绪论 C++ As A "Better C"

1.发明者

C：Dennis Ritchie

**C++：Bjarne Stroustrup**

2.OOP的特征

·以类和对象为基础

·具有三个基本特性

1）封装性：信息隐藏，对象本身包含数据与算法

2）继承性：代码重用，对象可以继承其他对象的特性

3）多态性：同名多义

章一 类

1.类的定义

类是一种变量为对象的数据类型，包含属性（数据成员）和行为（成员函数）

对类的定义包括1）数据成员及其类型的描述2）成员函数的描述

格式：1）{} 2）;

2.类的成员

1）数据成员：一般说来，数据成员是需要隐藏的对象；即外部的程序是不能直接访问这些数据的，应该通过成员函数来访问这些数据。所以一般情况下，数据成员通过关键字private声明为私有成员

客户程序只能通过该模块提供的公开操作来访问这些数据，而不能直接访问这些数据。这种保护措施称为信息隐藏（Information Hiding）

把这些数据与相关操作组织在一起的方式称为封装（Encapsulation）。

2）成员函数：在类定义时声明，除了要规定作用域外，定义方式和一般函数无异。

3.类与对象

类是抽象的数据类型，对象是类的实体。

声明类时，数据成员在内存中还不存在的，只有声明了对象时，才会为类变量分配存储空间。

->点运算符与作用域解析符

（::）用于标识（区分）类的成员，（.）用于标识对象的成员。

3.成员访问控制

**Public** – 程序任意地方皆可访问，struct默认为公有

**Private** – 仅本类成员函数可访问，class默认为私有

**Protected** – 本类及非私有继承的子类可访问

4.接口与实现分离

优点：1）更容易对代码进行修改2）编程可以拆分为定义与实现两部分

头文件 – 包含类的定义与函数原型

源文件 – 包含成员函数的实现

对于类的使用者（即用户代码，简称用户）而言，只需要获得头文件，即可调用类对象的公有函数访问其内部的数据成员。使用者无法直接访问私有成员，也无需知晓公有函数的内部实现。

5.构造函数

1）用于初始化数据成员的特殊成员函数，当对象创建时自动被调用

2）无返回类型，不反回任何值

3）函数名与类名相同

4）一般为公有成员

5）构造函数可以被重载

重载：C++允许同名函数多次定义，要求有不同数目或类型的参数，一般用于相同作用的函数

如何向构造函数传递参数：在声明类对象时，在对象名后直接写上实参列表，编译器就会根据实参的个数和类型选择调用合适的构造函数。

若声明类对象时没有实参列表，则：

(1) 若该类有缺省构造函数（即无参构造函数） ，则调用该构造函数，同时创建该对象。

(2) 若该类没有缺省构造函数，则创建该对象。但该对象的私有成员没有得到初始化。

如果设计的类没有构造函数，C++编译器会自动为该类型建立一个缺省构造函数。该构造函数没有任何形参，且函数体为空。

6.析构函数

1）类的成员函数，当对象被销毁时调用

2）不接收任何参数，无返回类型，不反回任何值

3）函数名为～类名

4）一般为公有成员

5）一个类只能有一个析构函数，析构函数不能重载

7.调用次序

由下而上构建，一般先构建后析构

1) 在全局范围中定义的对象(即在所有函数之外定义的对象)，它的构造函数在文件中的所有函数(包括main函数)执行之前调用。但如果一个程序中有多个文件，而不同的文件中都定义了全局对象，则这些对象的构造函数的执行顺序是不确定的。当main函数执行完毕或调用exit函数时(此时程序终止)，调用析构函数。

2) 如果定义的是局部自动对象(例如在函数中定义对象)，则在建立对象时调用其构造函数。如果函数被多次调用，则在每次建立对象时都要调用构造函数。在函数调用结束、对象释放时先调用析构函数。

3) 如果在函数中定义静态(static )局部对象，则只在程序第一次调用此函数建立对象时调用构造函数一次，在调用结束时对象并不释放，因此也不调用析构函数，只在main函数结束或调用exit函数结束程序时，才调用析构函数。

