面向对象方法与C++程序设计

第2章　　类与对象

大连理工大学
主讲人-赵小薇

构造函数

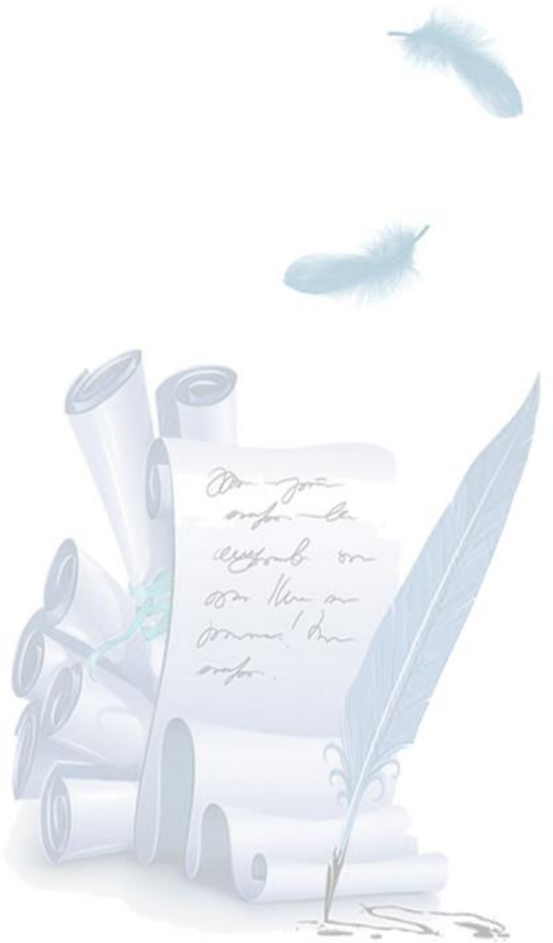


构造函数是一种特殊的函数，主要用来在创建对象时初始化对象，即为对象成员数据赋初始值。共有三个作用：

1. 为对象分配空间并初试化；
2. 对数据成员赋值；
3. 请求其他资源。

•通常将构造函数声明为公有成员函数（不是必须的）；
•构造函数的名字与类名相同，不能任意命名；
•构造函数不具有类型，无返回值，因而不能指定包括void在内的任何返回值类型。 。

使得每个对象生成时各不相同



构造函数语法格式



定义构造函数的语法格式为：

构造函数名（形参）:初始化列表

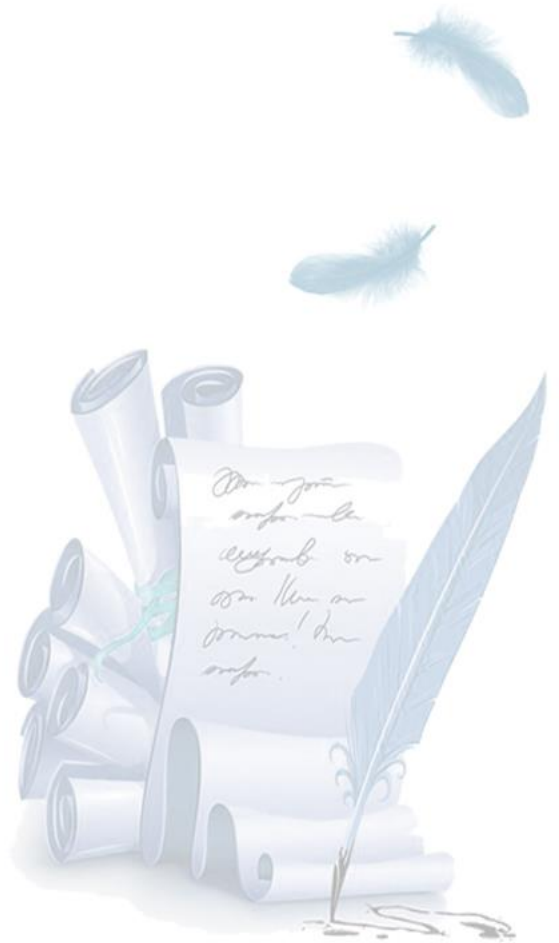
{ 函数体 }

初始化列表格式为：

成员名(实参)[, 成员名(实参)…]

