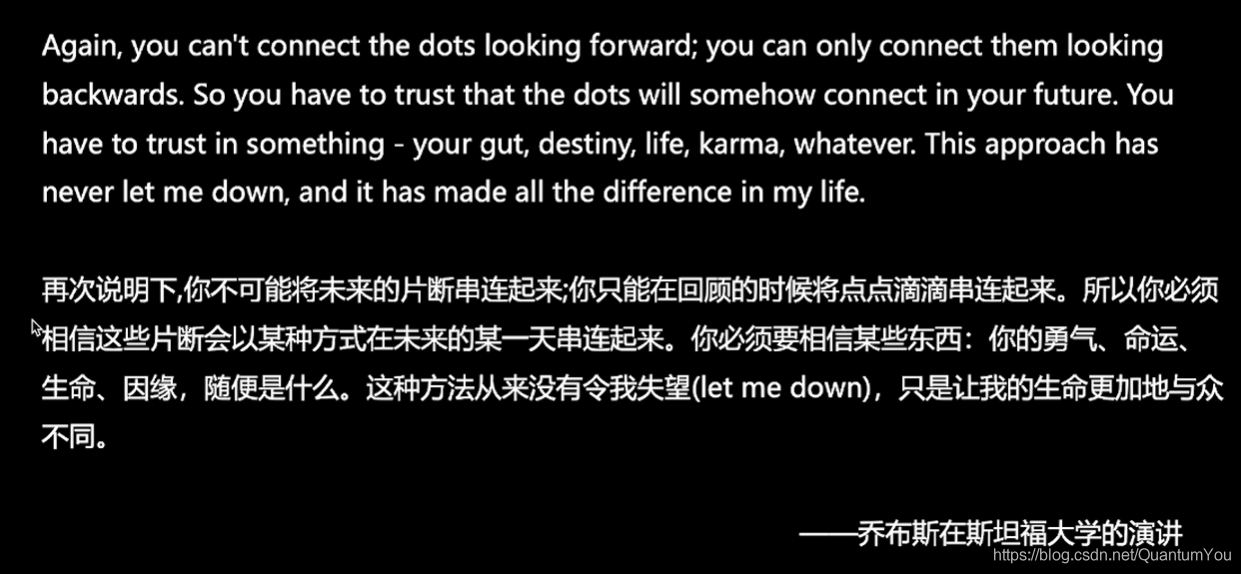
* 我希望你没有刻意为追求一个数字而生活，我希望你找到了真正的价值所在
* 你一定要**坚持**做自己，**静下心**来做自己**喜欢**的事，然后把自己交给**命运**

@[toc]

  
**类和对象续**

# 对象的初始化和清理

* C++中的面向对象来源于生活，每个对象也都会有初始设置以及 对象销毁前的清理数据的设置。

## 构造函数和析构函数

* C++利用了**构造函数**和**析构函数**解决上述问题，这两个函数将会被编译器自动调用，完成对象初始化和清理工作。

对象的初始化和清理工作是编译器强制要我们做的事情，因此如果**我们不提供构造和析构，编译器会提供**

**编译器提供的构造函数和析构函数是空实现。**

* 构造函数：主要作用在于创建对象时为对象的成员属性赋值，构造函数由编译器自动调用，无须手动调用。
* 析构函数：主要作用在于对象**销毁前**系统自动调用，执行一些清理工作。

**构造函数语法：**类名(){}

1. 构造函数，没有返回值也不写void
2. 函数名称与类名相同
3. 构造函数可以有参数，因此可以发生重载
4. 程序在调用对象时候会自动调用构造，无须手动调用,而且只会调用一次

**析构函数语法：** ~类名(){}

1. 析构函数，没有返回值也不写void
2. 函数名称与类名相同,在名称前加上符号 ~
3. 析构函数不可以有参数，因此不可以发生重载
4. 程序在对象销毁前会自动调用析构，无须手动调用,而且只会调用一次

class Person  
{  
public:  
 //构造函数  
 Person()  
 {  
 cout << "Person的构造函数调用" << endl;  
 }  
 //析构函数  
 ~Person()  
 {  
 cout << "Person的析构函数调用" << endl;  
 }  
  
