**C++课程索引**

黑马程序员匠心之作|C++教程从0到1入门编程,学习编程不再难

https://www.bilibili.com/video/BV1et411b73Z?p=127

如果有C语言基础的建议直接从p72开始（也可以看一下p2和p11）

**前87集中C和C++的区别：**

**①头文件 #include <iostream>**

**②标准命名空间using namespace std**

**③输入 cin>> 输出 cout<<<<endl;**

**④system("pause");按任意键结束**

**⑤string字符串，要带<string>头文件。**

**前面基本就这些，其他和C都一样，大家可以直接从p84开始（复习一下内存分区）**

缺的一P：30 静态成员变量

Shfit+TAB左对齐

TAB自动填写备选单词

﻿

**02 第一个C++程序-C++书写HelloWorld P2 - 05:07**

﻿

#include<iostream>

using namespace std;

cout<<"hello"<<endl;

system("pause");

﻿

**14 数据类型-布尔类型 P14 - 01:22**

﻿

bool布尔数据类型 true-非零 false-0

单精度float赋值建议加f如：

float a = 3.14f;

﻿

**06 通讯录管理系统-添加联系人-... P77 - 11:44**

﻿

通讯录中断(判断为非重点)

﻿

**01 程序的内存模型-内存四区-代码区. P84 - 03:20**

﻿

内存分区模型

程序运行前：代码区，全局区

程序运行后：栈区，堆区。

①代码区：共享，只读

﻿

**02 程序的内存模型-内存四区-全局区 P85 - 01:12**

﻿

②全局区

全局变量global，static静态变量，字符串常量，const修饰的全局变量(全局常量)

常量区

局部变量，const修饰的局部常量(局部常量)

﻿

**03 程序的内存模型-内存四区-栈区 P86 - 03:02**

﻿

栈区：编译器使用

不要返回局部变量地址(虽然编译器会帮你保留一次)

﻿

**04 程序的内存模型-内存四区-堆区 P87 - 02:30**

﻿

堆区程序结束会回收

new

是C++关键字，开辟内存

返回指针

int \* p = new int (10);

注意此处指针在栈区(局部变量)

数据在堆区(程序员管理)

﻿

**05 程序的内存模型-new运算符 P88 - 00:33**

﻿

delete释放内存空间，对应new

﻿

**05 程序的内存模型-new运算符 P88 - 08:55**

﻿

new创建数组

int \* arr = new int [10]

用中括号

delete[] arr需要中括号

﻿

**06 C++中的引用-引用的基本语法 P89 - 03:20**

﻿

C++的别名引用

语法：数据类型+&别名=原名

int a = 10;

int &b = a;

修改b相当于修改a

﻿

**07 C++中的引用-引用的注意事项 P90 - 01:53**

﻿

引用①必须初始化②不可更改

即不能先int &b ;

也不能int &b = a又int&b = c;

﻿

**08 C++中的引用-引用做函数参数 P91 - 05:46**

﻿

①值传递

②地址传递\*，每一次都要\*

③引用传递&

﻿

**09 C++中的引用-引用做函数返回值 P92 - 03:37**

﻿

①不可返回局部变量的引用

﻿

**09 C++中的引用-引用做函数返回值 P92 - 05:44**

﻿

②函数的调用(全局的引用)可以作为左值

﻿

**10 C++中的引用-引用的本质 P93 - 07:00**

﻿

&取别名，本质是指针

﻿

**11 C++中的引用-常量引用 P94 - 02:44**

﻿

int &ref = 10 不合法

const int &ref = 10 合法，且不可修改值

const加入函数形参，避免误改变

﻿

**12 函数高级-函数的默认参数 P95 - 03:09**

﻿

函数定义可以带默认值，使得数据缺省时用默认值

int fun(int a,int b = 20 , int c = 30){}

当fun(10，30)时，a=10,b=30,c=30诸如此类

﻿

**12 函数高级-函数的默认参数 P95 - 06:33**

﻿

①注意，默认值设定之后，从这个位置往后都要设置默认值，之前的不需

﻿

**12 函数高级-函数的默认参数 P95 - 09:08**

﻿

②函数声明 和 函数实现 不能同时定义默认

声明如果定义了默认，函数不可再定义默认

﻿

**13 函数高级-函数的占位参数 P96 - 02:53**

﻿

占位参数，即作为形参，不需要命名，但是需要相应的实参。

占位参数还可以定义默认值，这样连实参都不需要调用

﻿

**14 函数高级-函数重载-基本语法 P97 - 02:00**

﻿

函数重载

①必须在同一个作用域下(如全局域)

②函数名相同

③形参有区别(形参没区别不成立，函数类型不同也不行)

