面向对象概述

面向过程思想

- 面向过程思想

  来回想一下，之前我们完成一个需求的步骤：首先是搞清楚要做什么，然后再分析怎么做，最后再通过代码体现。一步一步去实现，而具体的每一步都需要我们去实现和操作。这些步骤相互调用和协作，从而完成需求。在上面的每一个具体步骤中我们都是参与者，并且需要面对具体的每一个步骤和过程，这就是面向过程最直接的体现。

- 面向过程编程

  面向过程编程，其实就是面向着具体的每一个步骤和过程，把每一个步骤和过程完成，然后由这些功能函数相互调用，完成需求。

面向对象思想

- 面向对象思想

  面向对象的思想是尽可能模拟人类的思维方式，使得软件的开发方法与过程尽可能接近人类认识世界、解决现实问题的方法和过程，把客观世界中的实体抽象为问题域中的对象。面向对象以对象为核心，该思想认为程序由一系列对象组成。

- 面向对象编程（Object Oriented Programming，OOP）

- 面向对象思想的特点：

  1．是一种更符合人类思维习惯的思想

  2．可以将复杂的问题简单化

  3．将我们从执行者变成了指挥者

面向对象思想举例

- 洗衣服

  面向过程思想：1.往盆里接水 —— 2.放入衣服 —— 3.加入洗衣粉 —— 4.浸泡一会 —— 5.搓一搓 —— 6.取出过水 —— 7.拧干衣服 —— 8.晒衣服

  面向对象思想：1.把衣服放入洗衣机 —— 2.加入洗衣粉 —— 3.开启洗衣机 —— 4.晒衣服

- 吃饭

  面向过程思想：1.买菜 —— 2.择菜 —— 3.洗菜 —— 4.倒油到锅中 —— 5.把菜放入锅中 —— 6.炒菜 —— 7.加入调料 —— 8.把菜盛出 —— 9.吃

  面向对象思想：1.去饭店 —— 2.点菜 —— 3.吃

- 面向对象编程（OOP）的重要特性：

  抽象、封装和数据隐藏、多态、继承、代码的可重用性

Thanks

类和对象

抽象和类

- 编写程序的目的

  我们编写程序的目的就是为了模拟现实世界的事物，解决现实中的问题，实现信息化。

- 如何描述现实世界的事物

  属性：就是该事物的描述信息　　行为：就是该事物能够做什么

- 类是一种将抽象转换为用户定义类型的 C++ 工具，它将数据表示和操纵数据的方法组合成一个整洁的包。

  属性 —— 成员变量　　行为 —— 成员方法

- 类是一组相关的属性和行为的集合，对象是该类事物的具体体现。

定义类

- 一般来说，类规范由两部分组成：

  类声明：以数据成员的方式描述数据部分，以成员函数（被称为方法）的方式描述公有接口。

  类方法定义：描述如何实现类成员函数。

- 简单地说，类声明提供了类的蓝图，而方法定义则提供了细节。
- 通常，将接口（类声明）放在头文件中，并将实现（类方法的代码）放在源代码文件中。
- C++ 使用关键字 class 声明类。

Thanks

运算符重载概述

运算符重载

- 运算符重载(operator overloading)是一种形式的 C++ 多态。
- 运算符重载就是对已有的运算符重新进行定义，赋予其另一种功能，以适应不同的数据类型。
- 很多 C++ 运算符已经被重载，例如将 * 运算符用于地址，将得到存储在这个地址中的值；但将它用于两个数字时，得到的是它们的乘积。
- C++ 允许将运算符重载扩展到用户定义的类型。例如，允许使用 + 将两个对象相加。
- 运算符重载只是一种“语法上的方便”，可以使代码看起来更自然，是另一种函数调用的方式。
- 要重载运算符，需要使用被称为运算符函数的特殊函数形式。格式如下：

```
operatorop(argument-list)
```

