C++

桂林電子科技大学
GUILIN UNIVERSITY OF ELECTRONIC TECHNOLOGY

人工智能学院
SCHOOL OF ARTIFICIAL INTELLIGENCE

第六讲 继承与派生

赵彬

zhaobin@guet.edu.cn

6.1 继承与派生

6.1.1 继承的概念

在C++中，在定义一个新类时，如果该类与某个已有类相似（即新类有已有类的全部特点），则可以让新类继承已有类，从而拥有已有类的属性和行为，并且可以在已有类的基础上修改或扩展新的属性和行为。此时，称已有类为基类（base class）或父类（super class）；称新类为派生类（derived class）或子类（sub class）。

可以说：子类继承了基类，或基类派生出了子类。

6.1继承与派生

6.1.2 多重继承

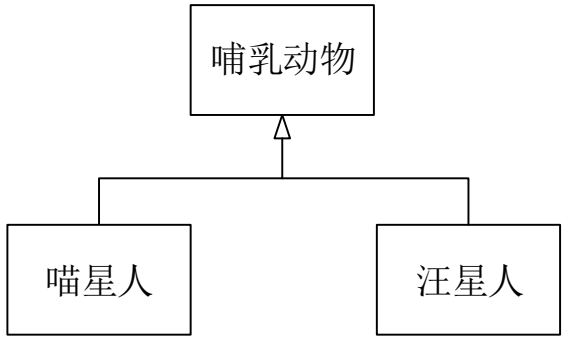


图1 单向继承

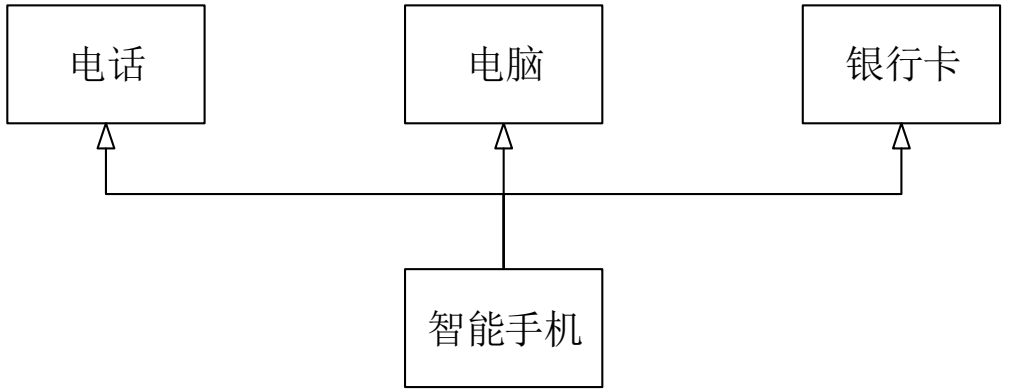


图2 多重继承

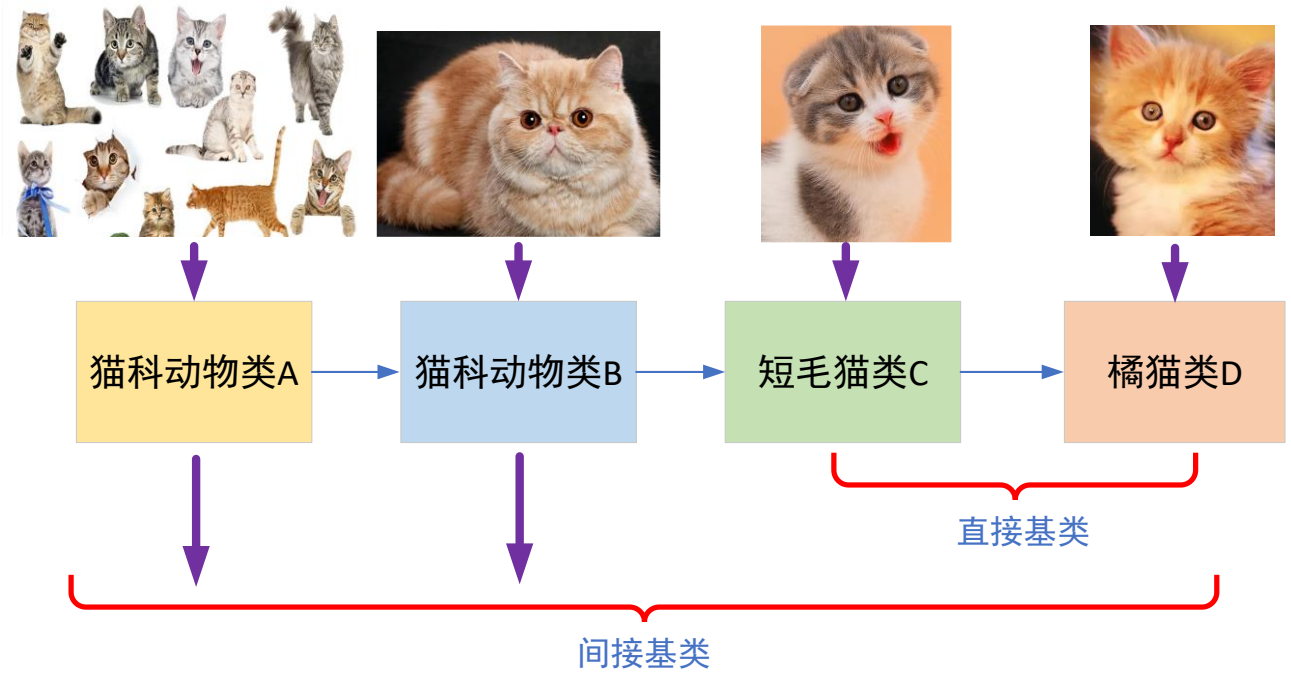
一个派生类也可以有两个或两个以上的基类，这样的继承方式称为多继承或多重继承（multiple inheritance,MI）

6.1继承与派生

6.1.3 多层继承（派生）

继承关系像一棵树一样，这种关系是可以传递的，继承得到的子类也可以作为基类，继续派生出新的子类，这样一层一层派生的方式称为多层派生，或者从继承的角度称为多层继承。最早的基类在最上层，最新的派生类在最底层。

在多层派生过程中，各层的基类、派生类之间形成了一个相互关联的类族。在这一类族中，派生类有直接基类和间接基类，直接基类是指直接派生出这个类的基类，而这个派生类的直接基类及其更高层的基类，都称为它的间接基类。



6.1继承与派生

6.1.3 多层继承（派生）

显然，多层继承（派生）是线性的，不能循环，即派生类不能成为其直接基类或间接基类的基类。