﻿

**14 函数高级-函数重载-基本语法 P97 - 05:36**

﻿

根据传入参数对应不同，编译器选择不同的函数

如func()

func(int a)

func(double b)

func(int a,double b)

func(double b,int a)

都有效

﻿

**15 函数高级-函数重载-注意事项 P98 - 02:37**

﻿

①引用& 可实现重载

int & a 和c onst int &a

int a =10调用前者

直接func(10)传入常量调用后者

﻿

**15 函数高级-函数重载-注意事项 P98 - 08:14**

﻿

②默认值设定也可重载，小心别写出二义性bug

﻿

**16 类和对象-封装-属性和行为作为整体 P99 - 02:52**

﻿

类，对象

C++三大特性：封装，继承，多态

﻿

**16 类和对象-封装-属性和行为作为整体 P99 - 06:54**

﻿

封装class

①访问权限public:

②属性

③行为(函数)

通过 类 创建的个体为 对象 ，有个体名字

创建对象即“实例化”

样例圆、学生

﻿

**17 类和对象-封装-案例-设计学生类 P100 - 08:19**

﻿

类的成员=属性+行为

属性=成员属性=成员变量

行为=成员函数=成员方法

可以先 行为 再 属性

可以各加一个public:

﻿

**18 类和对象-封装-访问权限 P101 - 00:15**

﻿

类的访问权限

①public 公共

②protected 保护

③private 私有

﻿

**18 类和对象-封装-访问权限 P101 - 03:40**

﻿

①public 成员 类内可以访问，类外可访问

②protected 成员 类内可以访问，类外不可访问

③private 成员 类内可以访问，类外不可访问

protected和private区别在于继承，

protected可以被儿子访问，private不可

public可在类外访问并修改，

后二者不可

﻿

**19 类和对象-封装-c++中class和struct的区别 P102 - 00:51**

﻿

struct和class区别

struct默认访问 是公有

class默认访问 是私有

﻿

**20 类和对象-封装-成员属性私有化 P103 - 01:08**

﻿

class私有化优点

①可以自己控制读写权限

②可以检验数据有效性

可设置

①只读②只写③可读可写

﻿

**20 类和对象-封装-成员属性私有化 P103 - 11:32**

﻿

private读写增加限制

﻿

**21 类和对象-封装-设计案例1-立方体类 P104 - 00:20**

﻿

继承案例1-立方体类

注意事项①用&减少空间占用

②全局函数和成员函数区别

全局函数需要调用两个对象，成员函数只需要一个：

可以把外部数据通过 成员函数 调入类内计算

﻿

**22 类和对象-封装-设计案例2-点和圆关系 P105 - 00:01**

﻿

继承案例2-点圆关系

类中可以包含另一个类作为成员

圆类 内包含 点类 的思想

圆心也算一个点类对象

﻿

**22 类和对象-封装-设计案例2-点和圆关系 P105 - 18:53**

﻿

类的头文件、源文件的声明和实现

①头文件

#pragma once

#include <iostream>

using namespace std;

类在头文件内留声明，其函数部分不留操作内容，属性不需要修改

﻿

**22 类和对象-封装-设计案例2-点和圆关系 P105 - 20:07**

﻿

②源文件

#include "point.h"

只保留函数实现，但是需要::作用域引用类属性

比如point::m\_X

Shift+TAB快速缩进

﻿

**23 类和对象-对象特性-构造函数... P106 - 01:31**

﻿

构造函数——初始化(调用时触发)

析构函数——清理(销毁时触发)

如果程序员没定义，编译器强制实行空函数

需要写在public下

﻿

**23 类和对象-对象特性-构造函数... P106 - 05:33**

﻿

相同点：

①不写返回类型，包括void也不写

②自动调用，只调用一次

③类名要一致

不同：

①构造函数可以有参数，可重载，析构不可

②析构前要加~

~类名(){}

﻿

**24 类和对象-对象特性-函数的分... P107 - 03:30**

﻿

构造参数：

①按照参数分类：

有参构造(默认构造)，无参构造

②按照类型分类：

普通构造，拷贝构造

拷贝构造函数 如 Person(const Person &p)

需要const保证本体不改变，拷贝提个一模一样的对象，&确保不占额外空间

﻿

**24 类和对象-对象特性-函数的分... P107 - 08:12**

﻿

调用构造函数方法：

①括号法

②显示法

③隐式转换法

﻿

**24 类和对象-对象特性-函数的分... P107 - 12:44**

﻿

①括号法调用默认函数不需要括号，也即无传参就不需要写括号，有参则需要括号

带()的话，编译器会当做函数声明

﻿

**24 类和对象-对象特性-函数的分... P107 - 16:07**

﻿

②显示法

用=

1.显示法等号=右边是“匿名对象”，一旦完成立即销毁(也即触发析构)

