

2.2C++ 类

类的定义与实现

类是对某一类对象的抽象，对象是类的具体实例。

C++将对象的属性抽象为 **数据成员**，将对象的行为抽象为 **成员函数**。数据成员又称 **成员变量**，成员函数又称为 **方法**。

C++类定义的基本形式：

```
1  class <类名>
2  {
3      private:
4          <私有数据成员和私有成员函数的声明列表>;
5      public:
6          <公有数据成员和公有成员函数的声明列表>;
7      protected;
8          <保护数据成员和保护成员函数的声明列表>;
9  };
```

例子：类的定义

```
1  // CTime.hpp
2  class CTime
3  {
4      private: //仅可内部使用
5          int hour;
6          int minute;
7          int second;
8      public:
9          void setTime(int h, int m, int s);
10         void showTime();
11     };
```

```
1  // CTime.cpp
2  void CTime::setTime(int h, int m, int s)
3  {
4      hour CTime::setTime(int h, int m, int s)
5      {
6          hour = (h >= 0 && h < 24) ? h : 0;
7          minute = (m >= 0 && m < 60) ? m : 0;
8          second = (s >= 0 && s < 60) ? s : 0;
9      }
10
11 void CTime::showTime()
12 {
13     cout << hour << ":" << minute << ":" << second;
14     cout << endl;
15 }
```

```
1  // main
2  void eg3_1()
```

```

3      {
4          CTime t1, t2, *pTime = &t1;    //用CTime类定义了两个对象和一个指针
5
6          t1.setTime(8, 10, 1);
7          t2.setTime(20, 30, 30);
8          t1.showTime(); //对象. 方法
9          t2.showTime();
10         pTime->showTime();
11     }
12
13     int main()
14     {
15         eg3_1();
16         return 0;
17     }

```

建议使用方法

```

1     #include<bits/stdc++.h>
2     using namespace std;
3
4     // CTime.hpp
5     class CTime
6     {
7     private:
8         int hour;
9         int minute;
10        int second;
11    public:
12        void setTime(int hour = 0, int minute = 0, int second = 0);
13        CTime(int hour = 0, int minute = 0, int second = 0);
14
15        friend ostream& operator<<(ostream& os, const CTime& time)
16        {
17            os << setw(2) << setfill('0') << time.hour << ":";
18            os << setw(2) << setfill('0') << time.minute << ":";
19            os << setw(2) << setfill('0') << time.second;
20            return os;
21        }
22    };
23
24    void CTime::setTime(int hour, int minute, int second)
25    {
26        this->hour = (hour >= 0 && hour < 24) ? hour : 0;
27        this->minute = (minute >= 0 && minute < 60) ? minute : 0;
28        this->second = (second >= 0 && second < 60) ? second : 0;
29    }
30
31    CTime::CTime(int hour, int minute, int second)
32    {
33        setTime(hour, minute, second);
34    }
35
36    void eg3_1()
37    {
38        CTime t1(12, 34, 56), t2(23, 59, 59);;

```

```

39     CTime *pTime = &t1;
40
41     cout << "t1 = " << t1 << endl;
42     cout << "t2 = " << t2 << endl;
43     cout << "pTime = " << *pTime << endl;
44 }
45
46 int main()
47 {
48     eg3_1();
49     return 0;
50 }

```

构造函数和析构函数

- 如何 **初始化** 成员变量？
 - 不能在类定义时进行，为什么？
 - 何时进行？ ==> 对象创建时
- 成员函数的执行 **时机**
 - 一般时机：被调用时
 - 对象创建时： **构造函数()**
 - 对象释放时：析构函数
 - 拷贝（赋值、实参传递）时：拷贝构造函数
 -

构造函数和析构函数

- 构造函数的名称与 **类名相同**，析构函数的名称必须在 **类名前加上"~"** 符号。
- 构造函数和析构函数不能指定任何返回值类型，包括void返回类型。构造函数可以有参数，析构函数不能有参数。
- 若没有定义构造函数和析构函数，系统会 **自动** 为类 **添加** 默认的构造函数和析构函数（不带参数，函数体为空）。

例子：构造函数和析构函数

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4
5  // CTime.hpp
6  class CTime
7  {
8  private:
9      int hour;
10     int minute;
11     int second;
12
13 public:
14     void setTime(int hour = 0, int minute = 0, int second = 0);
15
16     CTime(int hour = 0, int minute = 0, int second = 0);

```

```

17     ~CTime();
18
19
20
21     friend ostream &operator<<(ostream &os, const CTime &time)
22     {
23         os << setw(2) << setfill('0') << time.hour << ":";
24         os << setw(2) << setfill('0') << time.minute << ":";
25         os << setw(2) << setfill('0') << time.second;
26         return os;
27     }
28 };
29
30
31 void CTime::setTime(int h, int m, int s)
32 {
33     hour = (h>=0 && h<24) ? h:0;
34     minute = (m>=0 && m<60) ? m:0;
35     second = (s>=0 && s<60) ? s:0;
36 }
37
38
39 CTime::CTime(int hour, int minute, int second)
40 {
41     this->setTime(hour, minute, second);
42     cout << *this << ", constructor" << endl;
43 }
44
45
46
47 CTime::~CTime()
48 {
49     cout << *this << ", destructor" << endl;
50 }
51
52
53
54
55 int main()
56 {
57
58     CTime t1(12, 20), t2(23, 30);
59     CTime* pTime = &t1;
60
61
62     cout << t1 << endl;
63     cout << t2 << endl;
64     cout << *pTime << endl;
65     return 0;
66 }
67

```

例子：构造函数的2种格式