8.取值函数与赋值函数

指定为public的成员函数，是类的对外接口

指定为private的成员函数，是为本类中的成员函数所调用的，即为 utility function (helper function)

9.对象指针

对象是一种复合型的数据，往往占据比较多内存空间；如果程序中需要使用很多对象，可能容易造成内存紧张。

解决方法：在程序需要对象时创建对象，在对象使用完毕后撤销这个对象。

实现这一方法就要使用指向对象的指针。

this成员函数中指向对象自己

章二 友元、运算符重载

1.有元机制

友元机制允许一个类授权直接访问类的private和protected成员

友元包括1）友元函数 2）友元类 3）友元成员函数

友元关系破坏了类的封装性，不可滥用

友元（friend）关系允许类的设计者选择出一组其他的类或函数，使得它们可以访问该类的私有和受保护成员。

在类的声明中，用friend声明的函数或类，即是该类的友元。

2.友元函数

友元函数不是类成员，但可以访问类的私有成员。

3.传参

传引用比传值快，如果不想改变变量，在前面加上const。

声明时标记const的函数不会改变任何参数值

指针使用CONST

(1)指针本身是常量不可变char\* const pContent;

(2)指针所指向的内容是常量不可变const char \*pContent;

(3)两者都不可变const char\* const pContent;

类使用const

(1)const修饰成员变量

const修饰类的成员函数，表示成员常量，不能被修改，同时它只能在初始化列表中赋值。

(2)const修饰成员函数

const修饰类的成员函数，则该成员函数不能修改类中任何非const成员函数。一般写在函数的最后来修饰。

a. const成员函数不被允许修改它所在对象的任何一个数据成员。

b. const成员函数能够访问对象的const成员，而其他成员函数不可以。

(3)const修饰类对象/对象指针/对象引用

const修饰类对象表示该对象为常量对象，其中的任何成员都不能被修改。对于对象指针和对象引用也是一样。

const修饰的对象，该对象的任何非const成员函数都不能被调用，因为任何非const成员函数会有修改成员变量的企图。

4.运算符重载

对运算符语义的重新定义

1）用自然的方式使用用户自定义的类类型。

1】与基本类型的使用风格一致 2】提高程序的可读性

2）运算符重载的两种形式

1】友元运算符重载 2】类成员运算符重载

3）重载要求

1】至少有一个参数是类类型，参数数目和顺序不能改变

2】以友元或成员函数的形式进行重载

4】不能创建新的运算符

5】 . :: \* , ? sizeof不能被重载

4）两种重载对比

1】赋值算子必须用成员函数重载（如(), [], ->,=）

2】算子左侧参数为不同类型必须用友元函数重载（ << , >> ）

3】一元算子建议用成员函数重载，二元算子建议用友元函数重载

5.对象数组

要定义对象数组的类必须有不需要参数的构造函数

* 有无参构造函数；
* 有全部参数都是默认值的构造函数
* 没有定义任何构造函数；

6.拷贝构造函数

包含指针或动态分配内存的类要使用拷贝构造函数

形参类型为该类类型本身且参数传递方式为按引用传递，一般用const传参。

用一个已存在的该类对象初始化新创建的对象。

每个类都必须有拷贝构造函数：

用户可根据自己的需要显式定义拷贝构造函数。

若用户未提供，则使用系统提供的缺省拷贝构造函数，以逐位复制方式利用已存在的对象来初始化新创建的对象（相当于赋值=）。

拷贝构造函数调用：

* 建立一个新对象，并用另一个同类对象进行初始化
* 函数的参数为类的对象；
* 函数的返回值为类的对象

7.三大函数（Big-3）定义其一，必定义其他

1）拷贝构造函数

2）赋值操作符

3）析构函数

章三 组合与继承

1.组合

1）用于表示has-a的关系

2）一个类的成员包含对象或其他类

3）是程序重用的最普遍形式

为了初始化对象成员，类的构造函数必须调用对象成员所属类的构造函数

2.继承

* **Base class（基类）**／**Parent class（父类）／Super class（超类）**
* **Derived class（派生类）／Sub class（子类）**