2.不要拷贝构造函数 来 初始化匿名对象

﻿

**24 类和对象-对象特性-函数的分... P107 - 21:53**

﻿

③隐式转换法

Person p4 = 10 相当于 Person p4 = Person (10)

﻿

**25 类和对象-对象特性-拷贝构造... P108 - 07:58**

﻿

拷贝函数调用时机：

①用已有来创建新对象

②函数值传递中，类作为传参实际上是算拷贝

如func(Person p)

③函数返回的时候也会拷贝(因为局部变量p会销毁)

Person func(p)

{return p}

﻿

**26 类和对象-对象特性-构造函数... P109 - 00:40**

﻿

默认构造函数（空），

默认析构函数（空），

默认拷贝函数（值的拷贝）

调用规则：

①写了拷贝，编译器不自动写其他构造函数

②写了有参构造，编译器不提供默认(无参)构造，但仍提供拷贝构造

﻿

**27 类和对象-对象特性-深拷贝与浅拷贝 P110 - 00:14**

﻿

面试经典坑：

浅拷贝，深拷贝

浅拷贝：编译器简单拷贝

深拷贝：程序员写的拷贝函数，避免重复delete内存

﻿

**27 类和对象-对象特性-深拷贝与浅拷贝 P110 - 11:00**

﻿

浅拷贝会导致堆区内存的重复释放，利用深拷贝解决，

以及析构delete释放内存空间

﻿

**27 类和对象-对象特性-深拷贝与浅拷贝 P110 - 13:15**

﻿

浅拷贝解决方法实例

﻿

**28 类和对象-对象特性-初始化列表 P111 - 03:32**

﻿

初始化列表语法：

构造函数():属性1(值1),属性2(值2){}

﻿

**29 类和对象-对象特性-类对象作... P112 - 00:51**

﻿

类中有其他类的对象，称为对象成员

①先构造成员再构造本类

即先构造部分，再整体

②析构顺序则和构造相反(因为栈先进后出)

﻿

**31 类和对象-对象特性-静态成员-... P113 - 00:12**

﻿

静态成员static

静态成员变量 和 静态成员函数

①所有对象共享一个静态成员函数

②静态成员函数只能访问静态成员变量

﻿

**31 类和对象-对象特性-静态成员-... P113 - 02:55**

﻿

静态成员函数 访问方式

①通过对象访问p.func();

②通过类名访问Person::func();

因为static静态成员变量只有一个，所以静态函数访问不会有歧义，但是非静态的属性会在不同对象里，无法确定

﻿

**31 类和对象-对象特性-静态成员-... P113 - 07:49**

﻿

类外访问不到私有静态成员函数

静态成员类内声明，类外赋值

静态成员赋值如：int Person::m\_a = 10;

﻿

**32 类和对象-对象特性-成员变量... P114 - 00:38**

﻿

属性和函数分开存储

只有非静态成员变量才在类的对象上

static的变量、static函数和非static的函数都不在

﻿

**32 类和对象-对象特性-成员变量... P114 - 03:37**

﻿

①空对象占用1个字节，确保不会和其他冲突

②非空对象则不额外占用这1个字节，比如有一个int属性，(size of 对象)就是4个字节

③这里也只储存非静态的变量，其他不存在这里

﻿

**33 类和对象-对象特性-this指针的用途 P115 - 01:15**

﻿

this指针，非静态成员函数自带，用于判断是哪个对象正在调用函数

return \*this可以返回调用者本身

﻿

**33 类和对象-对象特性-this指针的用途 P115 - 05:42**

﻿

①this指针可以解决形参、自身属性的冲突

this->age = age

﻿

**33 类和对象-对象特性-this指针的用途 P115 - 10:12**

﻿

②\*this可以return自己 实现 链式编程

﻿

**34 类和对象-对象特性-空指针访... P116 - 02:52**

﻿

Person \* p =NULL可以定义一个空指针

(不指向确切地址，但是指针类型是Person \*)

①对于不需要调用成员变量的情况不会报错

②对于调用成员函数的情况会报错

加一个if(p==NULL){return;}来避免bug，增强健壮性

﻿

**35 类和对象-对象特性-const修饰成员函数 P117 - 00:01**

﻿

常函数，常对象 const

﻿

**35 类和对象-对象特性-const修饰成员函数 P117 - 05:50**

﻿

在成员函数后面加const，本质是让this指针指向的值不可改，(而this指针的指向本身就是不可改的)

如void show()const:

(this指针相当于const Person \* const this)