继承派生的优点：

C++的继承机制通过共享基类代码的方式，避免相似的类不断重复写相同的代码，继而避免“一处要改，处处要改”的冗余工作，不仅利于缩减代码量，更提高了对代码后期维护的效率，提升软件质量。

6.2 派生类的定义和构成

6.2.1 派生类的定义

派生类也是类，因此基本的定义格式与构成和类的定义与构成方式是一致的，二者的区别主要在于如何表达派生类与基类的关系。

派生类定义的语法为：

```
class 派生类名：继承方式1  基类名1, 继承方式2   基类名2,…
{
   private:
        派生类的私有数据和函数
   public:
        派生类的公有数据和函数
   protected:
        派生类的保护数据和函数
};
```

6.2 派生类的定义和构成

6.2.1 派生类的定义

- 单继承时，基类表中只有一个基类；多继承时，基类表中要列出所有的直接基类，基类之间用逗号分隔。
- 继承方式指定了派生类成员和外部对象对其从基类继承的成员的访问方式，继承方式有三种，即公有继承（public）、私有继承（private）和保护继承（protected）。
- 基类表中每个基类都按其继承方式继承，若不显示标明某基类的继承方式，则默认该基类为私有继承。

6.2 派生类的定义和构成

6.2.1 派生类的定义

一般来说，派生类的成员由3部分构成：

- 继承的基类成员

  基类的全部成员变量和成员函数都被派生类继承，作为派生类成员的一部分

- 改造基类成员得到的成员

  对基类成员的修改可以通过同名覆盖的方式进行，即派生类中的成员采用与基类成员相同的名称。这样定义的成员属于派生类的新成员，此时基类和派生类中有同名成员，但是表达了不同的内容。

  在基类和派生类有同名成员的环境下，若通过派生类生成的对象调用该同名成员，C++会自动调用派生类的成员，而不会调用从基类继承得到的同名成员，即调用的结果为派生类定义的属性或行为，以此达到改造成员的目的。

