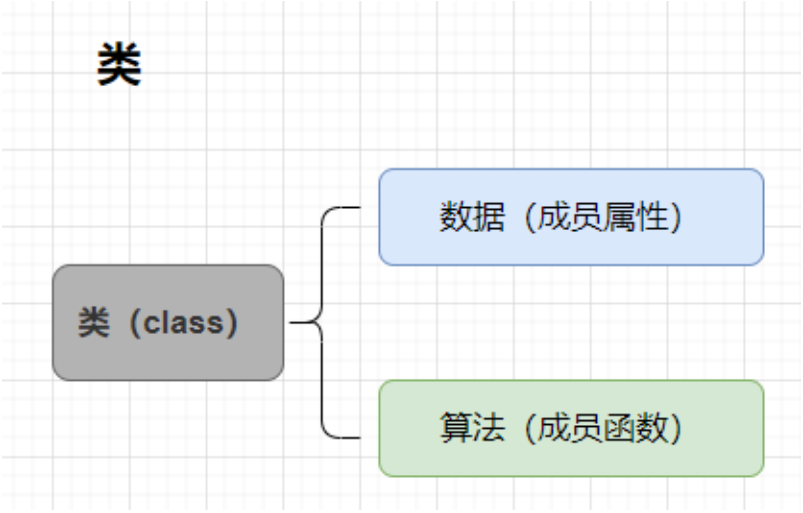
第2章-类基础

author：岳石磊 copyright：科林明伦 内部资料禁止外泄

1. 类封装

C语言中一般是由数据和算法组成，数据和算法彼此独立，关联性不强，在C++中将相互关联数据和算法封装起来，形成结构体或类，无论类还是结构体都是一个抽象的概念，只有定义类的变量时，数据才会真实存在，这个变量我们称之为对象，C++程序过程中，尽量避免单独的数据和算法，而是由一个个类对象组成的，这就是面向对象。
类(class)：完成某一功能的数据和算法的集合，是一个抽象的概念。
对象：类的一个实例，具体的概念，是真正存在于内存中的。



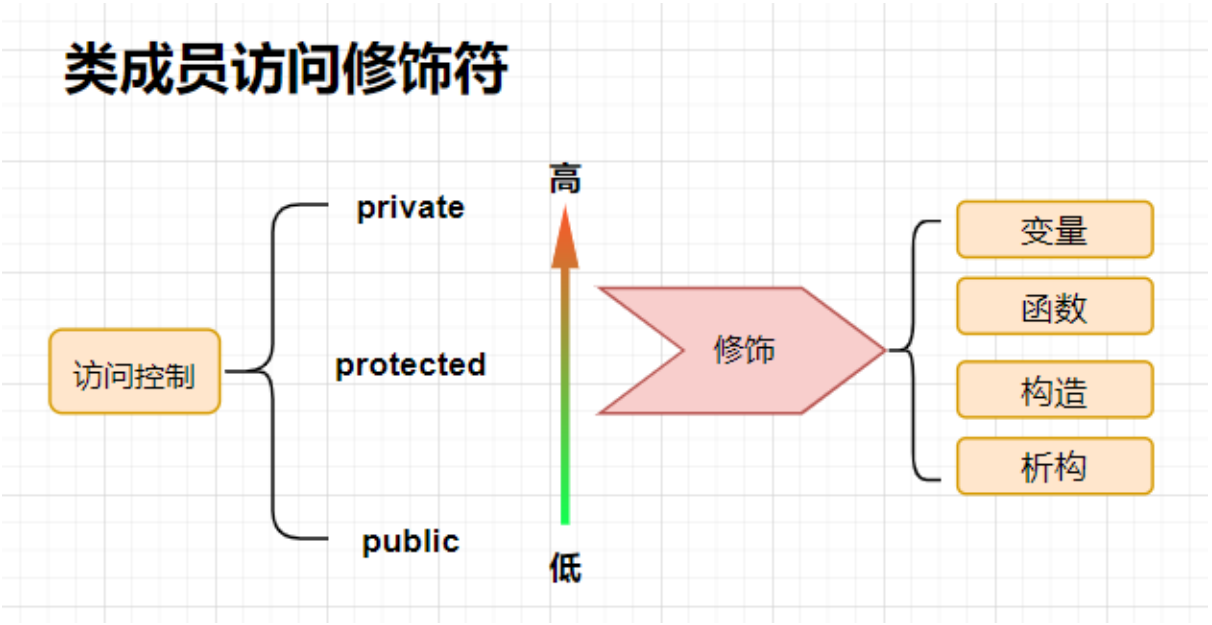
定义类的关键字 class，类名一般以大写的C开头，成员属性一般以m_开头。

```
class CTest{
    int m_a;      //成员属性（成员变量）
    void show(); //成员方法（成员函数）
};
```