例如

```
Clock(int h , int m , int s ): hour(h),minute(m)
{ second=s; }
```



构造函数的调用方式

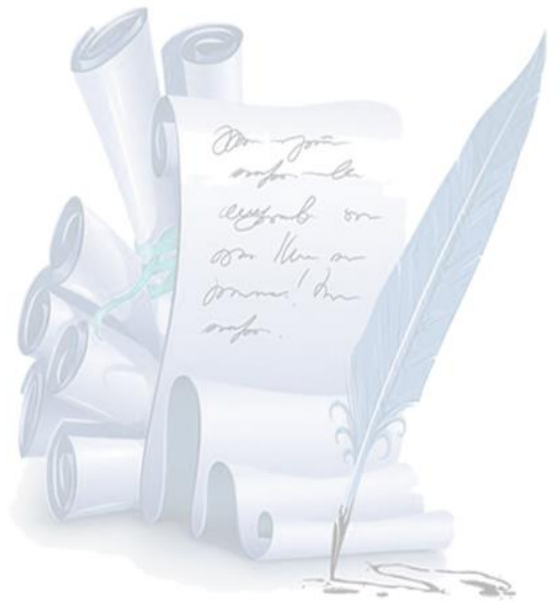


构造函数的调用方式是在定义对象时自动执行的特殊成员函数，用户不能显示调用构造函数。

类名 对象名(实参列表);
或
类名* 对象名=new类名(实参列表);

Clock c1(1,2,3);
Clock c2;

当构造函数无参数时，对象名后面不能加空的括号。需要重载3个参数和无参的构造函数，后面介绍



默认构造函数

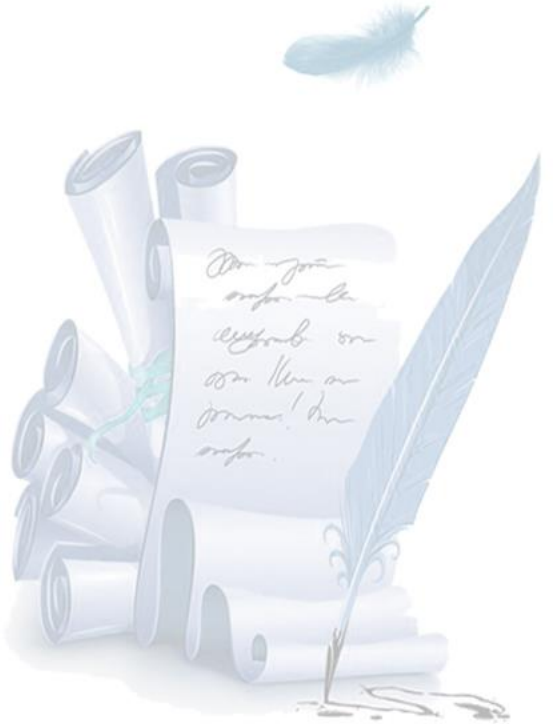


默认构造函数（default constructor）就是在没有显式提供初始化式（定义的对象名后无实参表）时调用的构造函数

（1）系统生成的默认构造函数

当类中没有定义构造函数，编译系统就自动生成一个默认的构造函数，这个默认的构造函数不带任何参数，函数体为空，形式为：**类名::构造函数名(){ }**

```
Clock( ){ }    //系统自动生成的
Clock clock1;  //注意clock1后面无空括号
```