- 增加的新成员

  这部分成员变量和成员函数就与普通类的成员定义一样，都是当前的派生类描述某些新属性、新行为所需的。

6.2 派生类的定义和构成

6.2.1 派生类的定义

【例1】人和超级英雄类

```
class Human
{public:
void 生活（）；
void 工作（）；
void 娱乐（）；
private:
float 身高;
float 体重;
char 性别; };
class SuperHero: public Human
{public:
void 变身（）；
void 拯救世界（）；
void 恢复（）；
private:
int 超能力类型;
int 能力等级; };
```

表1 派生类成员构成

类名	成员名	
SuperHero	Human:	身高、体重、性别
		生活()
		工作()
		娱乐()
	超能力类型、能力级别	
	变身()	
	拯救世界()	
	恢复()	

6.3 继承的方式

6.3.1 公有继承

定义派生类时，继承方式为public的基类即为公有继承的，如有类Basel和类Derivedl定义如下：

```
class Basel {…};
class Derivedl:public Basel {…}
```

则Derivedl类为Basel类的公有派生类，或者Derivedl公有继承Basel类

- **公有继承的特点：**
  - 基类的公有成员在派生类中仍然为公有成员，可以由派生类对象和派生类成员函数直接访问。
  - 基类的私有成员在派生类中，无论是派生类的成员还是派生类的对象都无法直接访问。
  - 保护成员在派生类中仍是保护成员，可以通过派生类的成员函数访问，但不能由派生类的对象直接访问。

6.3 继承的方式

6.3.1 公有继承

表2 公有继承基类成员属性变化

基类	public	private	protected
公有派生类	public	不可访问	protected

6.3 继承的方式

6.3.1 公有继承

【例2】公有继承的成员访问权限

```
/*******************************************
程序名：human.h
功 能：human类定义
*******************************************/
#include <iostream>
using namespace std;
class human
{
public:
human(int H = 170, int W = 60, char G = 'M')
{
this->Height = H;
this->Weight = W;
this->Gender = G;
}
void Info()
{
cout << "身高，体重，性别：" << this->Height << ", " << this->Weight <<", " << this->Gender << endl;
}
```

6.3 继承的方式

6.3.1 公有继承

【例2】公有继承的成员访问权限（续）

```
void Live()
{
cout << "吃喝拉撒。" << endl;
}
void Work()
{
cout <<"认真工作！" << endl;
}
void Entertain()
{
cout << "逛吃宅嗨..." << endl;
}
private:
int Height;
int Weight;
char Gender;
};
```

6.3 继承的方式

6.3.1 公有继承

【例2】公有继承的成员访问权限（续）

```
/*******************************************
程序名：superhero.h
功 能：superhero类定义
*******************************************/
#include "human.h"
using namespace std;
class superhero:public human
{
public:
superhero(int T = 0, int Level = 2)
{
this->PowerType = T;
this->Level = Level;
}
void ShowInfo()
{
Info();
cout << "能力类型: " << this->PowerType <<"," << "能力等
级 : " << this->Level << endl;
}
```

6.3 继承的方式

6.3.1 公有继承

【例2】公有继承的成员访问权限（续）

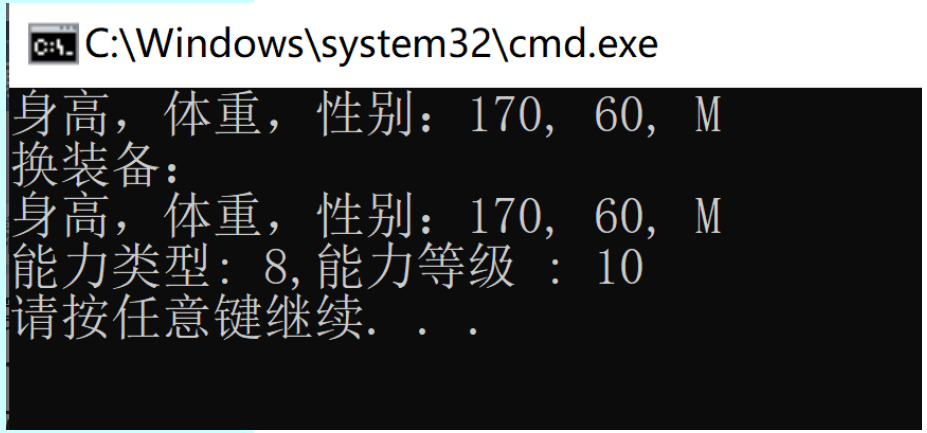
```
void SuitOn(int T, int L)
{
this->PowerType = T;
this->Level = L;
cout << "换装备： " << endl;
ShowInfo();
}

void SavingWorld()
{
cout << "打怪兽！ ！ ！ " << endl;
}
void SuitOff()
{
this->PowerType = 0;
this->Level = 2;
cout << "掩饰身份。" << endl;
}
private:
int PowerType;
int Level;
};
```