类成员访问修饰符：描述了类成员的访问控制，即所能使用的一个范围， 共有的public、保护的protected、私有的private（默认）。

```
public      ：对外公开，在类内、类外都可以使用。
protected   ：在类内和子类中可以使用。
private     ：只能在类内使用。
```

对于类内的私有成员，我们可以提供访问接口（getxxx，setxxx）来按照一定的规则进行访问。



2. 构造函数

通常在定义变量时要初始化，在定义类对象时，成员属性并没有得到初始化。C++类提供了一种特殊的函数-构造函数。
构造函数：其作用是用来初始化类成员属性。空类中存在一个默认的无参数的构造，函数名为当前类名，无返回值。

```
class CTest{
    CTest(){ }  //默认无参构造
};
```

构造函数并不需要我们手动调用，在定义对象的时候会自动调用，这个默认的无参构造是编译器给提供的，函数体代码为空，所以在定义对象时虽然调用了，但并没有这正给成员初始化。所以多数情况下需要我们手动重构构造函数。

一个类中的构造函数允许存在多个，他们是函数重载的关系，重构的构造函数可以指定参数来符合我们需要的初始化过程。
注意：**只要重构了任何的构造函数，编译器将不会再提供那个默认的无参构造了。**
定义多个对象可能会执行不同的构造，这就要看在定义对象时如何指定参数了，会根据参数的类型、数量自动匹配对应的构造，但一个对象最终只能执行其中一个构造。

3. 析构函数

析构函数：与构造函数相对应的析构函数，其作用是用来回收**类中成员申请的额外空间**，而不是对象本身。空类中存在一个默认的析构函数，函数名为~类名，无返回值，无参数。

```
class CTest{
    ~CTest(){ }  //默认析构
};
```

析构函数在对象的生命周期结束的时候，自动调用，编译器提供的默认析构函数函数体代码也为空，我们可以手动重构，一旦重构，编译器就不会再提供那个默认析构了，与构造不同的是析构函数只允许存在一个。
注意：析构在真正回收对象内存空间之前去调用，额外的空间回收完后，才真正回收对象内存空间。

4. 再谈面向过程与面向对象

面向过程编程（Procedure-Oriented Programming，简称POP）是一种编程模型，也是初始踏入编程时接受的思想，过程即由一系列要执行的计算步骤，以过程为中心依次把要解决问题的步骤、流程分析出来，用函数封装好形成一个一个的模块，在主流程中按照具体的步骤调用相应的函数。

面向过程编程的主体是函数（模块），它以函数为中心，始终关注的是怎么一步一步解决问题，从而实现函数的顺序执行。

总结：自上向下，顺序执行、逐步细化。
举例：把大象放进冰箱：

```
void OpenRefrigerator() {
    cout << "用手抓住冰箱门把手" << endl;
    cout << "缓慢打开门" << endl;
    cout << "拉到一定大的角度" << endl;

}

void PushElephant(int * p_refrig,int& h_refrig,int& h_ele) {
    cout << "找到冰箱的最大格子" << endl;
    if (h_refrig < h_ele) {
        cout << "压缩一下大象" << endl;
        h_ele = 5;
    }
    p_refrig = &h_ele;
    cout << "大象放到冰箱里" << endl;
}

void CloseRefrigerator() {
    cout << "开始推动冰箱门" << endl;
    cout << "继续推动冰箱门" << endl;
    cout << "冰箱门严丝合缝" << endl;
}
```

```
23
24  int main() {
25      int heightEle=10;  //大象的高度
26
27      int heightRefrig=10;  //冰箱的高度
28      int* pEle=nullptr;    //冰箱装的东西
29
30      //--------开始打开冰箱---------------------
31      OpenRefrigerator();
32
33      //--------结束打开冰箱---------------------
34
35      //--------开始装大象-----------------------
36      PushElephant(pEle, heightRefrig, heightEle);
37      //--------结束装大象-----------------------
38
39      //--------开始关闭冰箱---------------------------
40      CloseRefrigerator();
41      //--------结束关闭冰箱---------------------------
42
43      return 0;
44  }
```

