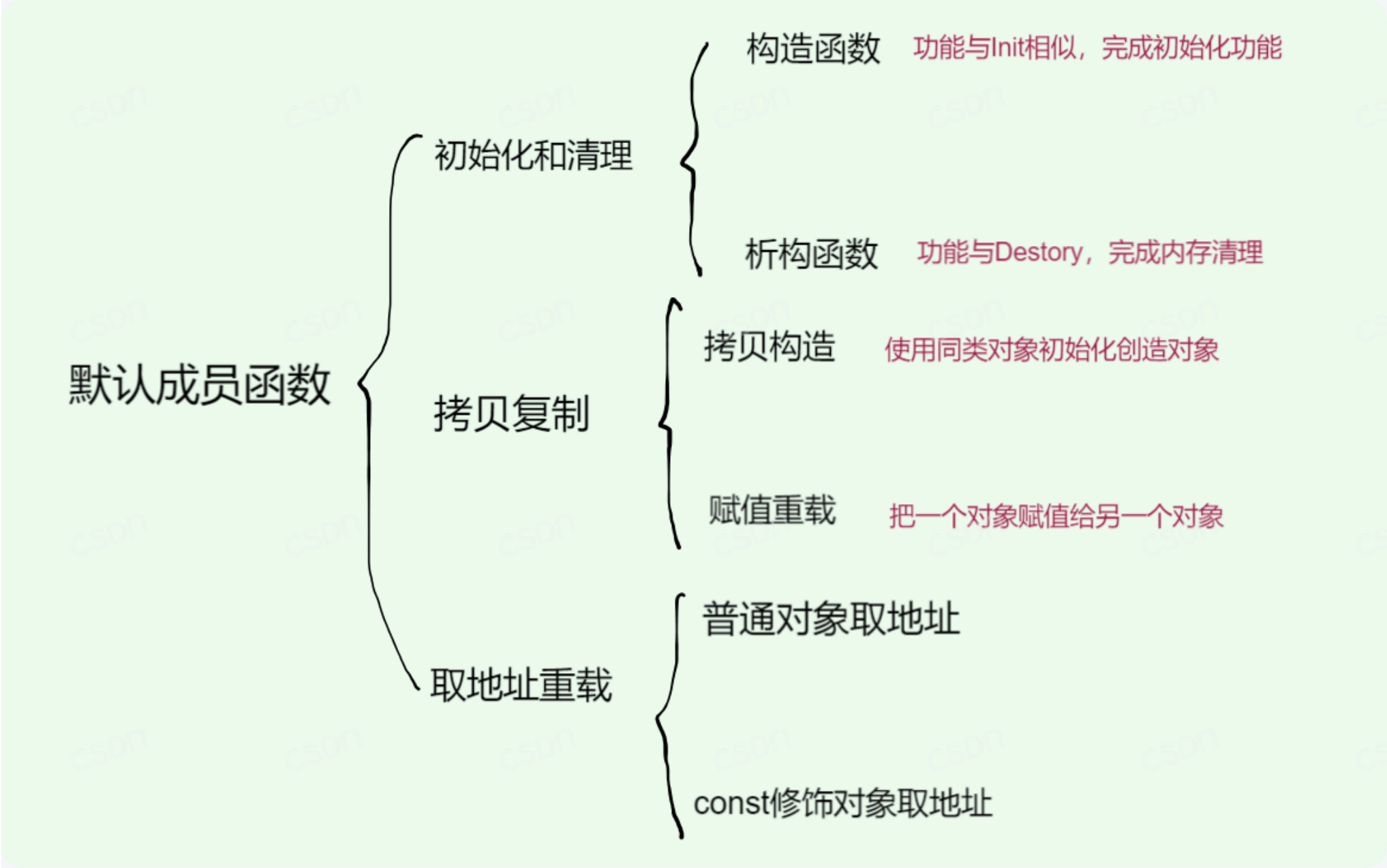
CPP从入门到入土之类和对象Ⅱ

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



本文详细介绍构造函数和析构函数

二、构造函数

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
CPP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**