①前一个const使指向的对应的值不可改

②后一个const使指向的位置不可改

﻿

**35 类和对象-对象特性-const修饰成员函数 P117 - 07:12**

﻿

mutable int b

在前面加上mutable使其成为特例，就可以在const常函数中对mutable的变量进行修改

(没有mutable修饰的变量无法在const修饰的函数中被修改)

﻿

**35 类和对象-对象特性-const修饰成员函数 P117 - 08:25**

﻿

常对象，对象不可修改

如const Person p;

const加在前面

不过有mutable属性的成员就例外，可以修改

﻿

**35 类和对象-对象特性-const修饰成员函数 P117 - 11:10**

﻿

常对象只能调用常函数，避免常对象成员被修改的可能

﻿

**36 类和对象-友元-全局函数做友元 P118 - 01:04**

﻿

友元 friend

①友元friend修饰全局函数

将全局函数声明复制一份，放在class最前面，再加个friend前缀，加个分号;

而且不需要写public等访问权限

之后该全局函数就可以在类外访问私有成员

﻿

**37 类和对象-友元-友元类 P119 - 03:45**

﻿

Person::Person(){}这样可以把构造函数等放在类外

可以把类的声明先写在前面，让后面不报错，如class A；这样

﻿

**37 类和对象-友元-友元类 P119 - 08:25**

﻿

②友元friend修饰类

把A类声明copy一份放在B类最前面，再加个friend前缀，加个分号;操作同①一样

A类就可以访问B类的private成员

如class B

{friend class A;

};

﻿

**38 类和对象-友元-成员函数做友元 P120 - 07:46**

﻿

③友元friend修饰类内"成员函数"

修饰方法基本同上

﻿

**39 类和对象-C++运算符重载-加号... P121 - 03:29**

﻿

运算符重载，使得运算符能计算新的类型

1.比如对象+对象等

形如:

①成员函数方式的重载：

Person operator+(Person &p)

{

Person temp;

temp.m\_a = this->m\_a + p.m\_a;

return temp;

}

可以直接p3 = p1+p2使用

②全局函数形式重载：

Person operator+(Person &p1,Person &p2)

成员函数和全局函数冲突，运算符重载不可同时存在

﻿

**40 类和对象-C++运算符重载-左移... P122 - 01:05**

﻿

2.<<重载，让cout<<可以输出自定义数据类型

只能全局函数重载，成员函数只能p<<cout，反了

﻿

**40 类和对象-C++运算符重载-左移... P122 - 12:06**

﻿

可以用友元friend全局函数，来让private成员也有效

ostream& operator<<(ostream &cout,Person &p)

{

cout << "a=" << p.m\_a << " b=" << p.m\_b;

return cout;

}//全局函数重载运算符<<。cout是ostream-标准输出流格式，可以右键转到定义查看。cout和Person都要&

//返回值是cout的类型ostream&，最后return cout，这样才能再在后面接内容，链式思想。否则连endl都会报错

//另-引用可以起别名，这里cout改成任意名字在局部函数内都可行

cout << p1 << endl;//算数符重载函数如果没返回cout，无法接endl等后续内容

﻿

**41 类和对象-C++运算符重载-递增... P123 - 00:04**

﻿

3.递增运算符重载 ++ --

﻿

**41 类和对象-C++运算符重载-递增... P123 - 12:29**

﻿

高仿的++a需要返回引用值myint & a

否则不是在操作同一个数据

﻿

**41 类和对象-C++运算符重载-递增... P123 - 15:30**

﻿

函数重载满足条件：参数个数、顺序、类型其一不同

而作用域、函数名称相同

返回值不同不算做函数重载，会冲突

﻿

**41 类和对象-C++运算符重载-递增... P123 - 16:05**

﻿

myint& operator++()就是前置递增：

①返回是引用，无参数

myint operator++(int)就是后置递增：

①返回是值，有int作为占位，编译器就会之别此为后置递增

﻿

**42 类和对象-C++运算符重载-赋值... P124 - 00:02**

﻿

4.赋值运算符=重载

类除了自动创建构造、析构、拷贝函数外，还会创建赋值函数这第四个函数，但同拷贝一样会有浅拷贝问题，用自定义赋值的深拷贝解决

﻿

**42 类和对象-C++运算符重载-赋值... P124 - 05:28**

﻿

int\* a的成员属性，用new int()创建新区域来赋值，结束时用~析构函数delete删除占用空间，直接这样写会报错，因为赋值的是地址，不能两次删除同一个地址