面向对象编程（Object-oriented programming，简写：OOP）：面向对象编程是一种编程模型，以对象（数据）为中心，它把要解决的问题分解成各个对象（变量、数据）而不是各个流程、步骤，更专注于对象与对象之间的交互（而不是数据和方法、方法与方法），建立对象的目的不是为了完成一个步骤，而是为了描述某个对象在解决整个问题步骤中的属性和行为。涉及到的属性和方法都被封装到一起包含在其内部。

世界上的每个事物都可以是一个独立的对象，其都有自己的属性和行为，对象与对象之间通过方法来交互。

面向对象编程的分析问题步骤：分析问题中参与其中的有哪些实体，这些实体应该有什么属性和方法，我们如何通过调用这些实体的属性和方法去解决问题。现实世界中，任何一个操作或者是业务逻辑的实现都需要一个实体来完成，实体就是动作的支配者，没有实体，就没有动作发生。

```
1  //定义大象的类
2  class CElephant {
3  public: CElephant(int h) :heightEle(h) {}
4  public:
5      int heightEle ;
6      /* ...其他的属性,如：体重、性别...  */
7
8      /* ...其他的方法,如：吃食、喝水...  */
```

```cpp
};

//定义冰箱的类
class Refrigerator {
public: Refrigerator(int h) :heightRefrig(h), pEle(nullptr){}
public:
    int heightRefrig ;
    CElephant* pEle ;

    /* ...其他的属性,如：冰箱的长、宽、最大容积 ...  */

    void OpenRefrigerator() {
        cout << "用手抓住冰箱门把手" << endl;
        cout << "缓慢打开门" << endl;
        cout << "拉到一定大的角度" << endl;
    }

    void PushElephant(CElephant& ele) {
        cout << "找到冰箱的最大格子" << endl;
        if (heightRefrig < ele.heightEle) {
            cout << "压缩一下大象" << endl;
            ele.heightEle = 5;
        }
        pEle = &ele;
        cout << "大象放到冰箱里" << endl;
    }

    void CloseRefrigerator() {
        cout << "开始推动冰箱门" << endl;
        cout << "继续推动冰箱门" << endl;
        cout << "冰箱门严丝合缝" << endl;
    }
};

int main() {
    Refrigerator refrig(10);
    CElephant ele(10);

    //--------开始打开冰箱---------------------
    refrig.OpenRefrigerator();

    //--------结束打开冰箱---------------------

    //--------开始装大象-----------------------
```

```
    refrig.PushElephant(ele);
    //--------结束装大象-----------------------

    //--------开始关闭冰箱-------------------------
    refrig.CloseRefrigerator();
    //--------结束关闭冰箱-------------------------

    return 0;
}
```

5. 练习1：封装链表

c++中的结构体和类很像，可以有构造-析构，也有访问修饰符，继承、多态等。区别有两点：

1. 类成员属性、方法默认是私有的(private)，而结构体默认是公有的(public)。
2. 当从基类、结构体中继承时，类的默认继承方式是私有的(private)，而结构体是公有的(public)。

此外在使用习惯上看也略有差别：一般情况，使用类来描述功能实现，而结构体通常只是纯粹的表示数据。比如 链表有复杂的数据、很多的方法一般定义成类，而节点只是描述了持有数据的类型和指向下一个节点的指针，一般被定义为结构体。

定义一个链表的结点结构：

```
struct Node {
    int   val;  //存储数据
    Node* pNext;  //指向下一个节点的指针
    Node(int v) {
        val = v;
        pNext = nullptr;
    }
};

int main() {
    Node node(10);
    //在类外可以直接使用
    cout << node.v << endl;   //10
}
```