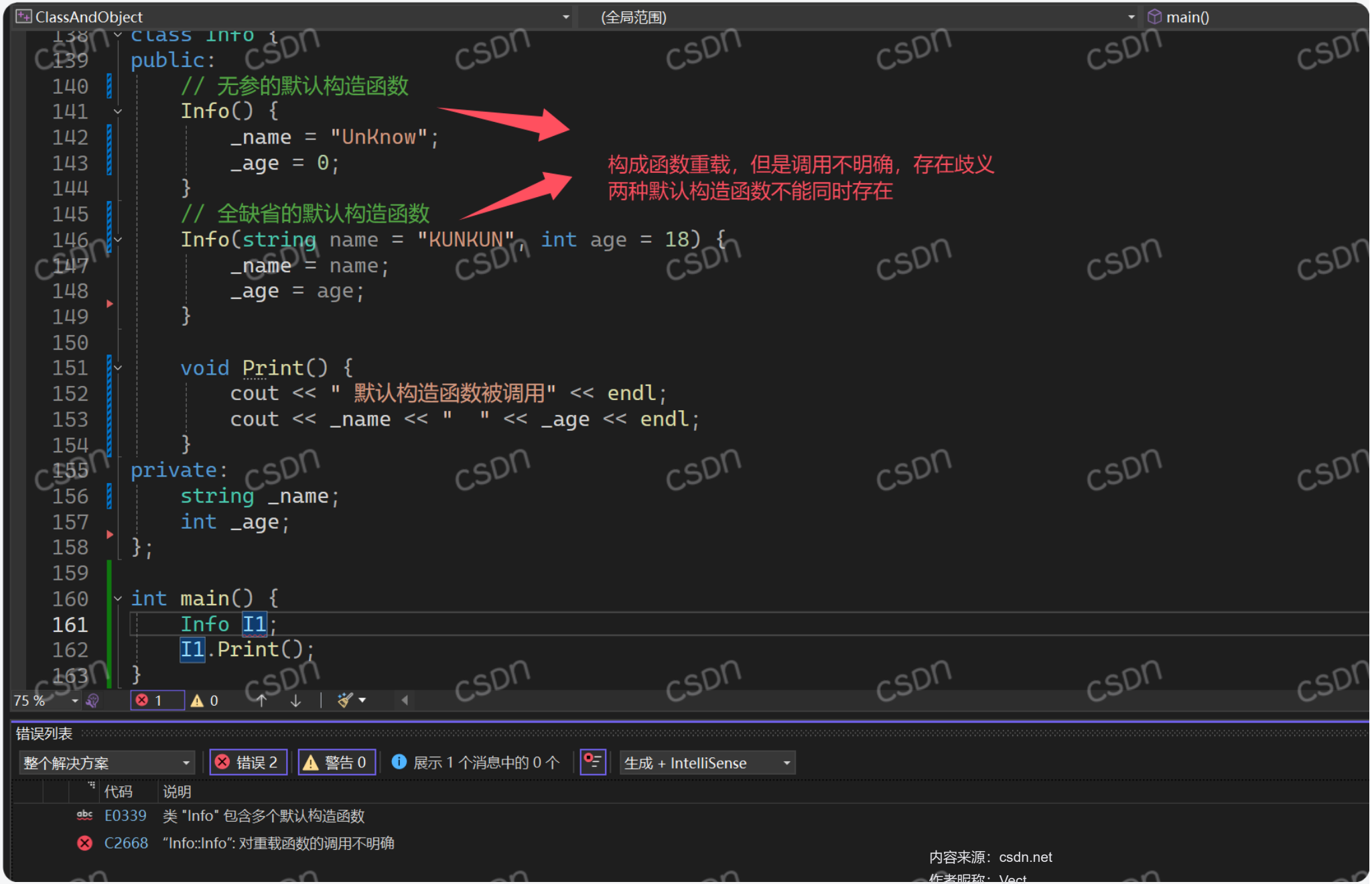
```
class Info{
public:
// 构造函数
    Info(){
    //...
    }
private:
//...
};
```
AI写代码

- **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，CPP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

这三个函数有且只能存在一个:




```
class Info {
public:
    // 无参的默认构造函数
    Info() {
        _name = "UnKnow";
        _age = 0;
    }
    // 全缺省的默认构造函数
    Info(string name = "KUNKUN", int age = 18) {
        _name = name;
        _age = age;
    }

    void Print() {
        cout << " 默认构造函数被调用" << endl;
        cout << _name << "   " << _age << endl;
    }
private:
    string _name;
    int _age;
};

int main() {
    Info I1;
    I1.Print();
}
```

