UnKnow";
143         _age = 0;
144     }
145     // 全缺省的默认构造函数
146     Info(string name = "KUNKUN", int age = 18) {
147         _name = name;
148         _age = age;
149     }
150
151     void Print() {
152         cout << " 默认构造函数被调用" << endl;
153         cout << _name << " " << _age << endl;
154     }
155 private:
156     string _name;
157     int _age;
158 };
159
160 int main() {
161     Info I1;
162     I1.Print();
163 }
```

构成函数重载，但是调用不明确，存在歧义
两种默认构造函数不能同时存在

错误列表

整个解决方案 | 错误 2 | 警告 0 | 展示 1 个消息中的 0 个 | 生成 + IntelliSense

代码	说明
E0339	类 "Info" 包含多个默认构造函数
C2668	"Info::Info": 对重载函数的调用不明确

内容来源: csdn.net

作者昵称: Vect

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/146404067>

作者主页: <https://blog.csdn.net/ting1fengyu1>

而**无参构造函数**和**全缺省构造函数**虽能构成函数重载，但是会产生调用歧义

总结一下：**默认构造函数**是不用传实参的构造函数

```
1 class Info {
2 public:
3     // 无参的默认构造函数
4     Info() {
5         _name = "UnKnow";
6         _age = 0;
7     }
8
9     void Print() {
10         cout << " 默认构造函数被调用" << endl;
11         cout << _name << "  " << _age << endl;
12     }
13 private:
14     string _name;
15     int _age;
16 };
17
18 int main() {
19     Info I1;
20     I1.Print();
21 }
```



AI写代码

内容来源: [csdn.net](https://blog.csdn.net/ting1fengyu1)

作者昵称: Vect.

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/146404067>

作者主页: <https://blog.csdn.net/ting1fengyu1>



2. 带参数的构造函数

```
1 class Info {
2 public:
3     // 带参的构造函数
4     Info(string name, int age) {
5         _name = name;
6         _age = age;
7     }
8
9     void Print() {
10 }
```

内容来源: csdn.net

作者昵称: Vect.

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/146404067>

作者主页: <https://blog.csdn.net/ting1fengyu1>