链表代码：

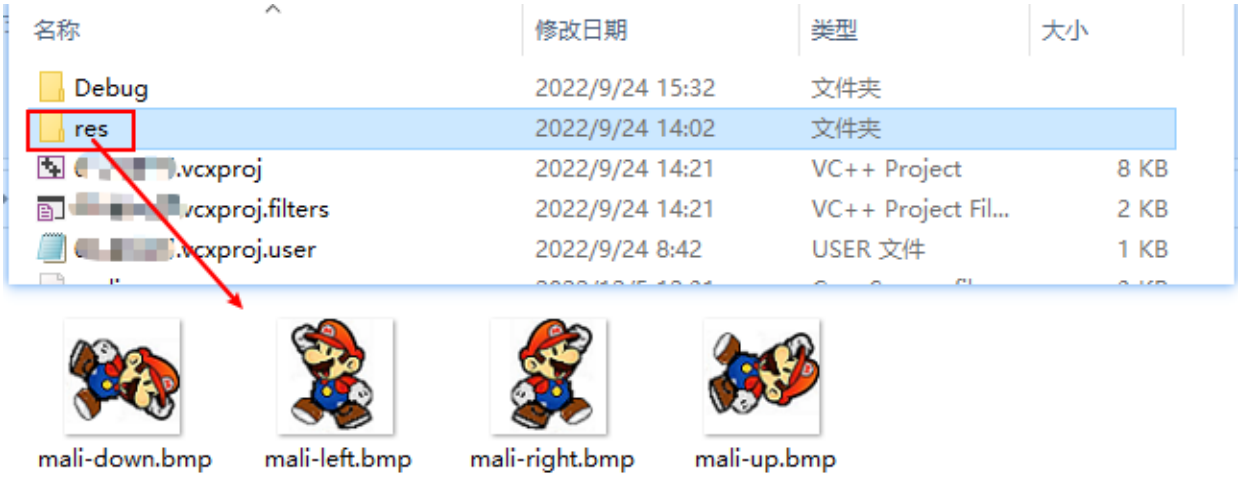
```
struct Node {
//public:   //默认的访问修饰符
    int   val;  //存储数据
```

```
    Node* pNext;  //指向下一个节点的指针
    Node(int v) {
        val = v;
        pNext = nullptr;
    }
};

class CMyList {
public:
    Node* m_pHead;
    Node* m_pEnd;
    int   m_nLength;
public:
    CMyList() {
        m_pHead = nullptr;
        m_pEnd = nullptr;
        m_nLength = 0;
    }
    ~CMyList() {
        Node* pTemp = nullptr;
        while (m_pHead) {
            pTemp = m_pHead;   //标记头结点
            m_pHead = m_pHead->pNext;  //头指针向后移动
            delete pTemp;  //删除标记的
        }
        m_pHead = nullptr;
        m_pEnd = nullptr;
        m_nLength = 0;
    }
public:
    void PushBack(int v) {
        Node* pNode = new Node(v);

        if (m_pHead)    //非空链表
            m_pEnd->pNext = pNode;  //指向下一个
        else   //空链表
            m_pHead =pNode;

        m_pEnd = pNode;

        ++m_nLength;  //长度增加
    }
    void PopFront() {
        if (m_pHead) {   //非空链表
```

```
            Node* pTemp = m_pHead;  //标记要删除的节点
            if (m_pHead == m_pEnd)    //1个节点
                m_pHead = m_pEnd= nullptr;
            else    //多个节点
                m_pHead = m_pHead->pNext;

            delete pTemp;   //删除之前标记的头结点
            pTemp = nullptr;

            --m_nLength;  //长度减少
        }
    }
    void ShowList() {
        Node* pTemp = m_pHead;
        while (pTemp) {
            cout << pTemp->val << "    ";
            pTemp = pTemp->pNext;   //向后移动标记
        }
        cout << endl;
    }
    int GetLength() {return m_nLength;}
};


int main() {
    CMyList lst;
    lst.PushBack(1);
    lst.PushBack(2);
    lst.PushBack(3);
    lst.PushBack(4);
    cout << "长度 = " << lst.GetLength() << endl;  //长度 = 4
    lst.ShowList();   //1    2    3    4

    lst.PopFront();
    lst.PopFront();
    lst.PopFront();
    cout << "长度 = " << lst.GetLength() << endl;   //长度 = 1
    lst.ShowList();   //4
    return 0;
}
```