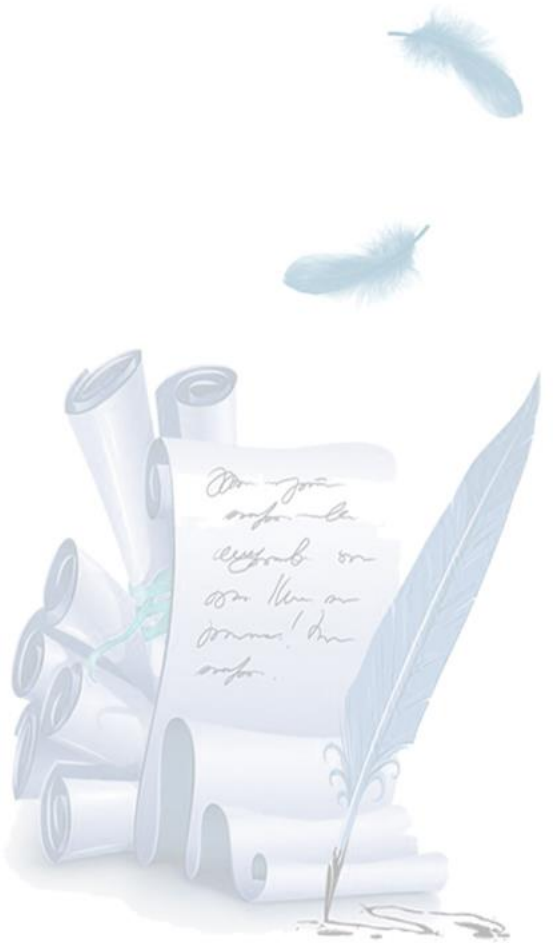
默认构造函数



（2）自定义的默认构造函数

利用系统默认生成的构造函数，并不能起到给对象赋初值的作用，为此，可以显示定义构造函数，使得对象按照构造函数约定的值给对象赋值。

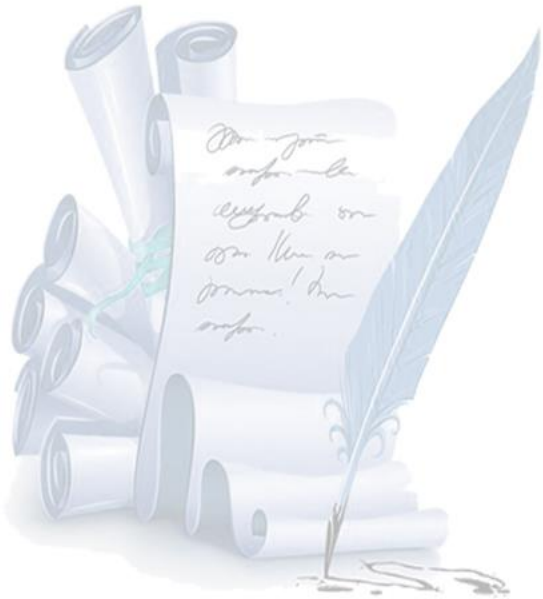
```
Clock( ){hour=0; minute =0; second=0; }
Clock clock1;  //注意clock1后面无空括号
```



有参数的构造函数



```
class  Clock
{
private :
    int hour, minute, second;
public :
Clock(int h , int m , int s ){hour=h; minute =m; second=s; }
void Clock::setClock (int h, int m, int s);
void Clock::showClock ( );
};
Clock clock1(1,2,3);
```



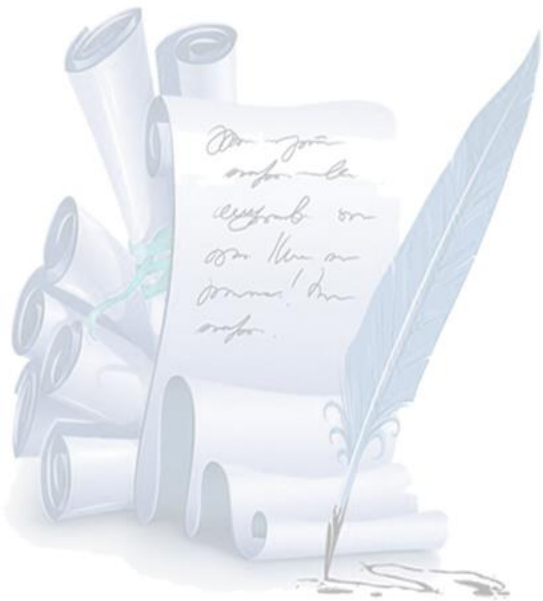
有参数的构造函数



（1）构造函数重载

```
Clock(int h , int m , int s ) :hour(h), minute(m), second(s){  }
Clock( ) :hour(0), minute(0), second(0){  }

Clock  clock1(1,2,3);     //需要3个参构造函数
Clock  clock2;            //需要无参构造函数
```



