《面向对象程序设计》朋辈课程 · 第六辑

对象生灭

信息学院 赵家宇

PART 01

构造函数

对象的生灭：构造与析构

- 在对象创建或是销毁（无论显示或是隐式）过程，都会有对应函数（构造函数、析构函数）自动执行。
- 构造函数是特殊的成员函数，只要创建类类型的新对象，都要执行构造函数；
- 析构函数是特殊的成员函数，在对象销毁的时候，会有析构函数自动执行。

回顾：已经接触过的对象

`string s1;`	默认构造函数，s1为空串
`string s2(s1);`	复制构造函数，将s2初始化为s1的一个副本(s1可以是字符数组或string)
`string s3(“value”);`	将s3初始化为一个字符串字面值副本
`string s4(n,’c’);`	将s4初始化为字符’c’的n个副本

- 对象创建过程一定伴随着构造函数的自动调用；
- 构造函数的运行时间特殊，往往在对象初始化时被自动调用；
- 多种初始化形式表明构造函数的重载；

构造函数与初始化

- 初始化，也就是定义对象的同时赋予初始值。
- 由于对象反映了客观事物的属性，因此在对象的创建时，数据成员应该有确定的值，因此需要作某些初始化的工作，如对数据成员赋初值。例如，date类的对象需要有具体的年月日等信息。
- 如果一个数据成员未被赋值，则它的值是不可预知的，因为在系统为它分配内存时，保留了这些存储单元的原状，这些将成为这些数据成员的初始值，具有数据访问的风险。

对象初始化方式

- 类的数据成员是不能在声明类时初始化，其原因在于：类并不是一个实体，而是一种抽象类型，不占存储空间。
- 如果一个类中所有的成员都是公用的，则可以在定义对象时对数据成员进行初始化（类似结构体各数据成员），但是如果数据成员是私有的，或是类中有protected的成员，则不能用这种方法初始化。

```
class Time {
   int hour=0;
   int minute=0;
   int sec=0;
}; //错误
```

```
class Time {
   public:
        int hour;
        int minute;
        int sec;
};
Time t1={14,56,30};//正确
```

构造函数与初始化

- C++提供了构造函数，用户可通过构造函数来完成期望在创建对象时需要完成的事情。
- 构造函数是一种特殊的成员函数，具有以下特点：
  - 不需要用户来调用，而是在建立对象时自动执行；
  - 构造函数可以建立有名称的对象（显式定义）和未命名的对象（隐式生成）；
  - 构造函数的名字必须与类名同名，而不能由用户任意命名；
  - 构造函数不具有任何类型，不返回任何值；

构造函数的重载

- 构造函数没有返回值，因此不需要再定义构造函数时声明类型（不能加void）；
- 构造函数不需用户调用，也不能被用户调用。下面用法错误

```
t1.Time(); //不能用调用一般成员函数的方法调用构造函数
```

- 如果用户没有定义构造函数，则C++系统会自动生成一个构造函数，只是该构造函数的函数体是空的，也没有参数，不执行初始化操作。

构造函数的分类

- 默认构造函数：由编译器隐式提供，没有参数，不进行任何处理；
- 用户自定义构造函数：函数功能由用户定义，根据初始化要求设计函数体和函数参数，如赋初值、值的有效性校验等。又包括带参数的构造函数和不带参数的构造函数两类。
- 若用户自定义了构造函数，则默认构造函数一定不会执行；否则默认构造函数会被隐式调用。

不带参数的构造函数

- 若构造函数不带参数，在函数体中对各数据成员赋值，此类方式使该类的每一个对象都得到同一组初值。

```
class Time {
    private:
        int hour, minute, second;
    public:
        Time() {
            hour = 11;
            minute = 11;
            second = 11;
        }//不带参数的构造函数
};//构造函数没有返回值，函数名与类名一致
```

带参数的构造函数

- 带参数的构造函数可以对不同的对象赋予不同的初值，在调用构造函数时，从主调函数中获得不同的数据并传递给构造函数，以实现不同的初始化；

```
Time d1(11, 11, 11);
```

```
class Time {
    private:
        int hour, minute, second;
    public:
        Time(int h, int m, int s) {
            hour = h;
            minute = m;
            second = s;
        }//带参数的构造函数
};//构造函数没有返回值，函数名与类名一致
```

带参数的构造函数

- 构造函数的首部的一般格式：构造函数名(类型1 形参1,类型2 形参2,..)
- 实参是在定义对象时给出：类名对象名(实参1,实参2,….)

```
class Date {
    private:
        int year, month, day;
    public:
        Time(int x, int y, int z) { }
};
Date::Date (int x, int y, int z) {
    year=x;
    month=y;
    day=z;
}//外部构造函数定义
```

```
int main() {
    int x=2024, y=4, z=18;
    Date d1(2024, 4, 17);
    Date d2(-1, -2, -3);
    Date d3(x, y, z);
    Date d4(d3.year, d3.month, d3.day);
}
```