﻿

**42 类和对象-C++运算符重载-赋值... P124 - 11:34**

﻿

解决方法是自定义赋值让赋值不是原来的地址，而是开辟新地区放值，再指向这个新地址

﻿

**42 类和对象-C++运算符重载-赋值... P124 - 13:27**

﻿

需要补充的一个不足

有c=b=a，3者最后都为a的值

即 赋值函数需要return \*this

﻿

**43 类和对象-C++运算符重载-关系... P125 - 00:12**

﻿

5.关系运算符重载 > < == !=

﻿

**43 类和对象-C++运算符重载-关系... P125 - 05:19**

﻿

bool operator==(Person &p)

bool operator!=(Person &p)

return true 和false

﻿

**44 类和对象-C++运算符重载-函数... P126 - 00:11**

﻿

6.函数调用运算符重载

即()的重载，也叫仿函数，因为它像函数调用，但本质是对象

void operator()(string name)

就是让myprint("hello")这样的语法有效

仿函数灵活无特定写法

在STL中大量使用

﻿

**44 类和对象-C++运算符重载-函数... P126 - 09:19**

﻿

匿名对象调用，可以不创建对象，直接用类来操作

﻿

**45 类和对象-继承-基本语法 P127 - 01:30**

﻿

继承-面向对象三大特性之二

﻿

**45 类和对象-继承-基本语法 P127 - 12:10**

﻿

继承语法：

class 子类 : 继承方式 父类

class Java : public BasePage

class Python : public BasePage

子类 也称 派生类

父类 也称 基类

从父类拿来的是共性，子类自己另有的是个性

﻿

**46 类和对象-继承-继承方式 P128 - 01:00**

﻿

继承方式

父类private子类不可访问，无论何种继承方式

继承方式意味着继承下来那部分的最低权限，

①继承方式为public。父类的public和protected不变

②继承方式为protected。public变成protected,

③继承方式为private。public和protected都变private

﻿

**47 类和对象-继承-继承中的对象模型 P129 - 03:50**

﻿

继承，是继承了所有非静态成员

①继承时虽然父类的private无法访问，但仍继承了，在sizeof中有所体现（父类private被编译器隐藏）

②继承后的sizeof大小是父类+子类的大小之和，包括private的大小

（size只有变量属性，函数和常量属性sizeof不算到，因为它们独一份）

﻿

**47 类和对象-继承-继承中的对象模型 P129 - 05:50**

﻿

VS开发人员命令提示符 工具使用，代码敲打

查看对象模型(类的结构)

D:跳转盘符

cd 路径 跳至路径

dir 查看当前目录内容

cl /d1 reportSingleClassLayout类名 "文件名.cpp"

文件名.cpp可以TAB快速填充

（cl是字母L，d1是数字1）

﻿

**48 类和对象-继承-构造和析构顺序 P130 - 03:14**

﻿

继承的构造、析构

现有父类的构造→子类的构造→子类的析构→父类的析构

（言外之意是子类构造时也创建了一个父类，虽然这父类没有名字，拿不到）

﻿

**49 类和对象-继承-同名成员处理 P131 - 00:30**

﻿

继承后，如果父类子类有重名成员，

子类直接用，父类需要作用域::前缀，属性和函数都适用

比如：

s.Base::m\_a;（子类单个对象s中继承的父类Base的m\_a值）

s.m\_a;（子类自己的m\_a值）

﻿

**49 类和对象-继承-同名成员处理 P131 - 08:06**

﻿

子类和父类的成员函数同名，子类会隐藏所有父类的同名函数，包括所有的重载形式，都不可用。

不过作用域::前缀依然可以解决这个问题

﻿

**50 类和对象-继承-同名静态成员处理 P132 - 01:16**

﻿

继承中 静态成员重名也是用作用域::区分父类成员。

静态变量在类内声明，类外初始化（毕竟独一份），父类子类分别初始化（用不同的作用域::）

﻿

**50 类和对象-继承-同名静态成员处理 P132 - 05:44**

﻿

静态成员可以不需要创建单个对象，而直接访问类来得到

有Son::Base::m\_a来访问，有效。

这里①Son::指通过类名方式访问

②Base::指父类作用域下

﻿

**50 类和对象-继承-同名静态成员处理 P132 - 09:51**

﻿

静态成员访问方式：

①通过单个对象访问

②通过类名访问，而不需创建对象

除此之外作用域::的使用方式和非静态成员一样

﻿

**51 类和对象-继承-继承语法 P133 - 00:12**

﻿

多继承 一个儿子多个爹，逗号分隔继承方式和父类即可

class A : 继承方式 B , 继承方式 C ……

多继承实际开发不建议，会引发成员重名，但仍可通过加相应作用域::解决（问题叫二义性）

﻿

**52 类和对象-继承-菱形继承问题... P134 - 00:02**

﻿

菱形继承

A→B，A→C

BC→D

﻿

**52 类和对象-继承-菱形继承问题... P134 - 09:49**

﻿

虚继承 解决菱形继承的数据重复问题（其实数据可能了）

class A : virtual 继承方式 B

加了virtual是虚继承，父类叫虚基类

只要多继承B和C的这一部分virtual就行，开始A和结束D不需要v加irtual

﻿

**52 类和对象-继承-菱形继承问题... P134 - 13:51**

﻿

virtual虚继承底层：新创建一个属性，原来的两个属性改为指向这个新属性

虚基类指针-vbptr

指向虚基类表-vbtable

ptr地址+偏移量=新属性的地址

（关于sizeof，因为数据变成了指针，而指针在x84是4字节，x64是8字节，所以根据情况size改变。比如两个char虚继承，就是1+1增加成了4+4+1，size大增。3个long long继承，就是8+8+8变成了4+4+4+8，size小了）