```
10     cout << " 默认构造函数被调用" << endl;
11     cout << _name << "  " << _age << endl;
12 }
13 private:
14     string _name;
15     int _age;
16 };
17
18 int main() {
19     // 调用带参数的构造函数
20     Info I2("kunkun", 18);
21     I2.Print();
22 }
23
```



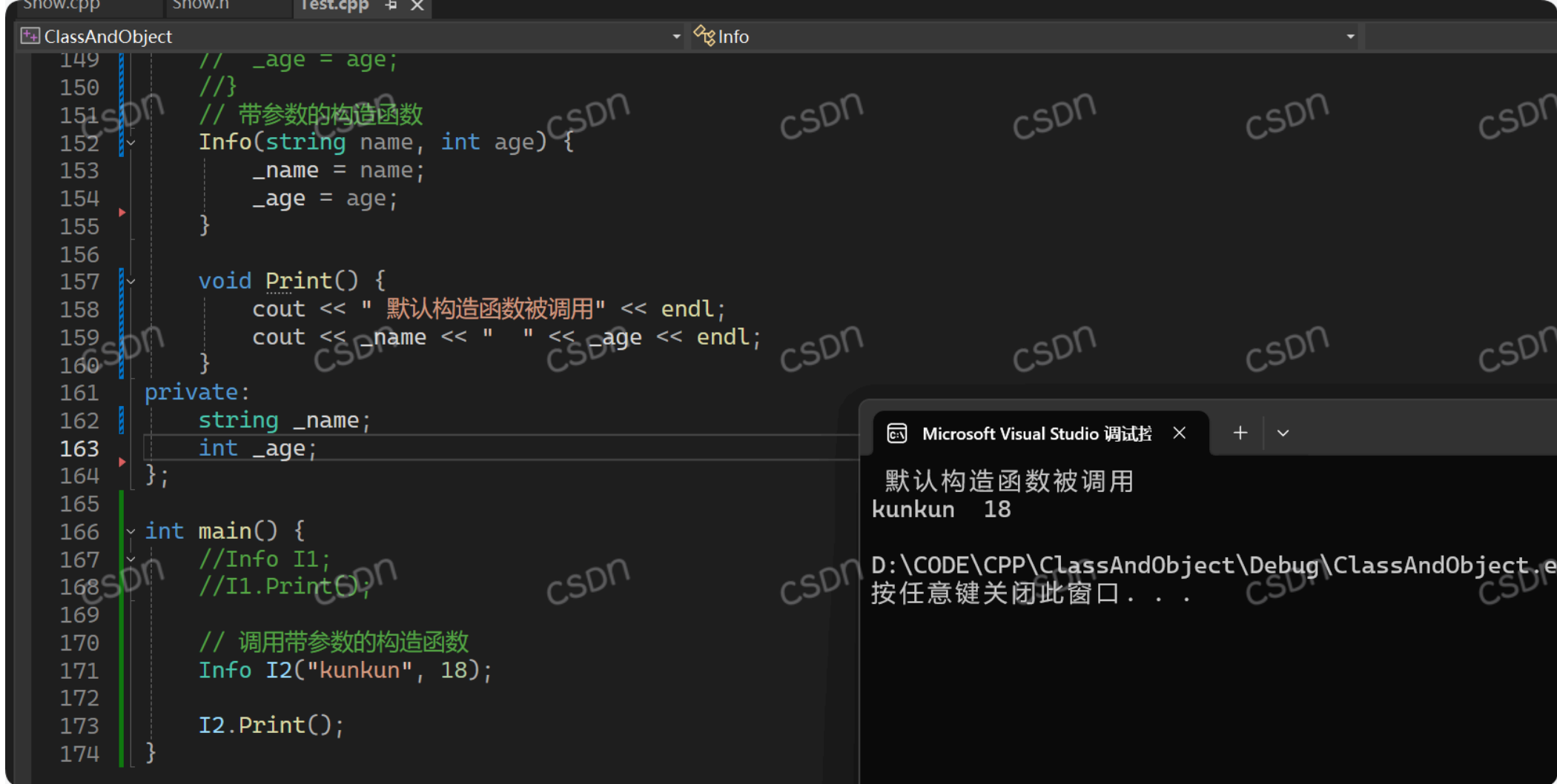
AI写代码

内容来源: [csdn.net](https://blog.csdn.net/ting1fengyu1)

作者昵称: Vect

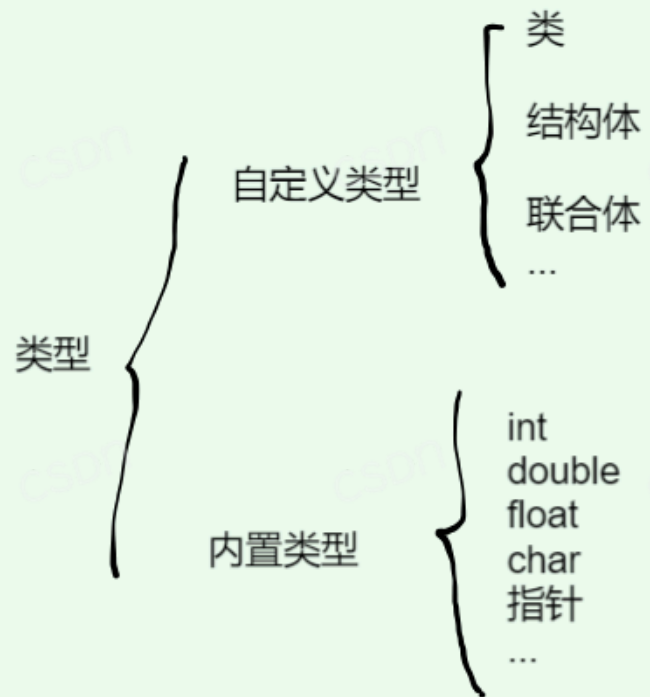
原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/146404067>

作者主页: <https://blog.csdn.net/ting1fengyu1>



编译器自动生成的默认构造函数深度剖析

我们如果没有显式定义构造函数，编译器自动生成的构造函数会将对象初始化成什么呢？
类型的分类：



内容来源: [csdn.net](https://blog.csdn.net/ting1fengyu1)

作者昵称: Vect.

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/146404067>

作者主页: <https://blog.csdn.net/ting1fengyu1>

- 内置类型：没有规定要处理（可处理可不处理，看编译器类型）