运算符重载实现

```
int a = 10;
int b = 20;
int c = a + b;
```

```
Time t1(2, 30);
Time t2(1, 40);
Time t3 = t1 + t2;
```

```
class Time {
public:
  int hours;
  int minutes;
  Time(int h, int m) {
     hours = h;
     minutes = m;
  }
}
```

```
Time addTime(Time& t) {
  Time time;
  time.hours = hours + t.hours;
  time.minutes = minutes + t.minutes;
  return time;
}
```

```
Time t3 = t1.addTime(t2);
```

运算符重载实现

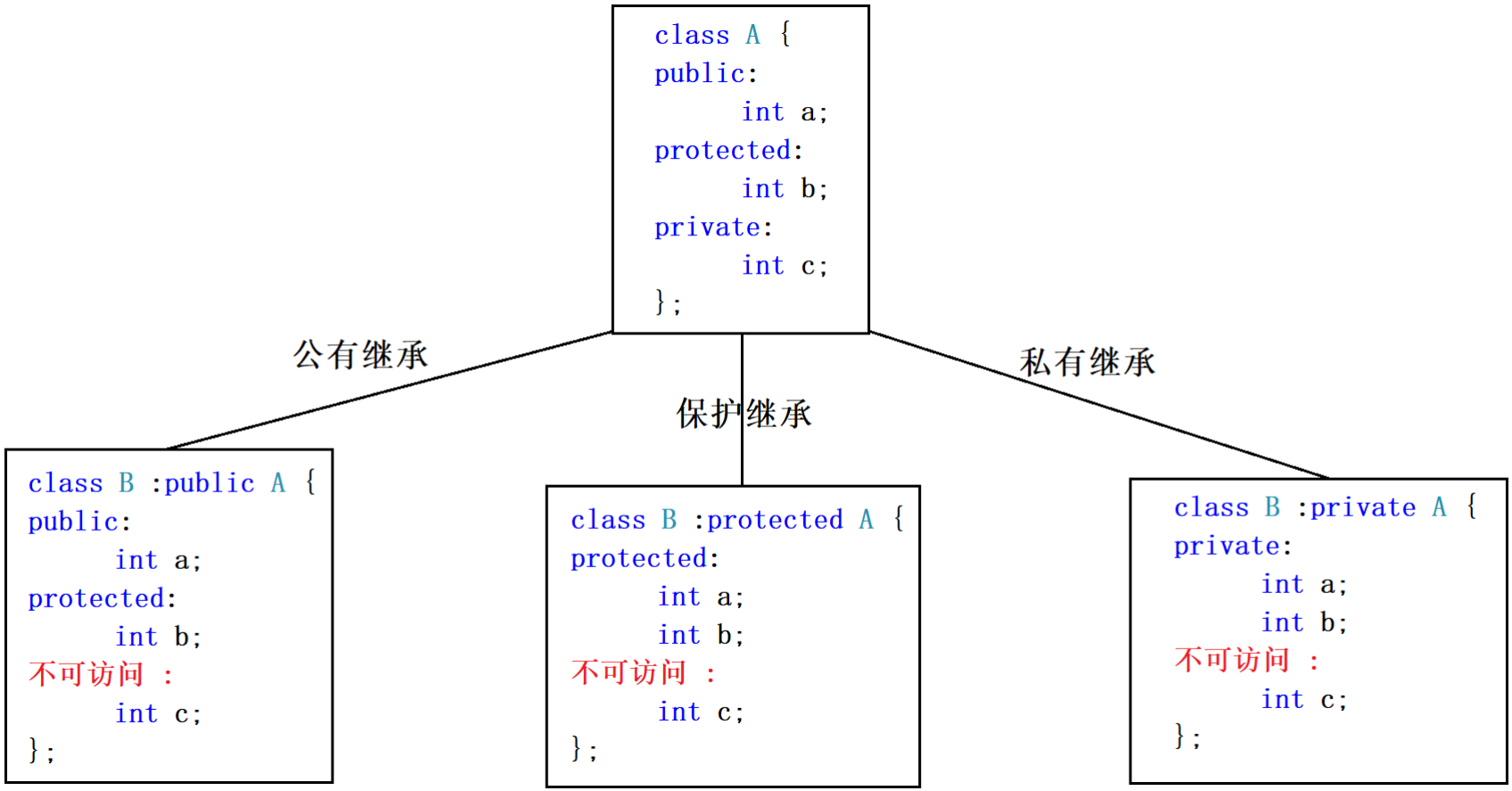
```
Time operator+(Time& t){
  Time time;
  time.hours = hours + t.hours;
  time.minutes = minutes + t.minutes;
  return time;
}
```

```
Time t1(2, 30);
Time t2(1, 40);
Time t3 = t1 + t2;
```

```
Time t3 = t1.operator+(t2);
```

```
// 全局函数实现运算符重载
Time operator+(Time& t1, Time& t2)
{
   // ...
}
```