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

```
class Info {
public:
    // 无参的默认构造函数
    Info() {
        _name = "UnKnow";
        _age = 0;
    }

    void Print() {
        cout << " 默认构造函数被调用" << endl;
        cout << _name << "  " << _age << endl;
    }
private:
    string _name;
    int _age;
};

int main() {
    Info I1;
    I1.Print();
}
```

AI写代码

2. **带参数的构造函数**

```
class Info {
public:
    // 带参的构造函数
    Info(string name, int age) {
        _name = name;
        _age = age;
    }

    void Print() {
```

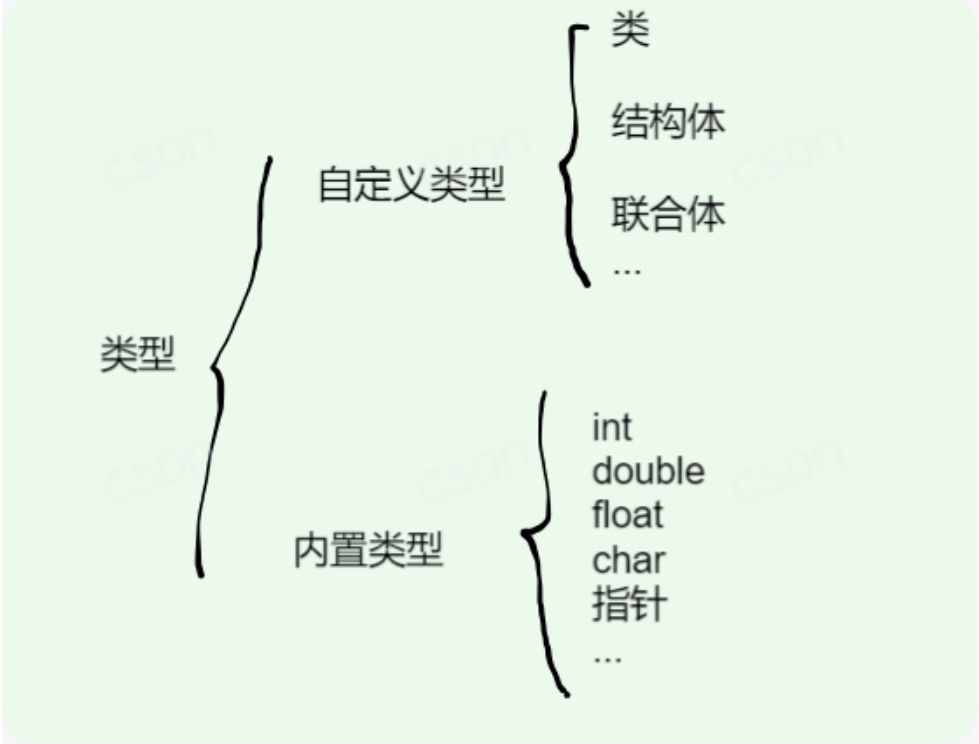
```
        cout << " 默认构造函数被调用" << endl;
        cout << _name << "  " << _age << endl;
    }
private:
    string _name;
    int _age;
};

int main() {
    // 调用带参数的构造函数
    Info I2("kunkun", 18);
    I2.Print();
}
```