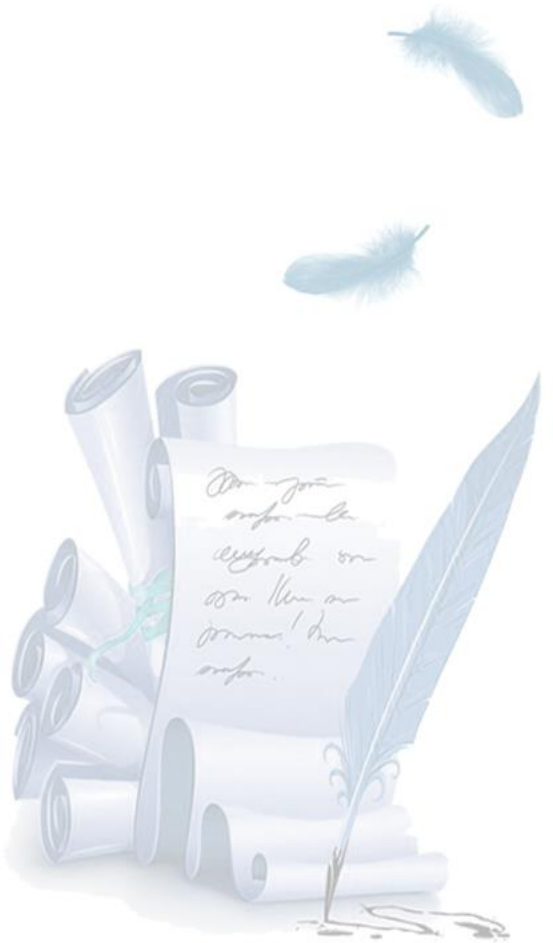
有参数的构造函数



（2）带默认参数的构造函数

有些构造函数的参数值通常是不变的，只有在特殊情况下才需要改变参数的值。

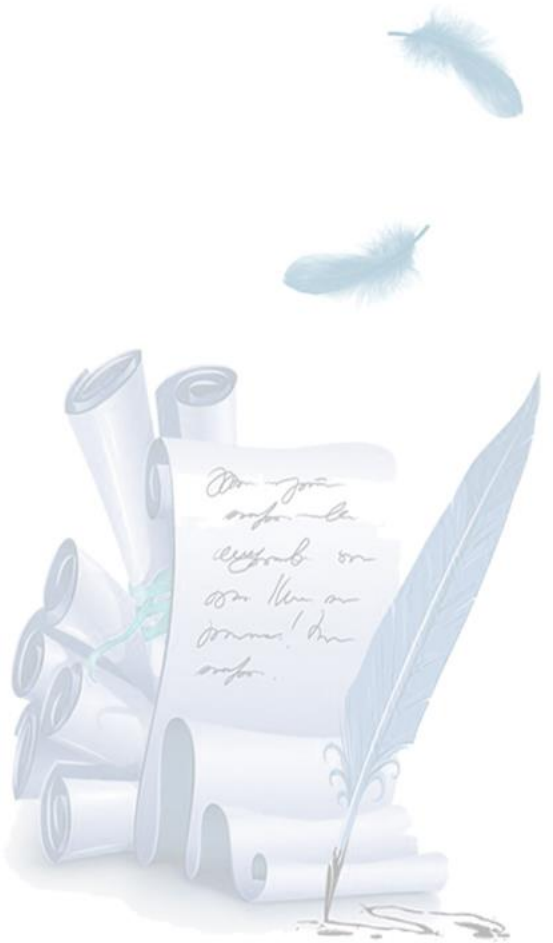
```
Clock(int h=0 , int m=0 , int s=0 ){hour=h; minute =m; second=s; }
Clock  clock0;          // clock0的数据成员的值为0、0、0
Clock  clock1(1);       // clock1的数据成员的值为1、0、0
Clock  clock2(1,2);     //clock2的数据成员的值为1、2、0
Clock  clock3(1,2,3); // clock3的数据成员的值为1、2、3
```