带默认参数的构造函数

- 构造函数中参数的值可以通过实参传递，也可以指定为某些默认值，即用户不指定实参值，编译系统就使用默认值；
- 在调用构造函数时若没有提供实参值，系统就按照默认的参数值对对象进行初始化，尤其适用于对每一个对象都有同样的初始化时；
- 如果在类定义中仅包含函数的声明，则默认的参数值应该放在声明中，而不能放在函数定义中；函数参数需要遵循普通函数设置默认参数的要求。

带默认参数的构造函数

- 构造函数中参数的值可以通过实参传递，也可以指定为某些默认值，即用户不指定实参值，编译系统就使用默认值；
- 在调用构造函数时若没有提供实参值，系统就按照默认的参数值对对象进行初始化，尤其适用于对每一个对象都有同样的初始化时；
- 如果在类定义中仅包含函数的声明，则默认的参数值应该放在声明中，而不能放在函数定义中；函数参数需要遵循普通函数设置默认参数的要求。

带默认参数的构造函数

- 如果在类定义中仅包含函数的声明，则默认的参数值应该放在声明中，而不能放在函数定义中；
- 函数参数需要遵循普通函数设置默认参数的要求。

```
class Date {
   private:
      int year, month, day;
   public:
      Time(int x=2024, int y=4, int z=17) { }
      Time(int x=2024, int y, int z) { }
};
Date::Date (int x=2024, int y=4, int z=17) { }
Date::Date (int x, int y, int z) {
    year=x; month=y; day=z;
}
```

- 回顾：函数带默认参数
- 一个函数有多个参数时，可以给该函数的部分参数或全部参数设置默认值；在给函数的部分参数设置默认值时，应该从参数表的右端开始，在设置了默认值的参数的右端不允许出现没有默认值的参数。

构造函数的初始化列表

- 在构造函数中，可以用显式的赋值语句来初始化对象的成员，还可以使用“构造函数初始化列表”，这是一种值得推荐的方式；
- 初始化列表以一个冒号开始，接着是一个以逗号分隔的数据成员列表，每个数据成员后面跟一个放在圆括号中的初始化式；构造函数初始化式只在构造函数的定义中而不是声明中指定；

```
Time::Time(int h,int m,int s):hour(h),minue(m),second(s) {  }
```

构造函数的初始化列表

- 构造函数初始化列表，仅给出用于初始化成员的各个值，并未对初始化成员执行的次序进行指定；成员被初始化的次序依赖于成员在类中定义的次序；

```
class Time {
    private:  //定义的顺序要与初始化列表一致
        int second;
        int hour;
    public:
        Time(int s):hour(1), second(s) { } //会有莫名的值
        Time(int h):second(1), hour(h) { } //正确
};
```

构造函数的初始化列表

- 很多时候，初始化列表可以转化为通过在函数体内对数据成员赋值来实现。但是在某些情况下，只能使用初始化列表：
  - 某个类中的类成员，没有默认构造函数（若没有为类成员提供初始化式，则编译器会隐式地使用类成员的默认构造函数。而该类成员若恰好没有默认构造函数，那么编译器尝试使用默认构造函数就会失败）；
  - 某个类中有const成员；
  - 某个类中有引用类型的成员；
  - 如果类存在继承关系，派生类必须在其初始化列表中调用基类的构造函数；

构造函数的初始化列表

```
class A  {
   public:
      A(int size): SIZE(size) {  };
      A(int size) { SIZE = size };
   private:
      const int SIZE;
};
int main()  {
   A a(100);
}
```

- 在类中声明变量为const类型，但是不可以初始化；
- const常量的初始化必须在构造函数初始化列表中初始化，而不可以在构造函数函数体内初始化。

默认构造函数

- 如果为类已经定义了一个带参数的构造函数，还想要使用无参构造函数，则必须自己定义自己的“默认构造函数”；使用默认构造函数时不需要添加( )。

```
class Time{
  private:
     int hour; int minute;
  public:
     Time(int x=1, int y=100):hour(x) minute(y) { }
     Time(int x) {hour=x; minute=30;}
};
Time T;//调用默认构造函数，不需要括号
Time T2(16);//调用带参数的自定义构造函数，需要括号
```

- 方法1：不带参数
- 方法2：全部默认值
- 左边的程序对默认构造函数进行了重载，此时编译器隐式形成的默认构造函数不会发挥作用。

类的特殊成员:类成员

- 类定义中有数据成员和成员函数，数据成员可以是内部数据类型的变量实体，也可以是对象实体;
- 在C++中，成员对象的构造顺序与它们在类中声明的顺序有关，而不是它们在初始化列表中的顺序；成员对象需要在初始化列表中调用其构造函数来初始化。

类的特殊成员：类成员

```
class B {
    B(int x) {}  // B的构造函数
};
class A {
    private:
        B b; // B类的对象作为A的成员对象
        int a;
    public:
        A(int x, int y) : b(x), a(y) {}
};
```