};  
  
void test01()  
{  
 Person p;  
}  
  
int main() {  
   
 test01();  
  
 system("pause");  
  
 return 0;  
}

## 构造函数的分类及调用

**两种分类方式**：

* 按参数分为： 有参构造和无参构造
* 按类型分为： 普通构造和拷贝构造

**三种调用方式**：

* 括号法
* 显示法
* 隐式转换法

**示例：**

//1、构造函数分类  
// 按照参数分类分为 有参和无参构造 无参又称为默认构造函数  
// 按照类型分类分为 普通构造和拷贝构造  
  
class Person {  
public:  
 //无参（默认）构造函数  
 Person() {  
 cout << "无参构造函数!" << endl;  
 }  
 //有参构造函数  
 Person(int a) {  
 age = a;  
 cout << "有参构造函数!" << endl;  
 }  
 //拷贝构造函数  
 Person(const Person& p) {  
 age = p.age;  
 cout << "拷贝构造函数!" << endl;  
 }  
 //析构函数  
 ~Person() {  
 cout << "析构函数!" << endl;  
 }  
public:  
 int age;  
};  
  
//2、构造函数的调用  
//调用无参构造函数  
void test01() {  
 Person p; //调用无参构造函数  
}  
  
//调用有参的构造函数  
void test02() {  
  
 //2.1 括号法，常用  
 Person p1(10);  
 //注意1：调用无参构造函数不能加括号，如果加了编译器认为这是一个函数声明  
 //Person p2();  
  
 //2.2 显式法  
 Person p2 = Person(10);   
 Person p3 = Person(p2);  
 //Person(10)单独写就是匿名对象 当前行结束之后，马上析构  
  
 //2.3 隐式转换法  
 Person p4 = 10; // Person p4 = Person(10);   
 Person p5 = p4; // Person p5 = Person(p4);   
  
 //注意2：不能利用 拷贝构造函数 初始化匿名对象 编译器认为是对象声明  
 //Person p5(p4);  
}  
  
int main() {  
  
 test01();  
 //test02();  
  
 system("pause");  
  
 return 0;  
}

**注意事项：**

* **注意1**： Person p2() ; 调用无参构造函数不能加括号，如果加了编译器认为这是一个函数声明 ;
* **注意2** ： 拷贝函数构造

Person(const Person& p) {  
 age = p.age;  
}

* **注意3** ：显示法中 Person p2 = Person(10); 的 Person(10) 相当于匿名对象，在构造完之后将会被销毁
* **注意4**： 不能利用 拷贝构造函数 初始化匿名对象 Person p5(p4);编译器认为是对象声明 Person (p5) === Person p4

[视频讲解链接 21:15](https://www.bilibili.com/video/BV1et411b73Z?p=107&spm_id_from=pageDriver)

## 拷贝构造函数调用时机

C++中拷贝构造函数调用时机通常有三种情况

* 使用一个已经创建完毕的对象来初始化一个新对象
* 值传递的方式给函数参数传值
* 以值方式返回局部对象

**示例：**

class Person {  
public:  
 Person() {  
 cout << "无参构造函数!" << endl;  
 mAge = 0;  
 }  
 Person(int age) {  
 cout << "有参构造函数!" << endl;  
 mAge = age;  
 }  
 Person(const Person& p) {  
 cout << "拷贝构造函数!" << endl;  
 mAge = p.mAge;  
 }  
 //析构函数在释放内存之前调用  
 ~Person() {  
 cout << "析构函数!" << endl;  
 }  
public:  
 int mAge;  
};  
  
//1. 使用一个已经创建完毕的对象来初始化一个新对象  
void test01() {  
  
 Person man(100); //p对象已经创建完毕  
 Person newman(man); //调用拷贝构造函数  
 Person newman2 = man; //拷贝构造  
  
 //Person newman3;  
 //newman3 = man; //不是调用拷贝构造函数，赋值操作  
}  
  
//2. 值传递的方式给函数参数传值  
//相当于Person p1 = p;  
void doWork(Person p1) {}  
void test02() {  
 Person p; //无参构造函数  
 doWork(p);  
}  
  