析构函数



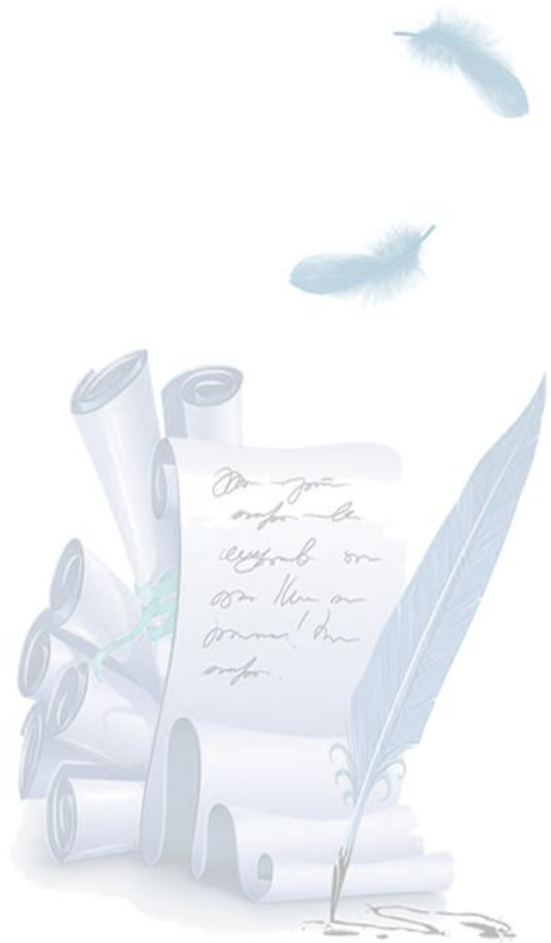
- 析构函数，在对象生命周期结束自动执行析构函数，完成清理内存工作，并可以执行指定的其他操作。
- 析构函数是一种在结束对象调用时自动执行的特殊的成员函数，一个类中只能定义一个析构函数。
- 析构函数声明为公有成员，析构函数名由波折号“~”与类名组合而成，析构函数不接受参数（所以不能重载，只有一个析构函数），没有返回值。