﻿

**53 类和对象-多态-多态的基本语法 P135 - 00:02**

﻿

多态-第三大面向对象特性（封装、继承、多态）

多态分类：

①静态多态：函数重载 和 运算符重载，复用函数名

②动态多态：派生类和虚函数实现运行时多态

区别：

①静态多态地址早绑定，编译阶段确定函数地址

②动态多态地址晚绑定，运行阶段确定函数地址

﻿

**53 类和对象-多态-多态的基本语法 P135 - 04:57**

﻿

父类引用可接收子类

即全局函数参数Animal &animal 可以传参cat进去，而cat是Animal子类Cat的对象cat

﻿

**53 类和对象-多态-多态的基本语法 P135 - 06:36**

﻿

子类、父类同名函数，全局函数因为地址先绑定（静态多态），所以输出父类结果

要想输出子类结果，需要父类的对应函数前加virtual实现动态多态晚绑定，叫虚函数

在父类已经加上virtual时，子类加不加效果一样

﻿

**53 类和对象-多态-多态的基本语法 P135 - 10:51**

﻿

动态多态，即父类函数引入子类，用虚函数实现：一个函数，传参子类不同，走的不同子类的同名成员，详见示例理解

﻿

**53 类和对象-多态-多态的基本语法 P135 - 12:23**

﻿

动态多态满足条件：

①有继承关系

②子类重写父类的虚函数

重写和重载不同：

重写是格式完全一样，作用域不同（继承）

重载是参数不同，在同一作用域

﻿

**53 类和对象-多态-多态的基本语法 P135 - 14:17**

﻿

动态多态使用场景：

父类引用&指向→子类对象

﻿

**54 类和对象-多态-多态的原理剖析 P136 - 02:44**

﻿

vfptr-虚函数（表）指针，表指Table虚函数表

virtual 虚函数增加一个指针（4或8字节），因为函数本身是独一的，它不储存在类的size里，所以额外增加size

（但是类本身就没有自己的属性，只占1字节时，size就直接替换4上去，而不是5）

vfptr指向地址&Animal::speak

﻿

**54 类和对象-多态-多态的原理剖析 P136 - 12:45**

﻿

动态多态原理：

就是指针，指针指向的表的不同

﻿

**55 类和对象-多态-案例1-计算器类 P137 - 16:15**

﻿

多态好处：

①结构清晰②可读性强③利于维护和拓展

开闭原则：对扩展开放，对源码关闭

﻿

**56 类和对象-多态-纯虚函数和抽象类 P138 - 01:15**

﻿

当父类的虚函数内容确实用不上时，可写成纯虚函数。

纯虚函数不需要写实现内容，直接让虚函数=0就行，不需要带大括号。同时这个类就叫做抽象类。

子类需要重写父类的纯虚函数，如果子类不重写纯虚函数，那子类也是抽象类。

﻿

**56 类和对象-多态-纯虚函数和抽象类 P138 - 05:11**

﻿

抽象类无法实例化对象，也就是不能创建对象

﻿

**58 类和对象-多态-虚析构和纯虚析构 P140 - 07:51**

﻿

内存泄漏-子类析构函数未调用问题

父类指针指向子类，即多态场景下，delete父类指针不会触发子类的析构，使得内存泄露

﻿

**58 类和对象-多态-虚析构和纯虚析构 P140 - 09:19**

﻿

内存泄漏解决方法：父类析构函数前加virtual成虚析构，就会先走子类析构，再走父类析构，否则会不走子类析构

﻿

**58 类和对象-多态-虚析构和纯虚析构 P140 - 12:17**

﻿

纯虚析构

类内声明virtual ~Animal()=0;