//3. 以值方式返回局部对象  
Person doWork2()  
{  
 Person p1;  
 cout << (int \*)&p1 << endl;  
 return p1;  
}  
  
void test03()  
{  
 Person p = doWork2();  
 cout << (int \*)&p << endl;  
}  
  
  
int main() {  
  
 //test01();  
 //test02();  
 test03();  
  
 system("pause");  
  
 return 0;  
}

### 构造函数调用规则

默认情况下，c++编译器至少给一个类添加3个函数

1．默认构造函数(无参，函数体为空)

2．默认析构函数(无参，函数体为空)

3．默认拷贝构造函数，对属性进行值拷贝

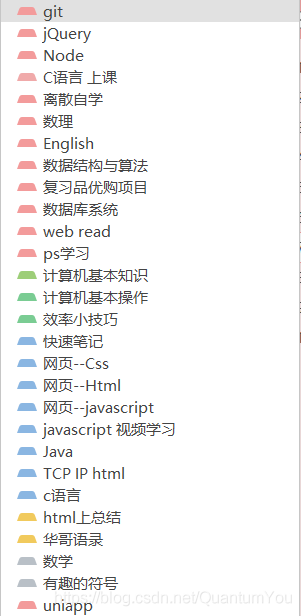
构造函数调用规则如下：

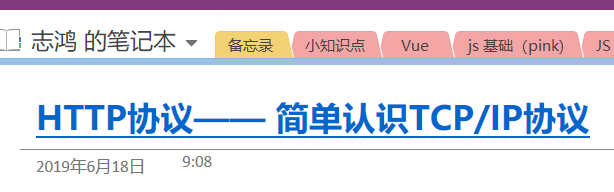
* 如果用户定义有参构造函数，c++不在提供默认无参构造，但是会提供默认拷贝构造
* 如果用户定义拷贝构造函数，c++不会再提供其他构造函数

示例：

class Person {  
public:  
 //无参（默认）构造函数  
 Person() {  
 cout << "无参构造函数!" << endl;  
 }  
 //有参构造函数  
 Person(int a) {  
 age = a;  
 cout << "有参构造函数!" << endl;  
 }  
 //拷贝构造函数  
 Person(const Person& p) {  
 age = p.age;  
 cout << "拷贝构造函数!" << endl;  
 }  
 //析构函数  
 ~Person() {  
 cout << "析构函数!" << endl;  
 }  
public:  
 int age;  
};  
  
void test01()  
{  
 Person p1(18);  
 //如果不写拷贝构造，编译器会自动添加拷贝构造，并且做浅拷贝操作  
 Person p2(p1);  
  
 cout << "p2的年龄为： " << p2.age << endl;  
}  
  
void test02()  
{  
 //如果用户提供有参构造，编译器不会提供默认构造，会提供拷贝构造  
 Person p1; //此时如果用户自己没有提供默认构造，会出错  
 Person p2(10); //用户提供的有参  
 Person p3(p2); //此时如果用户没有提供拷贝构造，编译器会提供  
  
 //如果用户提供拷贝构造，编译器不会提供其他构造函数  
 Person p4; //此时如果用户自己没有提供默认构造，会出错  
 Person p5(10); //此时如果用户自己没有提供有参，会出错  
 Person p6(p5); //用户自己提供拷贝构造  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}







