面向对象特点1

以前使用C语言是基于过程的程序设计。而C++是基于面向对象的程序设计。、

凡是以类对象为基本构成单位的程序称为基于对象的程序。面向对象程序设计有4个主要特点：抽象、封装、继承、多态性。C++的类对象体现了抽象和封装的特性，在此基础上再利用继承和多态性，就成为真正的面向对象的程序设计。

oop的特性：1）抽象

2）封装和数据隐藏

3）继承

4）多态

5）代码可重用性

类的特性

1）抽象

2）封装

3）继承

4）多态

一.面向对象基础：

1. 面向对象的程序设计

把面向对象的思想应用到软件开发中，并指导开发维护软件

基本思想与现实生活中人类思考方式基本相似。

面向过程思想

面向对象思想(都是解决问题的思路)

面向过程思想：将复杂问题分解成很多简单问题，按照相应的流程去做

分解思想

面向对象思想：找到问题中所有的参入者（人、物体、逻辑等）将参入者进行分类

比如一个歌唱比赛： 1） 张三 工作人员

王五 工作人员

李丹 工作人员

2）李四 主持人

3）美美 参赛者

呵呵 参赛者

4） 音响 设备

灯 设

设计四个类 使用这些类创建对象

2．面向对象的概念

1）类

分类思维

类：全新的类型 类类型

结构体：类型 结构体类型

分类：将具有共同特征和行为的事物，归为一类。取一个名称，概称，总称

灯类

人类

汽车类

手机类

类：抽象的概念

2）对象 对象：具体的概念

一切皆对象（理解：你能看到的任何东西都是一个对象）

杯子 类

车 楼下的那辆红色的宝马车

人 吴宗明

对象：是类的实例

要操作，只能操作对象

类型 变量

int a;

类 对象

string name;

类：类型

对象：变量 --> 数据

3）抽象

面临大量事物

分类

提取共同特征和行为，作为一个类

属性：有什么 数据

行为：能干什么 函数

人

{

private:

姓名;身高;体重; 年龄;职业;

public:

说话();

打架();

谈情说爱();

};

4）封装 ：设置访问权限

5）继承 人

在既有类的基础上去派生出新的类

人 男人

灯 LED灯

电器 电脑

电脑 笔记本电脑

LED灯 是 灯 具有灯所具有的一切特征和行为

提高代码的复用性

提高类的开发效率

MP3类 NEWMAP3

6）多态 ：多种形态（本质相同，细节不同）

3. 面向对象软件开发步骤

1.面向对象分析 OOA

2.面向对象设计 OOD

3.面向对象编程 OOP

二．类（class）（类是C++对象的类型）

简单的说类就是数据类型。

1.类特性

C: 程序是由函数组成，问题 --> 函数来解决

C++: 程序是由对象组成，问题 --> 对象来解决 (对象去调用函数)

C++:

设计类 --->创建对象 --->用对象解决问题

2.由来

1. C++ struct

1.支持数据成员，也支持函数成员

2.成员没有访问权限，即外部可以直接访问

3.使用成员运算符 . 来访问成员

4.直接使用 类型名 表示结构体类型

2.C++ class

1.支持数据成员，也支持函数成员

2.成员有访问权限

public 公共的

protected 受保护的

private 私有的

默认为：private

3.使用成员运算符 . 来访问成员

4.直接使用 类型名 表示结构体类型

3.类声明

class 类名

{ 数据成员;

函数成员; };

class 类名

{ private:

成员;

protected:

成员;

public:

成员; };

Eg:class student

{private:

int age;

char name[20];

public:

void setName(char \*p);

void setAge(int n);

};

4.成员函数的定义

1）类中

class Point

{

int x;

int y;

public:

void setPos(int a, int b) // 特性：默认成为内联函数

{

x = a, y = b; // 建议：写在类外

}

void show();

};

2）类外

void Point::show()

{

cout << "[" << x << "," << y << "]" << endl;

}

格式:

返回值类型 类名::函数名(形参列表)

{……………..}

就表示这个函数是一个类的成员函数

通常：数据成员私有化，函数成员公有化

通过公有的函数成员去间接的访问数据成员

三.构造函数和析构函数

1．构造函数分类：

1.普通构造函数

2.复制构造函数

2.构造函数的理解：

int a = 20;

int a(20); // 构造函数

string b = "123";

string b("123"); // 构造函数

1).特殊的函数

2).创建对象时，自动调用 （一个对象只能调用一次构造函数）；只有在创建时，自动调用 (隐式调用)；构造函数式不能显式调用的（显式：写出来）（隐式：不用写出来，但也发生了）