类外定义Animal::~Animal(){析构函数调用}

纯虚析构也会让类成抽象类，无法创建单个对象

﻿

**58 类和对象-多态-虚析构和纯虚析构 P140 - 15:51**

﻿

虚析构 和 纯虚析构 区别：

①虚析构类内声明+定义。纯虚析构类内声明、类外定义

②纯虚析构 的类是抽象类，不可实例化对象

﻿

**60 类和对象-多态-案例3-电脑组... P142 - 00:08**

﻿

多态案例-电脑组装

﻿

**61 C++文件操作-文本文件-写文件 P143 - 00:56**

﻿

C++文件操作

头文件<fstream>

文本类型：①文本文件，ASCII码值

②二进制文件，二进制

﻿

**61 C++文件操作-文本文件-写文件 P143 - 02:47**

﻿

操作文件三大类：

①ofstream 输出out

②ifstream 输入in

③fstream 可读可写

﻿

**61 C++文件操作-文本文件-写文件 P143 - 06:09**

﻿

写文件

①头文件#include<fstream>

②创建流对象 ofstream ofs;

③打开文件 ofs.open("文件路径",打开方式);

④写数据，ofs<<"写入的数据";

⑤关闭文件ofs.close();

﻿

**61 C++文件操作-文本文件-写文件 P143 - 06:39**

﻿

打开文件方式

ios::in读

ios::out写

ios::ate初始位置在文件尾

ios::app追加方式写文件

ios::trunc先删除文件，再创建新文件

ios::binary二进制方式

文件方式可以两种方法一起用。用|操作符

ios::binary | ios::out 二进制写

﻿

**61 C++文件操作-文本文件-写文件 P143 - 13:57**

﻿

不指定路径，则文件默认放在当前项目路径下

#include<fstream>//读写文件的头文件

void test01()

{

ofstream ofs;//直接用，不需要创建类

ofs.open("text.txt", ios::out);//ofs.open打开文件，("路径",打开方式)，没有路径则放在当前目录下，ios::out是输出到文件

ofs << "姓名：张三" << endl;//相当于cout<<，但是是输出到文件

ofs << "性别：男" << endl;

ofs << "年龄：18" << endl;

ofs.close();//关闭文件

}

﻿

**62 C++文件操作-文本文件-读文件 P144 - 06:27**

﻿

读文件

需要判断文件是否打开成功

四种方式读取文件

fstream可以代替ofstream和ifstream，因为可读可写

﻿

**63 C++文件操作-二进制文件-写文件 P145 - 02:07**

﻿

写文件时，最好不用C++的string，可能会出问题

用C语言的char s[]

﻿

**63 C++文件操作-二进制文件-写文件 P145 - 04:01**

﻿

ofstream ofs("Person.txt", ios::out | ios::binary);

可以在创建对象时直接在()参数内写上打开文件的内容，而不需要再调用ofs.open，因为内置了相应构造函数

ofs.write((const char\*)&p, sizeof(p));

有乱码无妨，读回来正确就行

Person p = { "张三",18 };//默认没有相应构造函数，但是有这个相应赋值函数

﻿

**64 C++文件操作-二进制文件-读文件 P146 - 00:14**

﻿

ofs.write 写入

ifs.read 读取

﻿

**01 职工管理系统-需求分析以及案... P147 - 00:36**

﻿

职工管理系统(未学习)

P72的通讯录系统也未学习

﻿

**01 模板-模板的概念 P167 - 00:24**

﻿

泛型编程（模板）、STL部分开始

面向对象和泛型编程，C++两种编程思想

﻿

**02 模板-函数模板基本语法 P168 - 00:30**

﻿

函数模板和类模板

template<typename T>

typename表示后面的T数据类型，可换成class

T是数据类型，一般是大写字母，可任意名称

﻿

**02 模板-函数模板基本语法 P168 - 12:44**

﻿

函数模板使用方式：

①自动类型推导，编译器自己判断数据类型

②显示指定类型，程序员给定数据类型

﻿

**03 模板-函数模板注意事项 P169 - 07:16**

﻿

注意事项：

①编译器推导的数据类型需要一致

②编译器推不出数据类型时，需要<>指定（也就是没有传参的函数情况）

﻿

**04 模板-函数模板案例-数组排序 P170 - 00:04**

﻿

函数模板案例-数组排序

﻿

**05 模板-普通函数与函数模板区别 P171 - 01:23**

﻿

普通函数、函数模板区别

①普通函数 调用可发生隐式类型转换

②函数模板 用自动类型推导，不可发生隐式类型转换

③函数模板 用显示指定类型<>，可以发生隐式类型转换

﻿

**05 模板-普通函数与函数模板区别 P171 - 03:53**

﻿

隐式类型转换：

比如用int+char，会把char转化成ASCII码值作为int相加

﻿

**06 模板-普通函数与函数模板调用规则 P172 - 00:23**

﻿

普通函数和函数模板重名：

①优先调用普通函数

②空模板参数列表可以优先调用函数模板

③函数模板可以重载

④如果函数模板可以产生更好匹配，则优先

﻿

**06 模板-普通函数与函数模板调用规则 P172 - 06:39**

﻿

空模板：

调用模板函数时<>内不带数据类型，就可以优先调用函数模板

比如MyPrint<>(a,b);