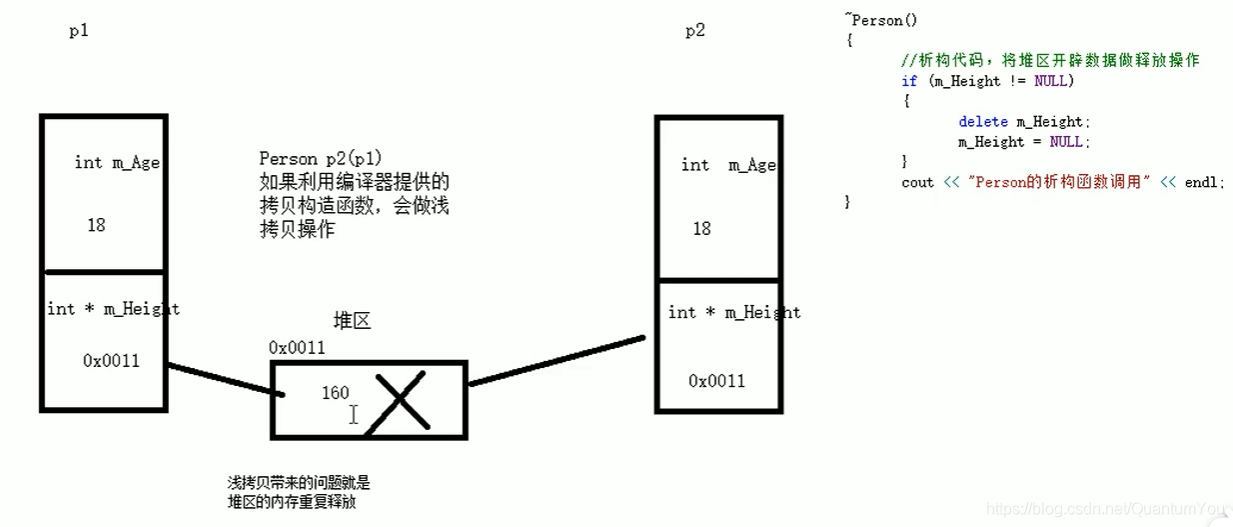
### 深拷贝与浅拷贝

* 浅拷贝：简单的赋值拷贝操作
* 深拷贝：在堆区重新申请空间，进行拷贝操作

**示例：**

class Person {  
public:  
 //无参（默认）构造函数  
 Person() {  
 cout << "无参构造函数!" << endl;  
 }  
 //有参构造函数  
 Person(int age ,int height) {  
   
 cout << "有参构造函数!" << endl;  
  
 m\_age = age;  
 m\_height = new int(height);  
   
 }  
 //拷贝构造函数   
 Person(const Person& p) {  
 cout << "拷贝构造函数!" << endl;  
 //如果不利用深拷贝在堆区创建新内存，会导致浅拷贝带来的重复释放堆区问题  
 m\_age = p.m\_age;  
 m\_height = new int(\*p.m\_height);  
   
 }  
  
 //析构函数  
 ~Person() {  
 cout << "析构函数!" << endl;  
 if (m\_height != NULL)  
 {  
 delete m\_height;  
 }  
 }  
public:  
 int m\_age;  
 int\* m\_height;  
};  
  
void test01()  
{  
 Person p1(18, 180);  
  
 Person p2(p1);  
  
 cout << "p1的年龄： " << p1.m\_age << " 身高： " << \*p1.m\_height << endl;  
  
 cout << "p2的年龄： " << p2.m\_age << " 身高： " << \*p2.m\_height << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：如果属性有在堆区开辟的，一定要自己提供拷贝构造函数，防止浅拷贝带来的问题



* 由先进后出可知，p2 先释放内存，然后p1 再释放内存 （非法操作） ---- > 这是浅拷贝出现的问题（运用深拷贝 解决）**如下代码操作**

Person(const Person& p) {  
 cout << "拷贝构造函数!" << endl;  
 //如果不利用深拷贝在堆区创建新内存，会导致浅拷贝带来的重复释放堆区问题  
 m\_age = p.m\_age;  
 m\_height = new int(\*p.m\_height);  
   
 }

* 其中 new int (\*p.m\_height) 为开辟一个新堆

## 初始化列表

**作用：**

C++提供了初始化列表语法，用来初始化属性

**语法：**构造函数()：属性1(值1),属性2（值2）... {}

**示例：**

class Person {  
public:  
  
 ////传统方式初始化  
 //Person(int a, int b, int c) {  
 // m\_A = a;  
 // m\_B = b;  
 // m\_C = c;  
 //}  
  
