# 复制构造函数的调用场合

复制构造函数，在利用复制构造来创建新的对象的时候被自动调用，用来对新对象进行初始化。

A a1;

A a2(a1);

1 明确的以复制构造的方式来创建新的对象

A a1;

A a2(a1);

A a3=a1;

a2 = a1; 对象间的赋值，C++默认就支持对象赋值，不会调用复制构造函数。

默认的赋值，也是一种完全复制。

2 当函数的形参是对象，实参也是同类的对象，进行参数传递的时候，形参是通过复制构造的方式产生的，这个复制构造接收的参数就是实参对象。

所以，当传递对象的时候，推荐使用引用的方式。

3 当函数返回值是对象，在函数返回对象时，会利用复制构造创建一个临时的对象。

这个性质和具体的编译器有关，有些编译器进行了优化，不再调用复制构造函数。

#include <iostream>

using namespace std;

class A{

public:

A(){

}

A(A & r){

cout<<"A的复制构造函数"<<endl;

}

};

void f(A &a){

}

A g(){

A a;

return a;

}

int main(){

A a1;

//A a2(a1);

//A a2=a1;

//A a2;

//a2 = a1;

//f(a1);

a1 = g();

}

//4\_2.cpp

#include <iostream>

using namespace std;

class Point { //Point 类的定义

public: //外部接口

Point(int xx = 0, int yy = 0) { //构造函数

x = xx;

y = yy;

}

Point(Point &p); //拷贝构造函数

int getX() {

return x;

}

int getY() {

return y;

}

private: //私有数据

int x, y;

};

//成员函数的实现

Point::Point(Point &p) {

x = p.x;

y = p.y;

cout << "Calling the copy constructor" << endl;

}

//形参为Point类对象的函数

void fun1(Point p) {

cout << p.getX() << endl;

}

//返回值为Point类对象的函数

Point fun2() {

Point a(1, 2);

return a;

}

//主程序

int main() {

Point a(4, 5); //第一个对象A

Point b = a; //情况一，用A初始化B。第一次调用拷贝构造函数

cout << b.getX() << endl;

fun1(b); //情况二，对象B作为fun1的实参。第二次调用拷贝构造函数

b = fun2(); //情况三，函数的返回值是类对象，函数返回时，调用拷贝构造函数

cout << b.getX() << endl;

return 0;

}

# 构造函数与类型转换（补充）

类型转换

显式转换 强制转换： (类型名) 变量名 类型名（变量名）

d = (double)i;

i = int(d);

隐式转换 自动转换

d = i; i = d;

前提是不同类型之间是可以兼容的

Point(2,2); 创建一个无名的对象，是一个临时的对象

point p1;

p1 = Point(4);

通过构造函数创建了一个临时对象，在赋值给p1.

但是，从类型转换的角度来说，也可以认为是将整形转换成立Point类型。

推论：如果一个类的构造函数支持只传递一个参数就可以调用，则相当于这个参数可以转换为该类的类型。

#include <iostream>

using namespace std;

class Point{

private:

int x,y;

public:

Point(int x1=0,int y1=0){

x=x1;

y=y1;

}

};

int main(){

int i = 4;

//Point p(1,2);

//Point p(2);

//Point(2,2);

Point p1;

//p1 = Point(4);

//p1 = (Point) i;

p1 = i;

}

explicit Point(int x1=0,int y1=0){

x=x1;

y=y1;

}

explicit 用来修饰构造函数，作用是取消隐式的类型转换。

# 类的组合

一个大的对象里面有时会包含很多小的对象。

这样的组合关系，在面向对象的编程里面可以被很好的支持。

类的组合：一个大的类里面的数据成员是其他类的对象，这个语法现象被称为是类的组合，类的组合模拟了真实世界中整体和局部的组合关系。

组合类（大的类，整体类）

内嵌对象：在大的类里面定义的其他类的对象。

内嵌类：内嵌对象所对应的类。

使用组合类时候注意事项：

1 如何对内嵌对象进行初始化？

思路是通过内嵌类的构造函数来初始化内嵌对象，如何利用内嵌类的构造来初始化内嵌对象？

Point p1(x1,y1)

public:

Line(int x1,int y1,int x2,int y2):p1(x1,y1),p2(x2,y2)

{

//p1.x = x1;

}

1组合类构造函数的初始化列表，在初始化列表里面通过构造函数对内嵌对象进行初始化。

2在创建组合类的对象的时候，一定是先创建它所包含的内嵌对象。意味着一定会调用内嵌类的构造函数。通常通过初始化列表来完成。

3 当内嵌对象有无参的构造函数的时候，初始化列表中的语言可以省略。初始化列表里面没有对内嵌对象显式初始化，则编译器自动调用对应的无参构造函数来完成初始化。

#include <iostream>

using namespace std;

class Point{

private:

int x,y;

public:

explicit Point(int x1,int y1){

x=x1;

y=y1;

cout<<"Point的构造函数被调用了"<<endl;

}

};

class Line{

private:

Point p1,p2;

public:

Line(int x1,int y1,int x2,int y2):p1(x1,y1),p2(x2,y2)

{

//p1.x = x1;

}

};

int main(){

Line l(1,2,3,4);

}

**组合类的复制构造函数的一般写法**

Line(Line &l):p1(l.p1),p2(l.p2){

}

**使用组合类时的析构函数的调用过程。**

组合类的对象消亡的时候，除了会调用组合类的析构函数，还会自动调用内嵌类的析构函数。并且调用顺序和构造函数的调用顺序相反。

**内嵌对象之间的构造函数的调用顺序？根据内嵌对象在类中的定义顺序来决定的。**

Class A{

B b;

C c;

A():c(1),b(2){

}

}