AI写代码

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：



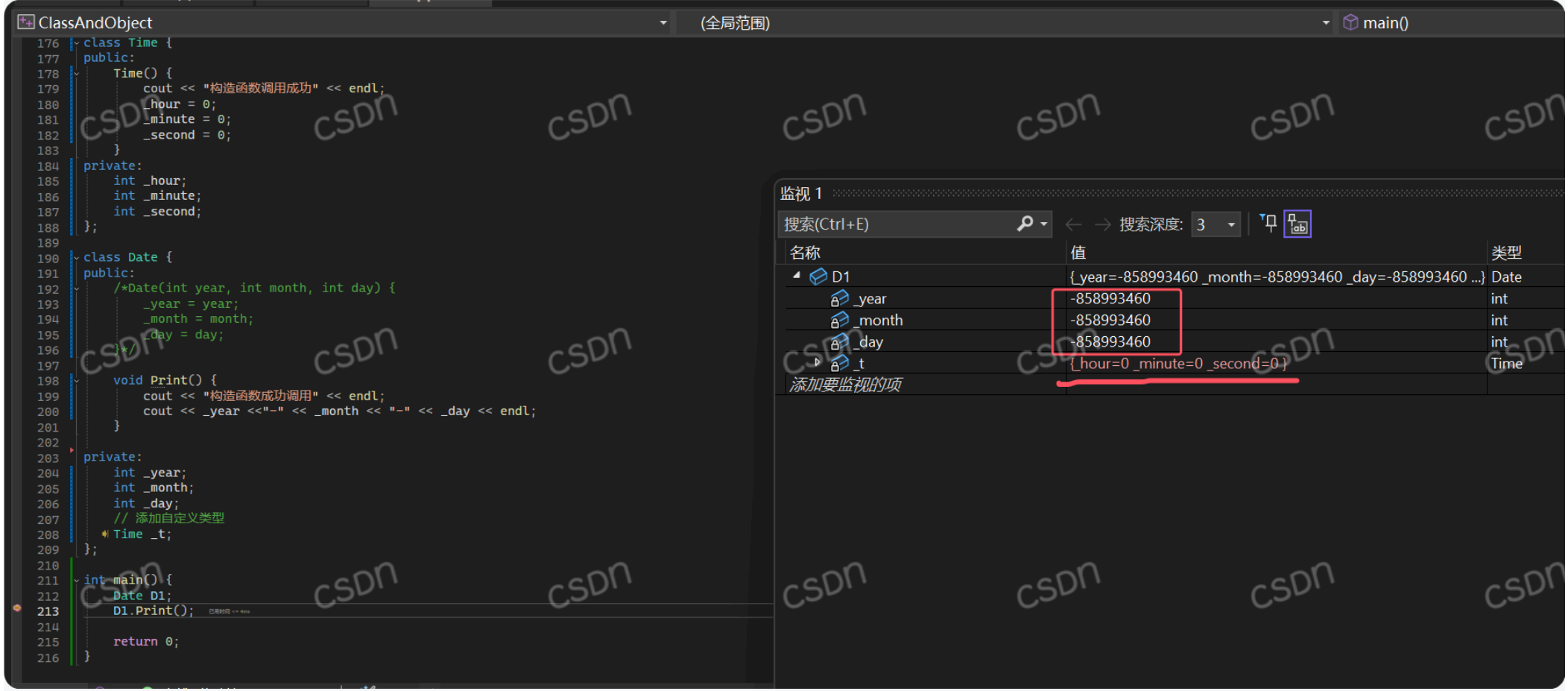
- 内置类型：没有规定要处理（可处理可不处理，看编译器类型）

```cpp
class Date {
public:
    /*Date(int year, int month, int day) {
        _year = year;
        _month = month;
        _day = day;
    }*/

    void Print() {
        cout << "构造函数成功调用" << endl;
        cout << _year <<"-" << _month <<"-" << _day << endl;
    }

private:
    int _year;
    int _month;
    int _day;
    // 添加自定义类型
    Time _t;
};

int main() {
    Date D1;
    D1.Print();



    return 0;
}
```

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 调试控
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！！！