 //初始化列表方式初始化  
 Person(int a, int b, int c) :m\_A(a), m\_B(b), m\_C(c) {}  
 void PrintPerson() {  
 cout << "mA:" << m\_A << endl;  
 cout << "mB:" << m\_B << endl;  
 cout << "mC:" << m\_C << endl;  
 }  
private:  
 int m\_A;  
 int m\_B;  
 int m\_C;  
};  
  
int main() {  
  
 Person p(1, 2, 3);  
 p.PrintPerson();  
  
  
 system("pause");  
  
 return 0;  
}

### 类对象作为类成员

* C++类中的成员可以是另一个类的对象，我们称该成员为 对象成员

例如：

class A {}  
class B  
{  
 A a；  
}

* B类中有对象A作为成员，A为对象成员
* 那么当创建B对象时，A与B的构造和析构的顺序是谁先谁后？

构造的顺序是 ：先调用对象成员的构造，再调用本类构造

析构顺序与构造相反

**示例：**

class Phone  
{  
public:  
 Phone(string name)  
 {  
 m\_PhoneName = name;  
 cout << "Phone构造" << endl;  
 }  
  
 ~Phone()  
 {  
 cout << "Phone析构" << endl;  
 }  
  
 string m\_PhoneName;  
  
};  
  
  
class Person  
{  
public:  
  
 //初始化列表可以告诉编译器调用哪一个构造函数  
 Person(string name, string pName) :m\_Name(name), m\_Phone(pName)  
 {  
 cout << "Person构造" << endl;  
 }  
  
 ~Person()  
 {  
 cout << "Person析构" << endl;  
 }  
  
 void playGame()  
 {  
 cout << m\_Name << " 使用" << m\_Phone.m\_PhoneName << " 牌手机! " << endl;  
 }  
  
 string m\_Name;  
 Phone m\_Phone;  
  
};  
void test01()  
{  
 //当类中成员是其他类对象时，我们称该成员为 对象成员  
 //构造的顺序是 ：先调用对象成员的构造，再调用本类构造  
 //析构顺序与构造相反  
 Person p("张三" , "苹果X");  
 p.playGame();  
  
}  
  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

### 静态成员

* 静态成员就是在成员变量和成员函数前加上关键字static，称为静态成员

静态成员分为：

* 静态成员变量
  + 所有对象共享同一份数据
  + 在编译阶段分配内存
  + 类内声明，类外初始化
* 静态成员函数
  + 所有对象共享同一个函数
  + 静态成员函数只能访问静态成员变量

**示例1** : 静态成员变量

class Person  
{  
   
public:  
  
 static int m\_A; //静态成员变量  
  
 //静态成员变量特点：  
 //1 在编译阶段分配内存  
 //2 类内声明，类外初始化  
 //3 所有对象共享同一份数据  
  
private:  
 static int m\_B; //静态成员变量也是有访问权限的  
};  
int Person::m\_A = 10;  
int Person::m\_B = 10;  
  
void test01()  
{  
 //静态成员变量两种访问方式  
  
 //1、通过对象  
 Person p1;  
 p1.m\_A = 100;  
 cout << "p1.m\_A = " << p1.m\_A << endl;  
  
 Person p2;  
 p2.m\_A = 200;  
 cout << "p1.m\_A = " << p1.m\_A << endl; //共享同一份数据  
 cout << "p2.m\_A = " << p2.m\_A << endl;  
  
 //2、通过类名  
 cout << "m\_A = " << Person::m\_A << endl;  
  
  
 //cout << "m\_B = " << Person::m\_B << endl; //私有权限访问不到  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**示例2** : 静态成员函数

class Person  
{  
  
public:  
  
 //静态成员函数特点：  
 //1 程序共享一个函数  
 //2 静态成员函数只能访问静态成员变量  
   
 static void func()  
 {  
 cout << "func调用" << endl;  
 m\_A = 100;  
 //m\_B = 100; //错误，不可以访问非静态成员变量  
 }  
  
 static int m\_A; //静态成员变量  
 int m\_B; //   
private:  
  
 //静态成员函数也是有访问权限的  
 static void func2()  
 {  
 cout << "func2调用" << endl;  
 }  
};  
int Person::m\_A = 10;  
  
  
void test01()  
{  
 //静态成员变量两种访问方式  
  
 //1、通过对象  
 Person p1;  
 p1.func();  
  
 //2、通过类名  
 Person::func();  
  
  
 //Person::func2(); //私有权限访问不到  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

注意事项：

* 在测试中调用类（class Person）中的函数fun()  
  方法一 ： Person p ; p.fun() ;   
  方法二： Person :: fun() ; (不需要创建对象，程序共享同一个函数)