析构函数的一般形式



```
class 类名
{
    public:
      ……
    ~类名（）
     { 指定的操作；}
};
```

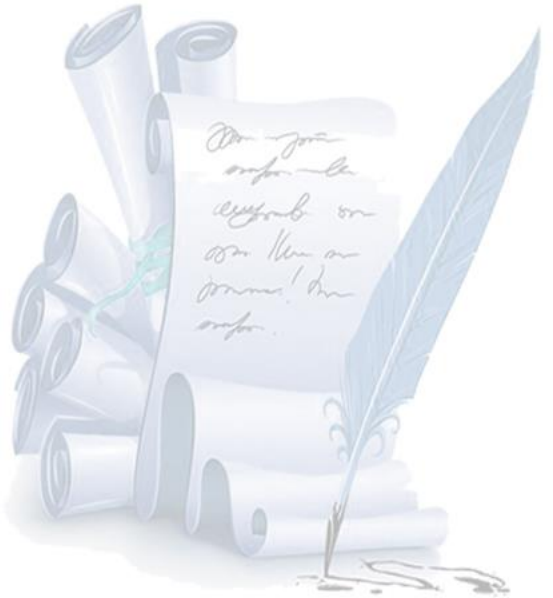


默认的析构函数



当类中没有显式地定义析构函数，则系统会自动生成一个默认的析构函数，该函数是一个空函数。默认的析构函数的格式如下：

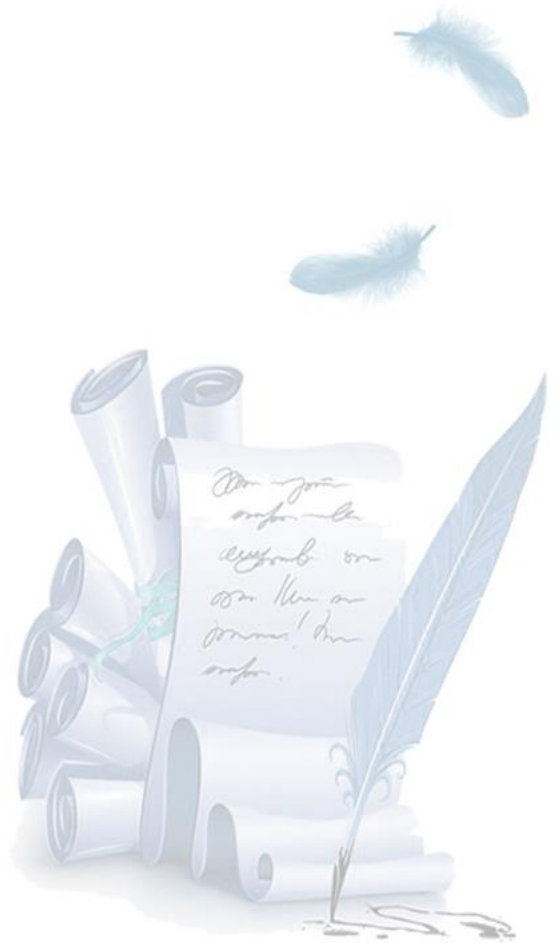
类名 :: ~类名( ){　　}



析构函数的用途



通常在构造函数中用new运算符为对象额外申请了一些空间（额外资源，不属于对象空间），在对象结束生命周期时需要使用析构函数，利用delete运算符释放new运算符所申请的空间。

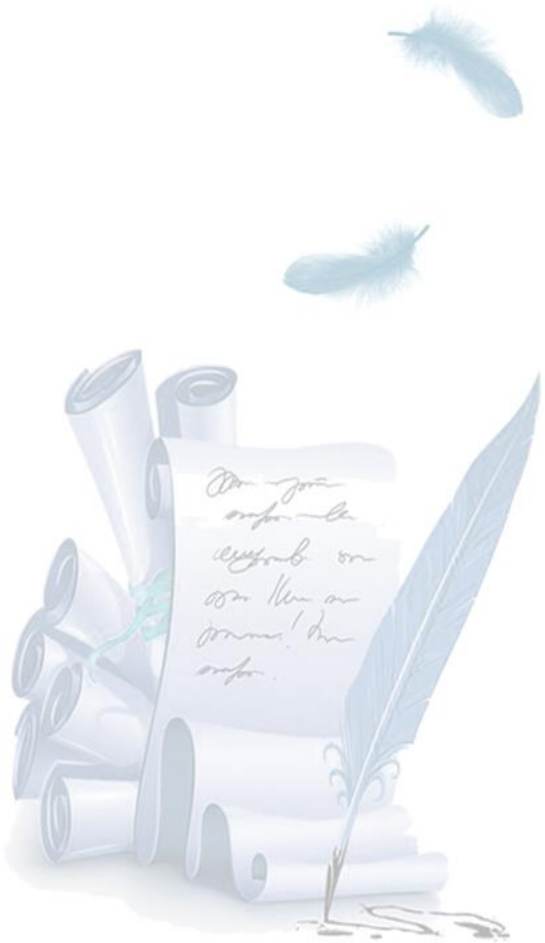


举例



```
class Point
{
private:
   double x, y;
   char *name;
public:
Point(char *n=NULL, double
x=0.0, double y=0.0);
   ~Point();
   void disp( );
};
```

```
Point::Point (char *n, double a,
double b){
  x = a;  y = b;
  if(n) {
      name=new char[strlen(n)+1];
      strcpy(name,n);
  }
  else{
       name=new char[8];
       strcpy(name,"no name");
}
cout<<name<<"  constructing"<<endl;
}
```



```
Point::~Point(){
    cout<<name<<"
destructing"<<end
    delete
}
void
Point
<x
```

```
ain(){
义对象
int p1("home",1.0,2.0);
t p2("school",3.0);
t p3;
出对象
<<"p1=";
sp();
<<"p2=";
disp();
cout<<"p3=";
p3.disp();
}
```

```
home  constructing
school  constructing
no name  constructing
p1=home:1,2
p2=school:3,0
p3=no name:0,0
no name  destructing
school  destructing
home  destructing
```

对象析构的顺序恰好和对象的构造顺序相反。