3).如果我们没有自定义构造函数 ；系统会提供默认的构造函数；一旦自定义构造函数，那么这个默认的构造函数就不再提供了。

3.构造函数格式：

普通函数格式： 返回值类型 函数名（形参列表）;

构造函数格式： 类名（形参列表）;

1).构造函数名 必须与 类名相同

2).构造函数 没有返回值类型；没有返回值，返回值类型为：void； 没有返回值类型，连 void 也没有

4.构造函数的作用：

1).给对象一个标识符 （不太准确）

2).为对象的数据成员分配空间

3.)对数据成员初始化 (或者赋值) ：需要程序员自己写

5.构造函数的特性

1）构造函数，创建对象时自动调用

2)构造函数，不能显式调用

3)创建对象时，必须提供一个匹配的构造函数，否则无法创建对象

4)构造函数支持重载 :提供各种各样的构造函数来确保我们可以去满足多种不同的对象实例化形

5)构造函数支持参数默认值:如果同时存在函数重载 参数默认值，可能会出现调用的冲突

核心：要想对象成功创建，则必须提供一个合适的构造函数

理解：构造函数是对象调用的第一个函数，相当于一个门户函数

不仅考虑是空间的分配，重要的是要确保空间中有有效值

这样，后面对象再去调用其他函数，才能保证是正常的

6.初始化:表达式是在构造函数中，用来对数据成员进行初始化的

本质：表达式

理解：用来初始化的表达式

用来给对象的数据成员初始化的

分配空间的同时，写入数据：初始化

1).初始化表达式只能用于构造函数中

2).初始化表达式位于构造函数形参列表后，函数体前

形参列表后，加：间隔

point::point(int a, int b) : 初始化表达式1,初始化表达式2

{

}

3).初始化的顺序执行顺序不是由成员变量在初始化表中的顺序决定，而是由成员变量在类

中被声明时的顺序决定的

4).初始化表达式中的变量顺序最好与其被声明的顺序一致

7.构造函数重载

像其它函数一样，一个构造函数也可以被多次重载(overload)为同样名字的函数，但有不同的参数类型和个数。编译器会调用与在调用时刻要求的参数类型和个数一样的那个函数。实际上，当我们定义一个class而没有明确定义构造函数的时候，编译器会自动假设两个重载的构造函数 (默认构造函数"default constructor" 和复制构造函数"copy constructor")。如：

class CExample {public:  
 int a,b,c;  
 void multiply (int n, int m) { a=n; b=m; c=a\*b; }; };

没有定义构造函数，编译器自动假设它有以下constructor 成员函数：

Empty constructor

它是一个没有任何参数的构造函数，被定义为nop (没有语句)。它什么都不做。

CExample::CExample () { };

2.Copy constructor

它是一个只有一个参数的构造函数，该参数是这个class的一个对象，这个函数的功能是将被传入的对象（object）的所有非静态（no static）成员变量的值都复制给自身这个object。

CExample::CExample (const CExample& rv) {  
 a=rv.a; b=rv.b; c=rv.c;  
 }  
//**注意：**这两个默认构造函数（empty construction 和 copy constructor ）只有在没有其它构造函数被明确定义的情况下才存在。如果任何其它有任意参数的构造函数被定义了，这两个构造函数就都不存在了。在这种情况下，如果你想要有empty construction 和 copy constructor ，就必需要自己定义它们。如：

Class CRectangle {

int width, height;

public:

CRectangle ();

CRectangle (int,int);

int area (void) {return (width\*height);} };

CRectangle::CRectangle () {

width = 5;

height = 5;

}

CRectangle::CRectangle (int a, int b) {

width = a;

height = b;

}

int main () {

CRectangle rect (3,4);

CRectangle rectb;

cout << "rect area: " << rect.area() << endl;

cout << "rectb area: " << rectb.area() << endl;

} ///输出12 25

9.析构函数

析构函数必须与class同名，加水波号 (~) 前缀，必须无返回值。析构函数特别适用于当一个对象被动态分别内存空间，而在对象被销毁的时希望释放它所占用的空间的时候。

