目录

[1 内存分区模型 3](#_Toc129506147)

[1-1 代码区 3](#_Toc129506148)

[1-2 全局区 3](#_Toc129506149)

[1-3 栈区 3](#_Toc129506150)

[1-4 堆区 4](#_Toc129506151)

[2 三大特性 5](#_Toc129506152)

[2-1 封装 5](#_Toc129506153)

[构造与析构 5](#_Toc129506154)

[类内静态成员 7](#_Toc129506155)

[this指针 7](#_Toc129506156)

[友元 9](#_Toc129506157)

[2-2 继承 9](#_Toc129506158)

[2-3 多态 9](#_Toc129506159)

1 内存分区模型

不同区域存放的数据，赋予不同的生命周期给我们更大的灵活编程

在程序编译后，生成exe可执行程序，未执行该程序前分为两个区域，代码区和全局区

1-1 代码区

存放函数体的二进制代码，由操作系统进行管理

存放CPU执行的机器指令

代码区是共享的，共享的目的是对于频繁被执行的程序，只需要在内存中有一份代码即可

代码区是只读的，使其只读的原因是防止程序意外的修改了它的指令

1-2 全局区

存放全局变量和静态变量（static）以及常量

全局变量和静态变量存放在此

全局区还包含了常量区，字符常量和其它常量也存在此

该区域的数据在程序结束后由操作系统回收

|  |  |
| --- | --- |
| 不在全局区中 | 全局区 |
| 局部变量  const修饰的局部变量（局部常量） | 全局变量  静态变量（static关键字）  字符串常量  const修饰的全局变量（全局常量） |

1-3 栈区

由编译器自动分配释放，存放函数的参数值（形参），局部变量等

注意：不要返回局部变量的地址，会由编译器自动释放

1-4 堆区

由程序员分配和释放，若程序员不释放，程序结束时由操作系统回收

主要利用new在堆区开辟内存，释放内存使用delete

释放数组的时候，要加 [ ] （delete[] arr;）

|  |  |
| --- | --- |
| 栈区 | 堆区 |
| int \* func() {  int a = 10;  return &a;  } | int \* func() {  int \*p = new int(10);  int \*arr = new int[10];  return p;  }  int main() {  int \*p = func();  delete p;  } |
| 使用函数返回的地址时非法操作 | 返回堆区存放数据的地址  释放后再调用为非法操作 |

2 三大特性

2-1 封装

将属性和行为作为一个整体，表现生活中的事物

将属性和行为加以权限控制

|  |  |
| --- | --- |
| 都可以表示类 | |
| struct | 默认权限为公共 |
| class | 默认权限为私有 |

|  |  |  |
| --- | --- | --- |
| public | 公共权限 | 类内可以访问，类外可以访问 |
| protected | 保护权限 | 类内可以访问，类外不可以访问，继承时儿子可以访问 |
| private | 私有权限 | 类内可以访问，类外不可以访问，继承时儿子不可以访问 |

构造与析构

**构造函数：**

构造函数，没有返回值也不写void

函数名称与类名相同

构造函数可以有参数，因此可以发生重载

程序在调用对象的时候会自动调用构造，无需手动调用，而且只会调一次

|  |  |  |  |
| --- | --- | --- | --- |
| class Person {  public:  /\* code \*/  } | | | |
| 默认构造函数 | Person() | |  |
| 有参构造函数 | Person(int a) | |  |
| 拷贝构造函数 | Person(const Person &p) | | 浅拷贝 |
| 注意：调用时Person p1() 会被解释为p1函数的返回值是Person类型 | | | |
| 括号法调用 | | Person p1(10) | |
| 显示法调用 | | Person p1 = Person(10) | |
| 隐式转换法 | | Person p1 = 10 | |
| 注意：Person(10) 为定义匿名对象，当前行执行结束会被系统回收  Person(p1) 不要利用拷贝构造函数，初始化匿名对象，系统会解释为  Person(p1) === Person p1 即报错p1重定义 | | | |

**析构函数：**

析构函数，没有返回值也不写void

函数名称与类名相同，在名称前加符号~

析构函数不可以有参数，因此不可以发生重载

程序在对象销毁前会自动调用析构，无需手动调用，而且只会调一次

写释放堆区空间的操作

**构造函数-初始化列表：**

Person(int a, int b, int c) : m\_A(a), m\_B(b), m\_C(c) { /\* code \*/ }

初始化了m\_A = a，m\_B = b，m\_C= c 这样的操作

**注意：**

1. 如果用户定义有参构造函数，c++则不再提供默认无参构造，但是会提供默认拷贝构造函数

2. 如果用户定义拷贝构造函数，c++则不会再提供其它构造函数

3. 创建一个类c++编译器会给每个类都添加至少3个函数：默认构造，析构函数，拷贝构造

4. 当其它类对象作为本类成员，构造函数先构造类对象，再构造自身，析构的顺序与其相反

类内静态成员

class Person {

public:

static int m\_A;

static void func() { /\* code \*/ }

}

int Person :: m\_A = 100;

int main{

Person p;

cout << p.m\_A << endl; // 100

Person pt;

pt.m\_A = 200; // 修改了另一个类里的静态变量

cout << p.m\_A << endl; // 200

}

1. 静态成员变量不属于某个对象上，所有对象都共享同一份数据

两种访问方式：

通过对象进行访问 Person p; p.m\_A;

通过类名进行访问 Person :: m\_A;

1. 静态成员函数

两种访问方式：

通过对象进行访问 Person p; p.func();

通过类名进行访问 Person :: func();

1. 编译阶段就分配内存
2. 类内声明，类外初始化
3. 静态成员变量和静态成员函数也有访问权限，类内声明在private
4. 静态成员函数只能访问静态成员变量
5. 只有非静态成员变量才属于类的对象上，静态成员变量不属于类的对象上
6. 成员变量和成员函数是分开存储的

this指针

this指针指向被调用的成员函数所属的对象

当形参和成员变量同名时，可用this指针来区分

在类的非静态成员函数中返回对象本身，可使用 return \*this

class Person{

Person& test() {   
 /\* code \*/

return \*this;

}

}

int main(){

Person p;

p.test().test().test();

}

注意：返回不是引用（将Person& 改为 Person），那么将会返回一个新的Person，而不是本体，那么在执行第二个test()函数时就是对一个的返回new出来新的Person上操作

class Person{

void test() {   
 cout << age << endl;

// cout << this -> age << endl;

}

void show() {

cout << “this is person” << endl;

}

int age;

}

int main(){

Person \*p = NULL;

p -> test(); // error

p -> show(); // this is person

}

注意：空指针调用test函数时会报错，报错信息为this指针为nullptr，而无操作的show函数正常可以打印，可以判断一下this == NULL然后退出函数

class Person{

void test() const {   
 age=100; // error

this -> name = 123; // success

}

int age;

mutable int name;

}

this指针不可以修改指针的指向

（定义：Person \* const this;）

在成员函数后加const则this指针指向的值也不可以修改，修饰的是this指向，叫做常函数（加一把锁）

（定义：const Person \* const this;）

在定义成员变量时加mutable关键字则成为特殊变量，即使在常函数中也可修改

（配一把钥匙）

在对象定义前加const变为常对象，与常函数相似，没有加mutable关键字的成员变量都不允许修改

常对象只能调用常函数，不可以调用普通成员函数，因为普通成员函数可以修改属性

2-2 继承

2-3 多态

静态多态的函数地址早绑定

动态多态的函数地址晚绑定

静态多态：

函数重载和运算符重载都属于静态多态，复用函数名

动态多态：

派生类和虚函数实现运行时多态