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（例如回收创建对象时消耗的各种资源），这个函数被称为析构函数。

析构函数（Destructor）也是一种特殊的成员函数，没有返回值，不需要用户调用，而是在销毁对象时自动执行。与构造函数不同的是，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，那么编译器会自动生成。

析构函数举例：

```
#include <iostream>
using namespace std;
class Student{
private:
    char *name;
    int age;
    float score;
public:
    //构造函数
    Student(char *, int, float);
    //析构函数
    ~Student();
    //普通成员函数
    void say();
};
Student::Student(char *name1, int age1, float score1):name(name1), age(age1), score(score1){}
Student::~Student(){
    cout<<name<<"再见"<<endl;
}
void Student::say(){
    cout<<name<<"的年龄是 "<<age<<"，成绩是 "<<score<<endl;
```

```
}
int main(){
    Student stu1("小明", 15, 90.5f);
    stu1.say();

    Student stu2("李磊", 16, 95);
    stu2.say();

    Student stu3("王爽", 16, 80.5f);
    stu3.say();
    cout<<"main 函数即将运行结束"<<endl;

    return 0;
}
```

运行结果：

```
小明的年龄是 15，成绩是 90.5
李磊的年龄是 16，成绩是 95
王爽的年龄是 16，成绩是 80.5
main 函数即将运行结束
王爽再见
李磊再见
小明再见
```

可以看出，析构函数在 main 函数运行结束前被执行，并且调用顺序和构造函数正好相反，为了方便记忆，我们可以将之理解为一个栈，先入后出。

> 析构函数的执行顺序为什么是反的，请看阅读下节《深入内存模型和函数调用机制，理解析构函数的执行顺序》。

析构函数在对象被销毁前执行；要知道析构函数什么时候被调用，就要先知道对象什么时候被销毁。

对象可以认为是通过类这种数据类型定义的变量，它的很多特性和普通变量是一样的，例如作用域、生命周期等。由此可以推断，对象这种变量的销毁时机和普通变量是一样的。

总结起来，有下面几种情况：

1) 如果在一个函数中定义了一个对象（auto 局部变量），当这个函数运行结束时，对象就会被销毁，在对象被销毁前自动执行析构函数。

2) static 局部对象在函数调用结束时并不销毁，因此也不调用析构函数，只有在程序结束时（如 main 函数结束或调用 exit 函数）才调用 static 局部对象的析构函数。

3) 如果定义了一个全局对象，也只有在程序结束时才会调用该全局对象的析构函数。

4) 如果用 new 运算符动态地建立了一个对象，当用 delete 运算符释放该对象时，先调用该对象的析构函数。

> 如果你对 auto、static、extern 等关键字不理解，请猛击：C语言动态内存分配及变量存储类别

注意：析构函数的作用并不是删除对象，而是在撤销对象占用的内存之前完成一些清理工作，使这部分内存可以分配给新对象使用。