Thanks


class A {
public:
int a;
protected:
int b;
private:
int c;
};
公有继承
保护继承
私有继承
class B :public A {
public:
int a;
protected:
int b;
不可访问 :
int c;
};
class B :protected A {
protected:
int a;
int b;
不可访问 :
int c;
};
class B :private A {
private:
int a;
int b;
不可访问 :
int c;
};

静态联编和动态联编

多态

- 多态是面向对象程序设计语言中数据抽象和继承之外的第三个基本特征。
- 多态性(Polymorphism)提供接口与具体实现之间的另一层隔离，从而将"what"和"how"分离开来。多态性改善了代码的可读性和组织性，同时也使创建的程序具有可扩展性，项目不仅在最初创建时期可以扩展，而且当项目在需要有新的功能时也能扩展。
- C++ 支持编译时多态(静态多态)和运行时多态(动态多态)，运算符重载和函数重载就是编译时多态，而派生类和虚函数实现运行时多态。
- 静态多态和动态多态的区别就是函数地址是早绑定(静态联编)还是晚绑定(动态联编)。

静态联编和动态联编

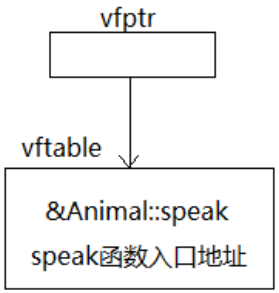
- 程序调用函数时，将使用哪个可执行代码块呢？编译器负责这个问题。
- 在C语言中，这非常简单，因为每个函数名都对应一个不同的函数。在 C++ 中，由于函数重载的缘故，这项任务更复杂。编译器必须查看函数参数以及函数名才能确定使用哪个函数。
- 将源代码中的函数调用解释为执行特定的函数代码块被称为函数名联编（binding）。
- 如果函数的调用，在编译阶段就可以确定函数的调用地址，并产生代码，就是静态联编（static binding），就是说地址是早绑定的，又称为早期联编（early binding）。
- 而如果函数的调用地址不能在编译期间确定，而需要在运行时才能决定，就是动态联编（dynamic binding），就是说地址是晚绑定，又称为晚期联编（late binding）。

Thanks

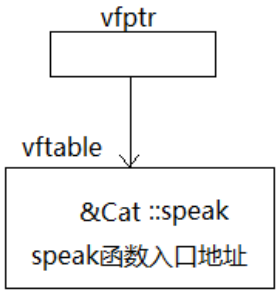
```cpp
// 动物类
class Animal {

public:

	// 虚函数 （采用的就是动态联编）
	virtual void speak() {
		cout << "动物在说话" << endl;
	}

};


// 猫类
class Cat :public Animal {

public:
	void speak() {
		cout << "猫在说话" << endl;
	}

};
```

当子类重写父类的虚函数后，那么子类的虚函数表入口地址就发生了覆盖（重写）。

Animal 类内部结构



Cat 类内部结构



vfptr 虚函数表指针
v virtual
f function
ptr pointer

vftable - 虚函数表
虚函数表记录了虚函数的函数入口地址

多态使用
父类指针或者引用指向子类对象

```cpp
Animal & animal = cat;
Animal * animal = new cat;

animal->speak();
```