﻿

**06 模板-普通函数与函数模板调用规则 P172 - 09:44**

﻿

如果函数模板可以产生更好匹配，则优先。

比如普通函数是int，模板是char，当传参是char时，虽然int可以隐式类型转换成ASCII计算，但char更匹配，优先模板

﻿

**07 模板-模板的局限性 P173 - 08:45**

﻿

当模板的T是自定义数据类型比如一个对象时，会无法执行，解决方法：

①模板特例。复制一份模板放在原来的后面，前面加template<>，不需要<>内写类型，传参改成具体的数据类型而不是T，内容也改成对应的内容。

②class的运算符重载（不推荐）

﻿

**08 模板-类模板基本语法 P174 - 06:05**

﻿

类模板

﻿

**09 模板-类模板与函数模板区别 P175 - 00:35**

﻿

类模板和函数模板区别：

①类模板 没有自动类型推导

②类模板 可以默认参数

﻿

**09 模板-类模板与函数模板区别 P175 - 06:01**

﻿

类模板 可以默认参数。在模板参数名后面加上 =数据类型

比如class AgeType = int，当调用模板时<>内没有指定类型，则默认AgeType是int类型

﻿

**10 模板-类模板中成员函数创建时机 P176 - 03:55**

﻿

①类模板还没有调用时，不会创建成员函数，因为数据类型并不确定

②普通类 的成员函数 则会一开始就创建好，放入内存

这也就意味着编译器不会提前给出语法错误，运行时才懂有没有错

﻿

**11 模板-类模板对象做函数参数 P177 - 04:14**

﻿

类模板 作为传参的方法

①指定传入类型（最常用），也就是把整个模板的数据类型放上去，比如Person<string,name>&p作为传参

﻿

**11 模板-类模板对象做函数参数 P177 - 07:24**

﻿

②把传参的参数模板化

比如Person<T1,T2>&p作为传参

用typeid().name，可以查看其数据结构，比如typeid(T1).name和typeid(T2).name这样

﻿

**11 模板-类模板对象做函数参数 P177 - 09:38**

﻿

③整个类模板化

直接把传参的整个类就作为模板，让编译器自己判断传入的数据类型

﻿

**12 模板-类模板与继承 P178 - 00:01**

﻿

类模板与继承

﻿

**12 模板-类模板与继承 P178 - 03:14**

﻿

①要想继承一个 类模板，可以使子类第一行后面加上<>，里面写上继承时选择的数据类型。否则会因为无法确定数据类型，继而无法申请确切大小的内存。

﻿

**12 模板-类模板与继承 P178 - 05:47**

﻿

②继承 类模板，子类也可以变成 类模板。

在子类创建对象的时候用<>给定T的数据类型就行。

比如父类类模板有一个T，子类类模板有一个T1和T2。T1是子类自己的，T2是继承自父类的T，那么需要类似Person<int,char>p1;这样的对象创建来给定所有T的数据类型

﻿

**13 模板-类模板成员函数类外实现 P179 - 03:22**

﻿

类模板的类外实现

①需要加上模板声明

②需要在作用域加上<T>数据类型

不管这个成员函数有没有用到模板参数T，只要它的类是模板类，作用域就需要<>，也即类模板的参数列表

﻿

**14 模板-类模板分文件编写 P180 - 00:01**

﻿

类模板 分文件编写(头文件、源文件问题)

类模板在调用时创建，所以文件编写时链接不到

也即当类模板的声明和实现分离到头文件和源文件时，会有问题

﻿

**14 模板-类模板分文件编写 P180 - 07:42**

﻿

解决 类模板 分文件编写方法：

①把相应头文件的引用改为.cpp源文件

原理就是让编译器直看cpp源码，不会跳过

②相应类模板的.h和.cpp写成一个.hpp，并引用.hpp头文件（推荐）

﻿

**15 模板-类模板与友元 P181 - 00:01**

﻿

类模板 与 友元

①全局函数类内实现（推荐），在类内全局函数前加friend允许其访问private成员

②全局函数类外实现（复杂），"全局函数"不需要作用域

但是需要把实现写在类之前，而且在更之前要声明类，以便让编译器看见。以及加上空<>表示它是函数模板

﻿

**16 模板-类模板案例-数组类封装... P182 - 00:01**

﻿

类模板案例-数组模板

﻿

**17 模板-类模板案例-数组类封装（上） P183 - 05:30**

﻿

数组是delete[]，需要加空的[]

p = new T[m\_capacity];数组内存开辟