子类自动获得父类的数据成员和成员函数，并可以有自己的独特属性与方法

1）用于表示is-a的关系

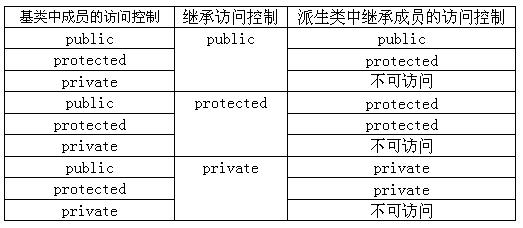
2）提高程序的可重用性

派生类重用基类类的代码可提高程序开发效率。派生类的定义通常基于设计完善、并经严格测试的基类，从而使程序设计工作建立在一个可靠的基础上，有助于高效地开发出可靠性较高的软件

这种重用是一种灵活的重用方式：子类在继承父类代码的基础上，可根据自己的特性进行调整。

->继承方式：（从一个／多个基类继承）单继承、多继承

->影响继承成员（派生类从基类中继承而来的成员）访问控制方式的两个因素：

* + 定义派生类时指定的继承访问控制
  + 该成员在基类中所具有的成员访问控制

3.组合与继承的比较

* "is a" relationship：Inheritance
* "has a" relationship：Composition

4.继承成员重定义与重载

1）重定义

1】派生类中修改继承成员函数的语义（修改函数体，而保持函数原型不变）

2】派生类中的名字屏蔽基类中的名字，基类函数仍能使用（加作用域::）

2）重载继承成员

* 函数名相同，但函数参数列表不同；当然，函数实现一般也不同）。
* 利用重载，实现新的功能。

5.继承的构造函数

* 继承时的构造函数基类的构造函数不被继承
* 派生类的构造函数中只需要对本类中新增成员进行初始化即可。对继承来的基类成员的初始化是通过调用基类构造函数完成的。
* 构造函数的调用次序（创建派生类对象时）
  + 首先调用其基类的构造函数（先虚基类，后直接基类；按照基类被继承时的声明顺序（类图上从左向右））。
  + 然后调用本类对象成员的构造函数（调用顺序按照对象成员在类中的声明顺序）。
  + 最后调用本类的构造函数。
* 撤销派生类对象时析构函数的调用次序与构造函数的调用次序相反
  + 首先调用本类的析构函数
  + 然后调用本类对象成员的析构函数
  + 最后调用其基类的析构函数

6.向基类构造函数传递实参

* 若基类构造函数带参数，则定义派生类构造函数时通过初始化列表显式调用基类构造函数，并向基类构造函数传递实参。

格式：派生类(实参):基类(实参){}

7.类型兼容性

1）对象赋值

1】子类赋值给基类，继承部分赋值，派生部分不赋值

2】基类赋值给子类，仅当赋值操作符重载时可用

2）指针

1】Upcasting：basePtr = static\_cast< BaseClass \* > derivedPtr;

2】Downcasting：derivedPtr = static\_cast< DerivedClass \* > basePtr;

只允许基类指针指向子类，但只能访问基类成员

章四 虚函数与多态

1.多态：一个接口，多种实现

向不同的对象发送同一消息，不同的对象在接收时产生不同的行为。

两类多态：

* 1）编译时多态：在编译时实现多态，包括

（1）function overloading (redefining)

（2）operator overloading

* 2）运行时多态性：在运行时以动态绑定方式实现（继承+虚函数）

绑定（binding）

* 连接函数调用与定义的过程称绑定

两类绑定

* + 静态绑定：编译期、非虚函数
  + 动态绑定：运行前、虚函数

2.虚函数.

* 声明时带关键字virtual的成员函数（定义时不需要virtual）
* “一旦为虚，永远为虚”
* 主要作用：与继承相结合以实现运行时多态性。在公有继承层次中的一个或多个派生类中对虚函数进行重定义，然后通过**指向基类的指针**（或引用）调用虚函数来实现运行时多态性。
* 派生类通过从基类继承的成员函数调用虚函数时，将访问到派生类的版本。
* 多态类: 含有虚函数的类.