- 类 A 有一个成员对象b。类A的构造函数使用初始化列表来调用B的构造函数并传递参数x来初始化b。
- 当创建类 A 的对象时，首先会按照成员对象在类定义中的声明顺序（即先b后a）构造它们，然后再执行构造函数的主体。
- 如果成员对象没有默认构造函数，那么必须在初始化列表中明确指定其构造函数和参数。否则，编译器不知道如何构造没有默认构造函数的成员对象。

类的特殊成员:指针成员

```
class Student {
  public:
    Student(int pid,char *p,float s);
    Student();
    void modify(float s);
    void display();
    ~Student();
  private:
    int id;
    char *name;
    float score;
};
```

```
Student::Student(int pid,char *p,float s) {
  id = pid;
  name = new char[strlen(p)+1];
  strcpy(name,p);
  score = s;
}
Student::Student( ) {
   id = 0;
   name = new char[11];
   strcpy(name,”No name”);
   socre = 0;
}
```

合成的默认构造函数与初始化

- 合成的默认构造函数是编译器创建的构造函数，将使用与变量初始化相同的规则来初始化成员，类成员通过运行各自的默认构造函数来实现初始化；
- 可以使用 `= default` 要求编译器自动生成默认构造函数；
- 内置或复合类型（如数组）的成员，若对象定义在全局作用域中，才初始化；而若对象定义在局部作用域中时，不进行初始化；鉴于该原则，若类中包含内置或复合类型的成员，则应该自定义构造函数来初始化这些成员。

构造函数的重载

- 通过重载构造函数提供不同的构造函数，允许用户指定不同的方式来初始化数据成员；
- 构造函数的重载与普通成员函数的重载一样，具有相同的名字，而参数的个数或参数的类型不相同；

```
class Person {
  private:
      string name; int age;
  public:
     Person(): name(""), age(0) { }
     Person(string& n): name(n), age(0) { }
     Person(string& n, int a): name(n), age(a){ }
};
int main() {
    Person person1;
    Person person2("Alice");
    Person person3("Bob", 30);
}
```

构造函数的重载

- 尽管构造函数同名，系统根据函数调用的形式（实参）去确定将选用的构造函数；
- 为防止编译器出现困惑，应该避免没有带参数的构造函数与全部为默认参数的构造函数同时出现（重载）；
- 若在建立对象时选用的是无参构造函数，应注意正确书写定义对象的语句，尤其注意避免写括号。

PART 02

析构函数

析构函数

- 当对象消亡时，在系统收回它所占的内存空间之前，对象类的析构函数会被自动调用。
- 对象的空间不是由析构函数收回的。当对象在创建过程申请资源，则需要自定义析构函数来归还。

```
class A {
  int x;
  pulbic:
     A( );
     ~A( ); //析构函数
};

void f( ) {
   A a;  //自动调用构造函数
   …
} //自动调用析构函数
```

析构函数

- 对象空间（本体空间）是由操作系统撤销的;
- 若程序执行过程中对象有申请资源（new或malloc），则需要在析构函数中显式调用相应的删除函数（delete或free）来进行资源的回收。(遇到指针要注意！)

```
class class{
   private:
      int* i;
   public:
      ResourceHolder() {
         i = new int[10];
      }
      ~ResourceHolder() {
         delete[] i;
      }
};
```

析构函数处理指针成员

```
class Student {
  public:
    Student(int pid,char *p,float s);
    Student();
    ~Student();
  private:
    int id;
    char *name;
    float score;
};
```

```
 Student::Student(int pid,char *p,float s) {
    id = pid; score = s;
    name = new char[strlen(p)+1];
    strcpy(name,p);
}
 Student::Student( ) {
    id = 0; socre = 0;
    name = new char[11];
    strcpy(name,”No name”);
}
```

```
 Student::~Student() {
    delete[] name;
 }
```

构造函数和析构函数调用顺序

- 在对象的生灭过程中，实际是一个函数关于栈的调用过程
- 关于局部对象（不是全局对象），调用构造函数与定义对象的顺序相同，而调用析构函数的次序正好与创建的顺序相反。
- 若有类成员出现时，类的构造函数调用遵循按声明顺序以此构造各类成员，最后生成自己；析构函数调用遵循栈。

构造函数和析构函数调用顺序

```
class Base {
private:
   int b;
public:
    Base(){cout<<"Base的构造函数\n";}
    ~Base() {cout<<“Base的析构函数\n";}
};
class A {
private:
   int a1; Base a2;
public:
    A() {cout<<"大类A的构造函数\n";}
    ~A() {cout<<"大类A的xigou函数\n";}
};
```

```
int main() {
      A x1;
}

/*
输出结果:
Base的构造函数
大类A的构造函数
大类A的析构函数
Base的析构函数
*/
```

感谢倾听

给个好评！