1.析构函数也是一个特殊的成员函数

2.当对象超过其作用域被撤销时，自动调用析构函数

构造函数：分配数据成员空间，用来对数据成员进行初始化

析构函数：不是释放数据成员空间

析构函数可以显式调用

块作用域: 在块中是有效的 ；在块中创建，块结束就撤销

3.析构函数是用来清理空间（堆区空间）（不是释放成员空间）

通常，在释放数据成员空间之前调用

4.析构函数形式：

1.函数名与类名相同，且前面加上 ~ 与构造函数区分

2.没有返回值类型

3.没有形参 -- 空的形参列表

格式： ~类名() {…………}

5.析构函数不能重载 1.析构函数不支持参数默认值2.析构函数只能有一个

6.如果没有自定义析构函数，编译器会提供;一个默认的空的函数体的析构函数;如果自定义了析构函数，默认的就不再提供

7.通常只要类的内部不出现堆区空间时，就使用默认的析构就行了;如果类的内部出现堆区空间，就必须自定义析构函数;通常指：指针成员 指向堆区空间;因为堆区空间：人为申请，只能人为释放;所以程序必须在析构函数中写相应的代码来释放

扩展：作用域：1.标识符的作用域

2. 名称的作用域:这个名称在什么区域可以使用;变量名 函数名 结构体名 类型别名

标识符 起作用的区域 ;可使用的区域,根据声明的位置以及自身的特性决定了

这个名称的作用域

1).块作用域 {int age; age = 20; } age = 80;

2.文件作用域

3).函数作用域

4).函数原型作用域

10.析构函数的调用顺序 与 构造函数调用的顺序相反

三.类的指针

1.对象空间的起始地址就是对象的指针。

如：class time

{ int hour;

int minute;

int sec;

void get\_time();

}; time \*p; time t1; p=&t1;

//定义p为指向time类对象的指针变量，t1为类对象，t1的起始地址给p;

指向对象成员的指针：对象有地址，存放对象的起始地址的指针变量就是指向对象的指针变量。对象中的成员也有地址，存放对象成员地址的指针变量就是指向对象成员的指针变量。

1.指向对象数据成员的指针

格式：数据类型名 \*指针变量名

2.指向对象成员函数的指针

指向普通函数的指针变量发法： 类型名（\*指针变量名）（ 参数列表）；

如：void(\*p)();//p是指向void型函数的指针变量 p=fun; //将fun的入口地址赋给指针变量p(\*p)();//调用函数fun

指向公用成员函数的指针变量形式： 数据类型名（类名::\*指针变量名）（参数列表）

如：void(time::\*p1)();//p1为指向time类中公用成员函数的指针变量

指针变量指向一个公用成员函数的一般形式：指针变量名=&类名::成员函数名

如:void (time::\*p2)(); p2=&time::get\_time();

2.this 指针

在每个成员函数中都包含一个特殊的指针，这个指针的名字固定成为this.this它是指向本类对象的指针，它的值是当前被调用的成员函数所在的对象的起始地址。

1）.成员函数中，通常会有一个隐含的 this 指针

它指向本类的对象 （那个调用函数的对象;由于函数成员被所有该类对象共同使用; 如何在成员函数中去访问不同对象的数据空间;利用成员函数中一个隐含的this指针（本类对象指针）

谁调用成员函数，this接收该对象的地址

this->成员名 的方式就可以访问这个对象的成员

a.show(); // this = &a;

show中 radius == this->radius == a.radius

通常 this 可以省略不写;但是也可以显式写出来;当成员函数内部出现了与成员名相同的名称时，则必须使用 this->成员名.

四：把数据定义为常量（公用数据的保护）

1.常量成员

数据成员可以由const修饰，这样，一经初始化，该数据成员便具有“只读属性”，在程序中无法对

其值修改。事实上，在构造函数体内或复制构造函数体内赋值const数据成员是非法的。const数据成 员只能通过构造函数中的初始化表达式进行初始化。

注意：所有的构造函数中的初始化表达式都要对这个const数据成员进行初始化.int const z;

1. 常对象：定义对象时加关键字const,指定对象为常对象。常对象要有初值。

格式：1.类名const 对象名[(实参表)]2.const 类名 对象名[(实参表)]如:time const t1(12,34,56);常对象t1.