```
183
184 ~ class Date {
185 public:
186     /*Date(int year, int month, int day) {
187         _year = year;
188         _month = month;
189         _day = day;
190     }*/
191
192     void Print() {
193         cout << "构造函数成功调用" << endl;
194         cout << _year << "-" << _month << "-" << _day << endl;
195     }
196
197 private:
198     int _year;
199     int _month;
200     int _day;
201     // 添加自定义类型
202     Time _t;
203 };
204
205 int main() {
206     Date D1;
207     D1.Print();
208
209
210
211     return 0;
212 }
```

不写构造函数，编译器自动生成默认构造函数
对内置类型不做处理

Microsoft Visual Studio 调试 × + ▾

构造函数成功调用
-858993460--858993460--858993460

D:\CODE\CPP\ClassAndObject\Debug\ClassAndObject.exe
按任意键关闭此窗口 . . .|

- 自定义类型：调用自定义类型对象的默认构造函数，本质是不断套娃，深挖!!!

内容来源: csdn.net

作者昵称: Vect

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/146404067>

作者主页: <https://blog.csdn.net/ting1fengyu1>

ClassAndObject

(全局范围)

main()

```
176 class Time {
177 public:
178     Time() {
179         cout << "构造函数调用成功" << endl;
180         _hour = 0;
181         _minute = 0;
182         _second = 0;
183     }
184 private:
185     int _hour;
186     int _minute;
187     int _second;
188 };
189
190 class Date {
191 public:
192     /*Date(int year, int month, int day) {
193         _year = year;
194         _month = month;
195         _day = day;
196     }*/
197
198     void Print() {
199         cout << "构造函数成功调用" << endl;
200         cout << _year << "-" << _month << "-" << _day << endl;
201     }
202
203 private:
204     int _year;
205     int _month;
206     int _day;
207     // 添加自定义类型
208     Time _t;
209 };
210
211 int main() {
212     Date D1;
213     D1.Print();
214
215     return 0;
216 }
```

监视 1

搜索(Ctrl+E)

搜索深度: 3

lab

名称	值	类型
D1	{_year=-858993460 _month=-858993460 _day=-858993460 ...}	Date
_year	-858993460	int
_month	-858993460	int
_day	-858993460	int
_t	{ hour=0 _minute=0 _second=0 }	Time
添加要监视的项		

分析一下：

D1这个对象中有三个内置类型成员变量和一个自定义类型成员变量，不写构造函数，首先自动生成 `Date()` 的默认构造函数，到 `private` 中发现三个内置类型，则不做处理，有个自定义类型 `_t`，则去调用 `Time()` 的构造函数，发现有构造函数，则按照构造函数初始化命令初始化，如果 `Time()` 没有构造函数呢？那么 `_hour` `_minute` `_second` 也是内置类型，不做处理

```

175
176 class Time {
177 public:
178     //Time() {
179     //    cout << "构造函数调用成功" << endl;
180     //    _hour = 0;
181     //    _minute = 0;
182     //    _second = 0;
183     //}
184 private:
185     int _hour;
186     int _minute;
187     int _second;
188 };
189
190 class Date {
191 public:
192     /*Date(int year, int month, int day) {
193     _year = year;
194     _month = month;
195     _day = day;
196     }*/
197
198     void Print() {
199         cout << "构造函数成功调用" << endl;
200         cout << _year << "-" << _month << "-" << _day << endl;
201     }
202
203 private:
204     int _year;
205     int _month;
206     int _day;
207     // 添加自定义类型
208     Time _t;
209 };
210
211 int main() {
212     Date D1;
213     D1.Print(); 已用时间 <= 1ms
214

```

监视 1

搜索(Ctrl+E)



搜索深度: 3



名称

值

▲ D1

{_year=-858993460 _month=-858993460 _day=-858993460 ...}

▢ _year

-858993460

▢ _month

-858993460

▢ _day

-858993460

▢ t

{_hour=-858993460 _minute=-858993460 _second=-858993...

添加要监视的项

注意: 没有默认构造函数会报错

内容来源: csdn.net

作者昵称: Vect

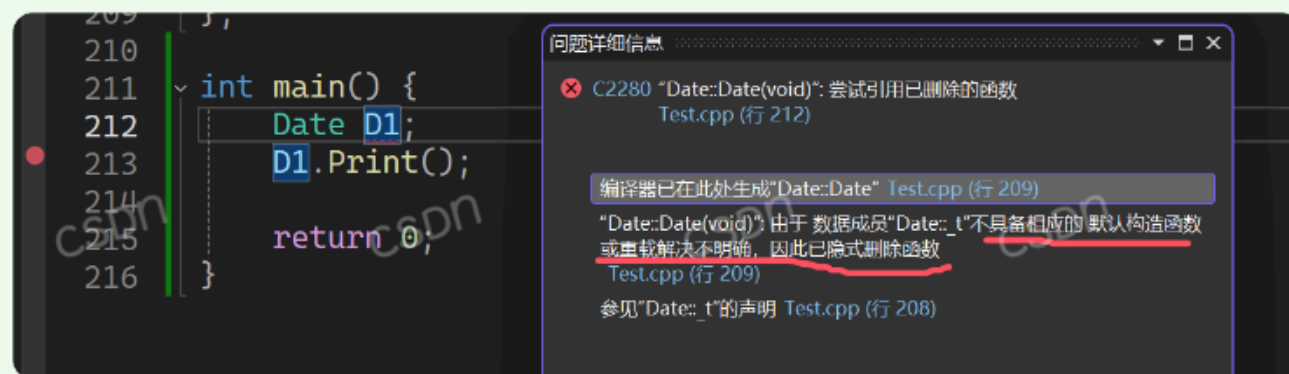
原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/146404067>

作者主页: <https://blog.csdn.net/ting1fengyu1>

