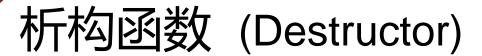
析构函数

郭 炜 刘家瑛





- ▲ 成员函数的一种
 - ・名字与类名相同
 - 在前面加 '~'
 - 没有参数和返回值
 - 一个类最多只有一个析构函数

构造函数

- 成员函数的一种
- 名字与类名相同
- 可以有参数, 不能有返回值
- 可以有多个构造函数用来初始化对象

析构函数 (Destructor)

- → 対象消亡时 → 自动被调用
 - 在对象消亡前做善后工作
 - 释放分配的空间等
- 定义类时没写析构函数,则编译器生成缺省析构函数
 - 不涉及释放用户申请的内存释放等清理工作
- ▲ 定义了析构函数,则编译器不生成缺省析构函数

```
class String{
    private:
        char * p;
    public:
        String () {
           p = new char[10];
       ~ String ();
String ::~ String() {
    delete [] p;
```

析构函数和数组

- 对象数组生命期结束时
- 对象数组的每个元素的析构函数都会被调用

```
class Ctest {
  public:
    ~Ctest() { cout<< "destructor called" << endl; }
int main () {
    Ctest array[2];
    cout << "End Main" << endl;</pre>
    return 0;
```

输出:

End Main

destructor called

destructor called

析构函数和运算符 delete

4 delete 运算导致析构函数调用

Ctest * pTest;

pTest = new Ctest; //构造函数调用

delete pTest; //析构函数调用

pTest = new Ctest[3]; //构造函数调用3次

delete [] pTest; //析构函数调用3次



构造函数和析构函数调用时机的例题

```
class Demo {
       int id;
   public:
       Demo(inti)
           id = i;
           cout << "id=" << id << " Constructed" << endl;
       ~Demo()
           cout << "id=" << id << " Destructed" << endl;
```

Demo d1(1);
void Func(){
static Demo d2(2);
Demo d3(3);
cout << "Func" << endl;
d2 is static, no need to destruct in the function but destruct in the main. d3 will be destructed in the end of function
Demo d4(4);
d4 = 6; 类型转换构造函数
cout << "main" << endl;
{
cout << "main ends" << endl;
return 0;
作用域,离开作用 域要destructed

id=1 Constructed id=4 Constructed id=6 Constructed id=6 Destructed main id=5 Constructed id=5 Destructed id=2 Constructed id=3 Constructed Func id=3 Destructed main ends id=6 Destructed

id=2 Destructed

id=1 Destructed

最先定义的,最后 destruct

构造函数和析构函数在不同编译器中的表现

- ▲ 各别调用情况不一致
 - 编译器有bug
 - 代码优化措施
- ▲ 前面讨论的是C++标准