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

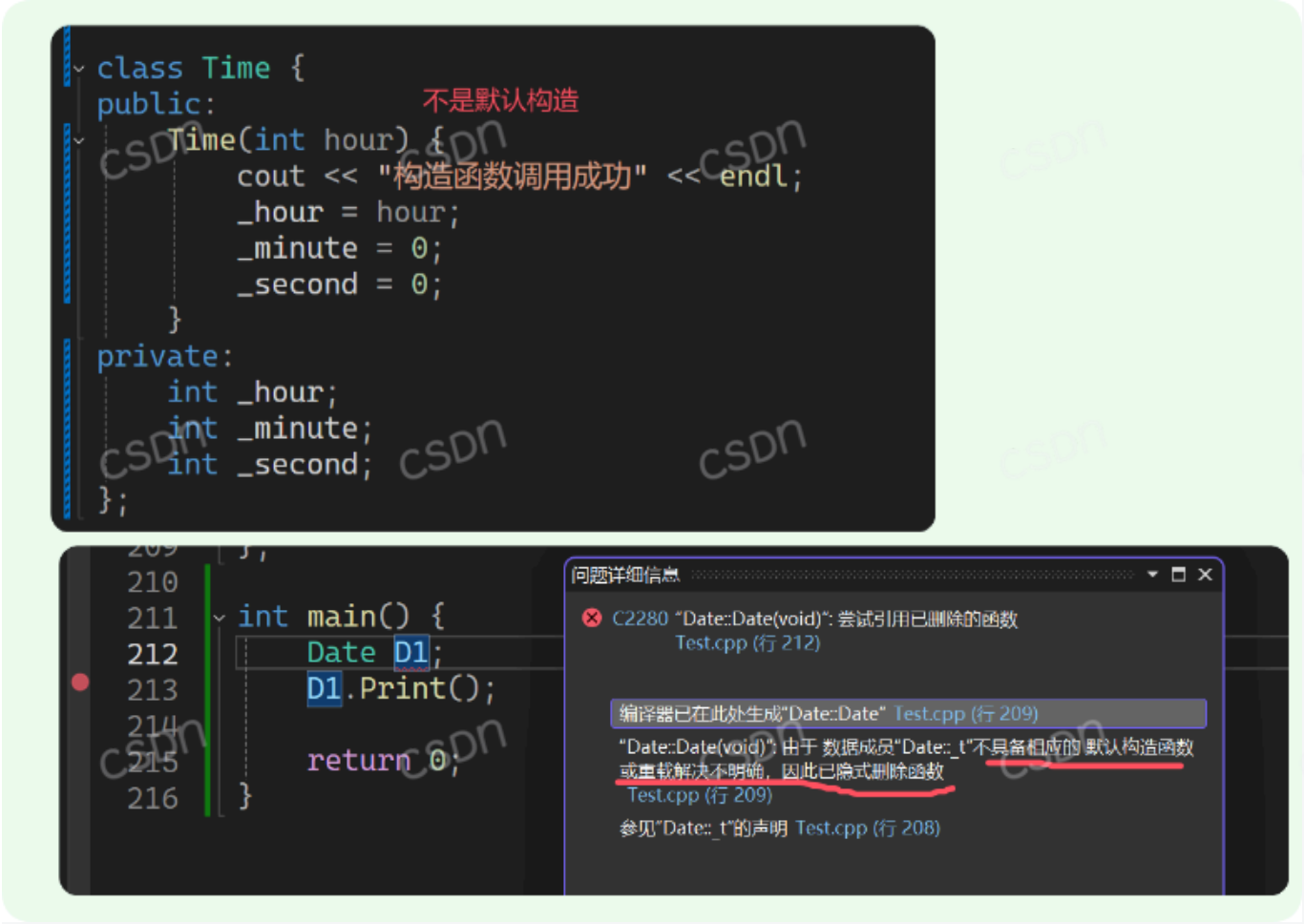
```cpp
class Time {
public:
    //Time() {
    //  cout << "构造函数调用成功" << endl;
    //  _hour = 0;
    //  _minute = 0;
    //  _second = 0;
    //}
private:
    int _hour;
    int _minute;
    int _second;
};

class Date {
public:
    /*Date(int year, int month, int day) {
        _year = year;
        _month = month;
        _day = day;
    }*/

    void Print() {
        cout << "构造函数成功调用" << endl;
        cout << _year <<"-" << _month << "-" << _day << endl;
    }

private:
    int _year;
    int _month;
    int _day;
    // 添加自定义类型
    Time _t;
};

int main() {
    Date D1;
    D1.Print();
```

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



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

- 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

```
class Test {
public:
    Test() {
        cout << "构造函数调用成功" << endl;
    }
    ~Test() {
        cout << "析构函数调用成功" << endl;
    }
private:
};

int main() {
    cout << "程序开始运行" << endl;
    {
        Test T;// 构造函数被调用
    }// T生命周期结束
    cout << "程序运行结束";
}
```

AI写代码

```cpp
// 析构函数
class Test {
public:
    Test() {
        cout << "构造函数调用成功" << endl;
    }
    ~Test() {
        cout << "析构函数调用成功" << endl;
    }
private:
};

int main() {
    cout << "程序开始运行" << endl;
    {
        Test T;// 构造函数被调用
    }// T生命周期结束
    cout << "程序运行结束";
}
```

```
Microsoft Visual Studio 调试控
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\ClassAnd
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:

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

```
    ~Engine() { std::cout << "Engine destroyed.\n"; }
};

class Car {
public:
    Engine engine;
    int speed;
}; // Car的析构函数不需要手写

```

AI写代码