注：1.定义的常对象，则通过该对象只能调用它的常成员函数，不能调用该对象的普通成员函数。常成员函数是常对象唯一的对外接口。怎么引用常对象的数据成员：将成员函数声明为const即可。如：void get\_time() const;//常成员函数。

2).常成员函数可以访问常对象中的数据成员，但依然不充许修改常对象中数据成员的值。

1. 常对象成员
2. 常数据成员：构造函数只能用参数初始化表对常数据成员进行初始化。如：类中定义constint hour; 在类外定义构造函数写的形式：提么time::time(int h):hour(h){ }
3. 常成员函数:如果将成员函数声明为常成员函数，则只能引用本类中的数据成员，而不能修改它们。声明常成员函数的形式：类型名 函数名(参数表)const
4. 指向对象的常指针

将指针变量声明为const型，这样的指针变量始终保持为处值，不能改变，即指向不变。声明形式：类名 \*const 指针变量名

如：time \*const p1=&t1;

1. 指向常对象的指针变量

定义指向常变量的指针变量的一般形式：const 类型名 \*指针变量名；

说明：1）.如果一个变量已被声明为常变量，只能用指向常变量的指针变量指向它，而不能用一般的（指向非const型便来变量的）指针变量去指向它。如：const char c[]=”boy”;

Const char \*p; p=c; char \*p2=c;//非法

1. 指向常变量的指针变量除了可以指向常变量外，还可以指向未被声明为const的变量。此时不能通过此指针变量改变该变量的值，如：char c1=’a’;const char \*p; p=&c1; \*p=’b’;//非法 c1=’b;//合法’
2. 对象的常引用

小结：C++经常使用常指针和常引用作函数的参数，可以保证数据的安全

|  |  |
| --- | --- |
| 形式 | 含义 |
| Time const t;  Void time ::fun()const;  Time \*const p;  Const time \*p;  Const time &t1=t; | t是常对象，其值在任何情况下都不能改变  fun是time类中的常成员函数，可以引用，但不能修改本类中的数据成员  p是指向time类对象的常指针变量，p的值（p指向）不能改变  p是指向time类常对象的指针变量，p指向的类对象的值不能通过p来改变  t1是time类对象t的引用，二者指向同一存储空间，t的值不能改变 |

五.对象的复制和赋值和静态成员

1.对象的复制：对象得复制则是从无到有地建立一个新对象，并使它与一个已有的对象完全相同（包括对象的结构和成员的值）

形式1).类名 对象2(对象1); 2).类名 对象1=对象2；

普通构造函数与复制构造函数区别：1.形式上：类名（形参列表）//普通构造函数的声明，如box（int h,int w,int len）; 类名（类名&对象名）//复制构造函数的声明如：box（box&b）;2.在建立对象时，实参类型不同。系统会根据实参的类型决定调用普通构造函数或复制构造函数。 box box1(12,34,56);//调用普通构造函数，实参为整数。box box(box1);//调用复制构造函数，实参是对象名。

什么情况下被调用：普通构造函数在程序中建立对象时被调用。复制构造函数在已有对象复制一个新对象时被调用。

需要复制构造函数的情况

1. 程序中需要新建一个对象，并用另一个同类的对象对它初始化。
2. 当函数的参数为类的对象时。在调用函数时需要将实参对象完整地传递给形参，也就是要建立一个实参的拷贝，这就是拿实参复制一个形参，系统调用复制构找函数来实现。
3. 函数返回值是类的对象。在函数调用完毕将返回值带回函数调用处时。需要将函数中的对象复制一个临时对象并传给该函数的调用处。.

2.对象的赋值

如果对一个类定义了两个或者多个对象,则这些同类的对象之间可以互相的赋值,或者一个对象的值可以赋给另一个同类的对象.(这里的值指对象中所有数据成员的值).格式:对象名1=对象名2;(这两个对象必须同类)

3.静态成员和静态成员函数

1).一个class 可以包含静态成员(static members)，可以是数据，也可以是函数。静态成员与全域变量具有相同的属性，但它享有类(class)的范围。因此，根据ANSI-C++ 标准，为了避免它们被多次重复声明，在class的声明中只能够包括static member的原型(声明)，而不能够包括其定义(初始化操作)。为了初始化一个静态数据成员，我们必须在class之外(在全域范围内)，包括一个正式的定义。如：

Class cd{

Public: int n;

} ;

Int cd ::n=10;

//它其实是一个全域变量。唯一的不同是它的名字跟在class的后面。就像我们会在class中包含static数据一样，我们也可以使它包含static 函数。它们表示相同的含义：static函数是全域函数(global functions)，但是像一个指定class的对象成员一样被调用。它们只能够引用static 数据，永远不能引用class的非静态(nonstatic)成员。它们也不能够使用关键字this，因为this实际引用了一个对象指针，但这些 static函数却不是任何object的成员，而是class的直接成员。

C++允许使用static（静态存储）修饰数据成员，这样的成员在编译时就被创建并初始化的（与之相比，对象是在运行时被创建的），且其实例只有一个，被所有该类的对象共享，就像住在同一宿舍里的同学共享一个房间号一样。静态数据成员可用于同类对象间通信；静态数据成员最好不要使用 this 来访问，应该使用 类名::成员名： eg: Point::x