6.3 继承的方式

6.3.1 公有继承

【例2】公有继承的成员访问权限（续）

```
#include "superhero.h"
using namespace std;
int main()
{
superhero IronMan;
//cout << IronMan.Height;
IronMan.Info();
//cout << IronMan.PowerType;
IronMan.SuitOn(8, 10);
}
```



6.3 继承的方式

6.3.2 私有继承2122312

定义派生类时，继承方式为private的基类即为私有继承的，如有类Basel和类Derivedl定义如下：

```
class Basel {…};
class Derivedl:private Basel {…}
```

则Derivedl类为Basel类的私有派生类，或者Derivedl私有继承Basel类

- **私有继承的特点：**
  - 基类的公有成员和保护成员被继承后作为派生类的私有成员，即基类的公有成员和保护成员被派生类继承后，派生类的其他成员函数可以直接访问它们，但是在类外部，不能通过派生类的对象访问它们。
  - 基类的私有成员在派生类中不能被直接访问。无论是派生类的成员还是通过派生类的对象，都无法访问从基类继承来的私有成员。

6.3 继承的方式

6.3.2 私有继承

- **私有继承的特点：**
  - 经过私有继承之后，所有基类的成员都成为了派生类的私有成员或不可访问的成员，如果进一步派生的，基类的全部成员将无法在新的派生类中被访问。因此，私有继承之后，基类的成员再也无法在以后的派生类中发挥作用，实际是相当于中止了基类的继续派生，出于这种原因，一般情况下私有继承的使用比较少。如果要使用私有继承，通常要配合使用同名覆盖等方法。

表3 私有继承基类成员属性变化

基类	public	private	protected
私有派生类	private	不可访问	private

6.3 继承的方式

6.3.2 私有继承

【例3】私有继承的成员访问权限

```
/*******************************************
程序名：human.h
功 能：human类定义
*******************************************/
#include <iostream>
using namespace std;
class human
{
public:
human(int H = 170, int W = 60, char G = 'M')
{
this->Height = H;
this->Weight = W;
this->Gender = G;
}
void Info()
{
cout << "身高，体重，性别：" << this->Height << ", " << this->Weight <<", " << this->Gender << endl;
}
```

6.3 继承的方式

6.3.2 私有继承

【例3】私有继承的成员访问权限（续）

```
void Live()
{
cout << "吃喝拉撒。" << endl;
}
void Work()
{
cout <<"认真工作！" << endl;
}
void Entertain()
{
cout << "逛吃宅嗨..." << endl;
}
private:
int Height;
int Weight;
char Gender;
};
```

6.3 继承的方式

6.3.2 私有继承

【例3】私有继承的成员访问权限（续）

```
/*******************************************
程序名：superhero.h
功 能：superhero类定义
*******************************************/
#include "human.h"
using namespace std;
class superhero:private human
{
public:
superhero(int T = 0, int Level = 2)
{
this->PowerType = T;
this->Level = Level;
}
void ShowInfo()
{
Info();
cout << "能力类型: " << this->PowerType <<"," << "能力等
级 : " << this->Level << endl;
}
```

6.3 继承的方式

6.3.2 私有继承

【例3】私有继承的成员访问权限（续）

```
void SuitOn(int T, int L)
{
this->PowerType = T;
this->Level = L;
cout << "换装备： " << endl;
ShowInfo();
}

void SavingWorld()
{
cout << "打怪兽！ ！ ！ " << endl;
}
void SuitOff()
{
this->PowerType = 0;
this->Level = 2;
cout << "掩饰身份。" << endl;
}
private:
int PowerType;
int Level;
};
```