6. 练习2：人物移动

字符集：Unicode 字符集。
在工程文件同级目录下新建【res】文件夹，在其中放置4中图片如下：




```cpp
#include<easyx.h>
#include<conio.h>

//定义方向键宏
#define DEF_UP        72
#define DEF_DOWN      80
#define DEF_LEFT      75
#define DEF_RIGHT     77
//移动步长
#define DEF_STEP      6

#define DEF_ESC       27

class CGameMali {
public:
    //人物的图片
    IMAGE m_imgUp;
    IMAGE m_imgDown;
    IMAGE m_imgLeft;
    IMAGE m_imgRight;
    //人物的坐标
    int   m_x;
    int   m_y;
    //人物的方向
    int   m_nDirect;
public:
    CGameMali() {
        //创建一个窗口  600x600
```

```
29          ::initgraph(600, 600);
30
31          ::setbkcolor(RGB(255, 255, 255));  //设定背景色白色
32          ::cleardevice();   //使设置的白色，立即生效
33
34          //图片资源的 IMAGE 绑定
35          ::loadimage(&m_imgUp,   L"./res/mali-up.bmp");
36          ::loadimage(&m_imgDown, L"./res/mali-down.bmp");
37          ::loadimage(&m_imgLeft, L"./res/mali-left.bmp");
38          ::loadimage(&m_imgRight,L"./res/mali-right.bmp");
39
40          //初始化成员属性
41          m_x = 300;
42          m_y = 300;
43
44          m_nDirect = DEF_RIGHT;  //初始化方向
45      }
46      ~CGameMali() {
47          ::closegraph();//关闭窗口
48      }
49
50  public:
51      void PaintGame() {
52          //开始绘图
53          ::BeginBatchDraw();
54          ::cleardevice();  //清屏
55
56          //真正的绘图，根据方向显示不同的图片
57          if (m_nDirect == DEF_UP)         ::putimage(m_x, m_y, &m_imgUp)
    ;
58          else if (m_nDirect == DEF_DOWN)  ::putimage(m_x, m_y, &m_imgDow
    n);
59          else if (m_nDirect == DEF_LEFT)  ::putimage(m_x, m_y, &m_imgLef
    t);
60          else if (m_nDirect == DEF_RIGHT) ::putimage(m_x, m_y, &m_imgRig
    ht);
61
62          //结束绘图
63          ::EndBatchDraw();
64      }
65
66      void RunGame() {
67          while (1) {
68              //_getch 这是一个阻塞的函数，当只要按下按键的时候，才会返回
```

```
            int key = _getch();   //获取按下的具体按键，
            if (key == DEF_ESC) {
                //弹出框（对话框）
                int res = ::MessageBox(NULL, L"是否确定退出？", L"提示",
MB_OKCANCEL);
                if (res == IDOK) {  //点击了确定，则退出循环
                    break;
                }
            }
            MoveMali(key);  //移动人物
            PaintGame();    //刷新窗口
        }
    }
    //人物移动
    void MoveMali(int direct) {
        switch (direct)
        {
        case DEF_UP:
        {
            //判断移动是否出界
            if (m_y - DEF_STEP <= 0)  m_y = 0;
            else m_y -= DEF_STEP;

            m_nDirect = DEF_UP; //重新指定人物移动的方向
        }
        break;
        case DEF_DOWN:
        {
            //判断移动是否出界
            if (m_y + DEF_STEP >= 540) m_y = 540;
            else m_y += DEF_STEP;

            m_nDirect = DEF_DOWN;
        }
        break;
        case DEF_LEFT:
        {
            //判断移动是否出界
            if (m_x - DEF_STEP <= 0) m_x = 0;
            else m_x -= DEF_STEP;

            m_nDirect = DEF_LEFT;
        }
        break;
```

```
112             case DEF_RIGHT:
113             {
114                 //判断移动是否出界
115                 if (m_x + DEF_STEP >= 540) m_x = 540;
116                 else m_x += DEF_STEP;
117 
118                 m_nDirect = DEF_RIGHT;
119             }
120             break;
121             }
122 
123         }
124     };
125     int main() {
126         CGameMali gameMali;
127         gameMali.PaintGame();
128         gameMali.RunGame();
129         return 0;
130     }
```