3.重载（overloading）、重写（overriding）、重定义（redefining）

虚函数的重定义称为重写

4.虚析构函数

析构函数定义为虚函数使子类析构时调用对应的析构函数，否则只调用基类。

5.纯虚函数：virtual 类型 函数名（形参） const = 0

基类中仅有声明而无定义的虚函数，每个子类只有重写该函数才能实例化。

含有虚函数的基类为抽象类。

6.抽象类和现实类

抽象类只能作为基类，不能实例化。

章五 虚基类与多重继承

1.多重继承：从多个基类中继承

2.多重继承的命名冲突：

不同基类中有同名的数据成员时会产生命名冲突

解决方法：1）使用作用域解析符指明2）子类中重定义该成员

3.重复继承：多路径继承同一基类

两种继承：

* 复制继承：同一基类有不同成员的拷贝（默认，会出现歧义）
* 共享继承：同一基类仅一份成员拷贝

4.虚基类

继承声明时，在继承访问控制前添加virtual。

与普通基类的唯一区别只有在派生类重复继承了某一基类时才表现出来

章六 模版

1.泛型编程

独立于任何特定数据类型的编程，不同类型的数据（对象）可以由同一代码操作。当从泛型代码创建实例代码（实例化）时，才确定特定的数据类型。

是编译时多态，它对数据类型本身进行参数化，从而使程序具有多态性特征

2.模版：模版用于泛型编程，包括函数模版和类模版。

1）函数模版：不知道数据类型或代码重复时使用

template <typename T1, typename T1,…模版形参表>

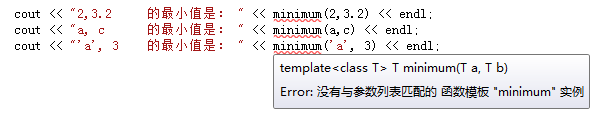
Return-T functionName(形参表){

//code block

}

1】模版形参表不能为空，且所有参数必须在模版中使用。可以有多重类型的参数，但每个类型参数必须用关键字 typename 或class 限定，也可以是固定类型的参数。

2】template与模版函数定义之间有任何语句

 3】调用模版函数时，1（模板实参推断）编译器根据函数调用中所给出的实参的类型，确定相应的模板实参。2（实例化）使用模板实参代替相应的模板形参，产生并编译函数模板的一个实例。实例化过程中不进行参数转换。

2）类模版：定义可以存放任意类型对象的通用容器类

template <typename T1, typename T1,…模版形参表>

class className(形参表){

//code block

}；

/\* ------------------------------------------ \*/

template <typename T1, typename T1,…模版形参表>

Return-T className<模版形参表>::functionName(形参表){

//code block

}

1】类模板是一个通用类模型，而不是具体类，不能用于创建对象，只有经过实例化后才得到具体类，才能用于创建对象。

形式：类模板名 < 模板实参表 > ，如：Stack<int> stack\_int；

3.如何确定调用哪个函数？

* 静态绑定：编译时将一个函数调用关联到特定的函数体代码的过程。
* 函数调用的静态绑定规则
  1. 如果某一同名非模板函数的形参类型正好与函数调用的实参类型匹配（完全一致），则调用该函数。否则，进入第2步。
  2. 如果能从同名的函数模板实例化一个函数实例，而该函数实例的形参类型正好与函数调用的实参类型匹配（完全一致），则调用该函数实例。否则，进入第3步。
  3. 对函数调用的实参作隐式类型转换后与非模板函数再次进行匹配，若能找到匹配的函数则调用该函数。否则，进入第4步。
  4. 提示编译错误。

4.STL（Standard Template Library）

* 标准模板库，提供了模板化的通用类和通用函数；
* 容器（顺序容器、关联容器、容器适配器）
* 迭代器
* 算法

附、类图画法

类必须有三格，第一格表示类名，第二格表示数据成员，第三格表示成员函数。

+表示public

-表示private

#表示protected

<virtual>表示虚函数

数据成员注明数据类型

类的继承关系用箭头表示，从子类指向基类。