```

1 CTime::CTime(int hour, int minute, int second)
2 {
3     this->hour = (hour>=0 && hour<24) ? hour:0;
4     this->minute = (minute>=0 && minute<60) ? minute:0;
5     this->second = (second>=0 && second<60) ? second:0;
6     cout << *this << ", constructor" << endl;
7 }

```

V2

```

1 CTime::CTime(int hour, int minute, int second):hour(hour), minute(minute), second(second)
2 {}

```

拷贝构造函数

- 是否可以用对象初始化对象？

```

1 int x = 0;
2 int y = x;
3
4 CTime t1(8, 10, 1);
5 CTime t2 = t1; //?

```

- 拷贝构造函数 (copyconstructor)

```

1 CTime(CTime& t1);
2 CTime::CTime(CTime& t1)
3 {
4     this->hour = t1.hour;
5     this->minute = t1.minute;
6     this->second = t1.second;
7 }

```

拷贝构造函数

- 若未定义拷贝构造函数，编译系统 **自动** 为类 **添加** 一个默认的拷贝构造函数。
- 默认拷贝构造函数采取 **浅拷贝**。
- 浅拷贝
 - 只进行 **基本数据类型** 的拷贝。
 - 当数据成员为指针类型时，存在潜在 **危险**，即两个对象的指针指向同一内存区域。
- 调用时机：
 - 用对象初始化对象。
 - 函数参数传递（此时参数为一个对象）

例子：构造、析构、拷贝构造函数

```

1 void demo_copy_constructor(CTime t1);
2 void eg3_5()
3 {
4     CTime t1(8);

```

```

5      CTime t2 = t1;
6
7      t2.setTime(11, 30);
8      cout << "t2 = " << t2 << endl;
9      demo_copy_constructor(t2);
10 }
11 void demo_copy_constructor(CTime t1)
12 {
13     cout << t1 << ", demo_copy_constructo..."<< endl;
14 }

```

问题：

- 不初始化，可以吗？
 - 可以
- 不定义构造或析构函数，可以吗？
 - 可以，系统会自动添加默认构造函数和激活函数
- 没有拷贝构造函数，会发生什么？
 - 采用浅拷贝
- 默认拷贝构造函数的风险到底是什么？
 - 浅拷贝存在风险
- C++的构造函数和析构函数的设计，还可以优化吗？
 - 将构造函数和析构函数定义成和类名无关的

例子：没有构造、析构函数

```

1  class CTime
2  {
3      private:
4          int hour;
5          int minute;
6          int second;
7      public:
8          void setTime(int hour = 0, int minute = 0, int second = 0)
9              //省略输出流重载...
10 };
11 //省略setTime()函数的定义...
12
13 //没有构造、解析函数
14 void eg3_6_1()
15 {
16     using namespace N_CTIME_1;
17     CTime t1, t2;
18     cout << "t1 = " << t1 << endl;
19     cout << "t2 = " << t2 << endl;
20 }

```

```
t1 = -272632624:32766:80758
t2 = 80800:01:-1732163744
```

系统自动加上的构造函数

```
class CTime {
private:
    int hour;
    int minute;
    int second;
public:
    void setTime(int hour=0, int minute=0, int second=0);
    CTime(); //系统自动添加的默认构造函数（定义）
    ~CTime(); //系统自动添加的默认析构函数（定义）
    //省略输出流重载...
};

#include "CTime.hpp"

CTime::CTime() { //系统自动添加的默认构造函数（实现）
}
CTime::~~CTime() { //系统自动添加的默认析构函数（实现）
}
//省略setTime的实现...
```

例子：没有拷贝构造函数的风险

```
1  class CDog
2  {
3      public:
4          char* name;
5          CDog(char* name)
6          friend os & operator << (ostream & os, const CDog & dog1)
7          {
8              os << "I'm a dog, name is "<<dog1.name;
9              return os;
10         }
11     };
12     CDog::CDog(char* name)
13     {
14         this->name = name;
15     }
16     void eg3_6_2()
```

```

17 {
18     using namespace N2_CDOG;
19     char name1[] = "Lao Hei";
20     CDog dog1(name1), dog2 = dog1;
21
22     dog2.name[0] = 'H';
23     cout << "dog1:"<< dog1 << endl;
24     cout << "dog2:"<< dog2 << endl;
25
26 }

```

浅拷贝

this指针

- this指针是一个指向当前对象的特殊指针。
- 每个非静态成员函数隐藏有一个this指针的函数参数，当通过一个对象调用成员函数时，编译器将把当前对象的地址传递给this指针。

//我们的代码

```

void CTime::showTime() {
    cout << hour << ':' << minute << ':'
        << second << endl;
}

```

//编译器处理的代码

```

void CTime::showTime(CTime* const this) {
    cout << this->hour << ':' << this->minute
        << ':' << this->second << endl;
}

```

```

CTime t1(11, 30);
t1.showTime(); //我们的代码
t1.showTime(&t1); //编译器处理的代码

```