静态数据成员和以前学习的静态变量一样，程序执行时，该成员已经存在，一直到程序结束，任何对象都可对其进行访问。静态数据成员的初始化必须在类申明之外进行，且不再包含static关键字，格式如下：

类型 类名::变量名 = 初始化表达式; //普通变量

类型 类名::对象名(构造参数); //对象变量

访问静态数据成员：在类外可以通过对象名访问公用的静态数据成员，也可以通过类名引用静态数据成员。

2).静态成员函数: 静态成员函数是类的一部分而不是对象的一部分。

成员函数也可以定义成静态的，与静态成员变量一样，系统对每个类只建立一个函数实体，该实体为该类的所有对象共享。静态成员函数与普通成员函数都只有一个实体，被所有的对象共同使用；

普通成员函数，必须通过对象去调用；通过this指针保存这个对象的地址，去访问这个对象的空间；既可以访问普通数据成员，也可以访问静态成员；

静态成员函数，可以不通过对象就调用，也可以通过对象调用；只能访问静态成员（数据成员 和 函数成员）不访问非静态成员；类提供了很多功能；正常情况下，都是先创建一个对象，然后让对象去调用对应的函数；特殊情况下，可以不创建对象，直接使用静态成员函数，实现某些功能。二、特殊的函数成员

访问格式：要用类名和域运算符如：box::volume();

目标：普通函数 分类：

void show();

void put() const ; // const成员函数// 只读成员函数

static void hehe(); // 静态成员函数

inline void sss(); // 内联成员函数

六，类之间的关系

1. 友元

1). 友元的理解;友好的单元;A把B当友元;类的成员，要给别人访问，为不同的“人”

设置不同的权限;正常情况下，类外部访问类内部的资源都要;通过权限;如果类内部将某个xxx声明为该类的友元;表明这个xxx可以直接访问该类的成员（包括私有）无需权限。

2).友元的声明: friend xxxxx; 注意：必须放在类的内部

class A{ int x;

int add(); // 成员函数声明 add 属于 A类的

friend int sub(); // 友元声明 sub 只是 A类的友元

friend B; };

int A::add()

{ x = 10; // this->x }

int sub(){ A a; a.x=70}

友元的形式：1).友元普通函数2).友元成员函数3).友元类

友元函数：

1.将普通函数声明为友元函数如：class time{public:

time(int,int,int); friend void display(time &);//声明display函数为time类的友元函数 private：int hour;int minute;int sec;}；

2.友元成员函数：friend函数不仅可以是一般函数（非成员函数），而且也可以是另一个类中的成员函数。如：class date; class time { public:time (int,int,int);void display(date &)//声成员函数，形参是date类对象的引用;private:int hour,minute,sec;} class date{ public:date(int,int,int); friend void time::display(date &);//声明time中的display函数为本类的友元成员函数；private:int month,day,year;}

1. 友元类：不仅可以将一个函数声明为一个类的“朋友”，而且可以将一个类（如B类）声明为另一个类（如A类）的“朋友”，这时B类就是A类的友元类。友元类B中的所有函数都是A类的友元函数，可以访问A类中的所有成员。
2. 类模板

有两个或者多个类，其功能是相同的，仅仅是数据类型不同。为了减少重复工作，可以使用类模板；声明一个通用的类模板，它可以有一个或多个虚拟的类型参数。

1.声明类模板时要加一行：template<class 类型参数名>

Template是模板，是声明类的关键字，（类模板是类的抽象，类是类模板的实例）

类模板：template<class numtype> //声明一个模板，虚拟类型名为numtype

Class compare{ public: //类模板名为compare

Compare(numtype a,numtype b){ x=a;y=b} //定义构造函数

numtype max(){return(x>y)?x:y;}

numtype min(){return(x<y)?x:y;}

private: numtype x,y;}

格式：类模板名<实际类型名> 对象名（参数列表）如：compare<int> cmp(4,7)；

1.用类模板定义对象时格式：1.类模板名<实际类型名>对象名如：compare<int> cmp；2.类模板名<实际类型名>对象名（实参表）如：compare<int> cmp(4,7)；

2.如果在类模板外定义成员函数，应写成类模板形式：template<class 虚拟类型参数>

函数类型 类模板名<虚拟类型参数>::成员函数名（函数形参列表）{……}如：

template<class numtype>

numtype compare<numtype>::min(){};

.