```

class Time {
public:
    Time(int hour) {
        cout << "构造函数调用成功" << endl;
        _hour = hour;
        _minute = 0;
        _second = 0;
    }
private:
    int _hour;
    int _minute;
    int _second;
};

```



三、析构函数

析构函数与构造函数功能相反，析构函数不是销毁对象，比如局部对象存在栈中，函数栈帧结束就自动销毁释放内存。

析构函数的功能是在对象销毁时完成对象中资源的清理释放

析构函数的功能类似于 Destroy()，析构函数就可以完美替代 Destroy() 了

析构函数的特点

- 无参数无返回值，与构造函数类似
- 函数名与类名相同，在类名前加字符“~” eg: Name()
- 一个类只能有一个析构函数，所以析构函数不能重载如果没有显式定义，系统会自动生成默认的析构函数

内容来源: csdn.net

作者昵称: Vect

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/146404067>

作者主页: <https://blog.csdn.net/ting1fengyu1>

- 对象生命周期结束，会**自动调用析构函数**
- 与构造函数相同，**编译器自动生成的析构函数对内置类型不做处理，对自定义类型则会调用它的析构函数**
注意：我们显式写析构函数，自定义类型成员会调用它的析构函数，换句话说，**自定义类型成员无论什么情况下都会调用析构函数**

析构函数的语法

```
1 class Test {
2     public:
3         Test() {
4             cout << "构造函数调用成功" << endl;
5         }
6         ~Test() {
7             cout << "析构函数调用成功" << endl;
8         }
9     private:
10 };
11
12 int main() {
13     cout << "程序开始运行" << endl;
14     {
15         Test T; // 构造函数被调用
16     } // T生命周期结束
17     cout << "程序运行结束";
18 }
```



AI写代码

内容来源: [csdn.net](https://blog.csdn.net/ting1fengyu1)

作者昵称: Vect.

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/146404067>

作者主页: <https://blog.csdn.net/ting1fengyu1>

```

217
218 // 析构函数
219 class Test {
220 public:
221     Test() {
222         cout << "构造函数调用成功" << endl;
223     }
224     ~Test() {
225         cout << "析构函数调用成功" << endl;
226     }
227 private:
228 };
229
230 int main() {
231     cout << "程序开始运行" << endl;
232     {
233         Test T; // 构造函数被调用
234     } // T生命周期结束
235     cout << "程序运行结束";
236 }

```

Microsoft Visual Studio 调试器

程序开始运行
 构造函数调用成功
 析构函数调用成功
 程序运行结束
 D:\CODE\CPP\ClassAndObject\Debug\ClassAndObject.exe
 按任意键关闭此窗口 . . .

总结:

- 有资源需要手动清理, 需要写析构函数
- 有两种场景不需要写析构函数, 利用默认生成的即可:
 1. 没有资源需要清理, 例如: `Date()` 日期列表全是局部成员
 2. 内置类型没有资源需要清理, 剩下的全是自定义类型成员, 且这些类有正确的析构函数 eg:

```

1 public:
2     Engine() { std::cout << "Engine created.\n"; }

```

内容来源: [csdn.net](https://blog.csdn.net/ting1fengyu1)

作者昵称: Vect

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/146404067>

作者主页: <https://blog.csdn.net/ting1fengyu1>

```
3     ~Engine() { std::cout << "Engine destroyed.\n"; }
4 };
5
6 class Car {
7 public:
8     Engine engine;
9     int speed;
10 }; // Car的析构函数不需要手写
11
```

AI写代码

内容来源: [csdn.net](https://blog.csdn.net/ting1fengyu1)

作者昵称: Vect.

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/146404067>

作者主页: <https://blog.csdn.net/ting1fengyu1>