6.3 继承的方式

6.3.2 私有继承

【例3】私有继承的成员访问权限（续）

```
#include "superhero.h"
using namespace std;
int main()
{
superhero IronMan;
//cout << IronMan.Height;
IronMan.Info();
//cout << IronMan.PowerType;
IronMan.SuitOn(8, 10);
}
```

派生类对象无法直接调用

6.3 继承的方式

6.3.2 私有继承

若想运行成功，可利用同名覆盖原则，在派生类中增加一个同名的公有Info()成员，由于派生类的内部函数可以访问派生类的私有成员，因此再在该成员中调用基类的Info()的方法即可达到使用基类原有接口的目的。

```
#include “human.h"
using namespace std;
class superhero:private human
{
….;
void Info()
{
human::Info();
}
}
```

6.3 继承的方式

6.3.2 私有继承

- 利用同名覆盖的原则的优势:
  - 可以方便快捷地对基类成员的内容进行改造、扩充。
  - 可以使派生类能够更方便地在基类的基础上进行“继承性的创新”—接口不变，内容改变
  - 可以使基类成员隐藏信息，又可以被重复利用
  - 有利于提高代码的可读性和代码编写效率

6.3 继承的方式

6.3.3 保护继承

定义派生类时，继承方式为protected的基类即为保护继承的，如有类Basel和类Derivedl定义如下：

```
class Basel {…};
class Derivedl:protected Basel {…}
```

则Derivedl类为Basel类的保护派生类，或者Derivedl保护继承Basel类

- **保护继承的特点：**
  - 基类的公有成员和保护成员被继承后作为派生类的保护成员，即基类的公有成员和保护成员被派生类继承后，派生类的其他成员函数可以直接访问它们，但是在类外部，不能通过派生类的对象访问它们。
  - 基类的私有成员在派生类中不能被直接访问。无论是派生类的成员还是通过派生类的对象，都无法访问从基类继承来的私有成员。

6.3 继承的方式

6.3.3 保护继承

表4 公有继承基类成员属性变化

基类	public	private	protected
保护派生类	protected	不可访问	protected

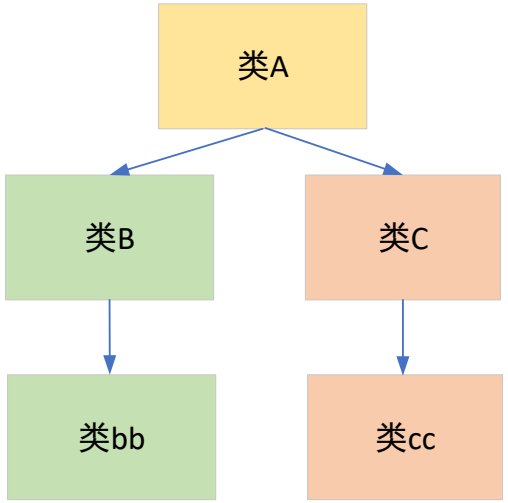
表5 继承基类成员访问控制属性总表

基类属性继承方式	public	protected	private
public	public	protected	不可访问
protected	protected	protected	不可访问
private	private	private	不可访问

6.3 继承的方式

6.3.3 保护继承

- 在类的存取控制属性中，protected属性和private属性对所修饰的成员本身的存取限制是相同的，即可以被类的内部成员访问，但不能通过类的对象被访问(即外部无法访问)，所以保护继承后，得到的派生类对基类成员的访问限制情况也与私有继承的情况一致
- 保护继承和私有继承的主要区别体现在多层派生的情况下，假设有基类A,类B为类A的保护继承类，类C为类A的私有继承类，若再往下派生，类B派生出类bb,类C派生出类cc，则无论类cc是以哪种方式继承的，都无法访问到类A的成员；基类A中的公有和保护成员都被类B以保护成员的属性间接继承，尽管类B对象不能访问这些成员，但是B的所有内部成员都可以访问它们，若bb是公有继承或保护继承自B，则A的那些特性仍然可以被保留下来。



6.4 派生类的构造与析构

6.4.1 派生类构造函数

- 以类外定义格式为例，派生类构造函数定义的一般格式为：

```
派生类名(参数总表): 基类名1(参数表1),...,基类名m (参数表m),成员对象名1(成员对象参数表1),...,成员对象名n(成员对象参数表n)
{
    派生类新增成员的初始化;
}
```

6.4 派生类的构造与析构

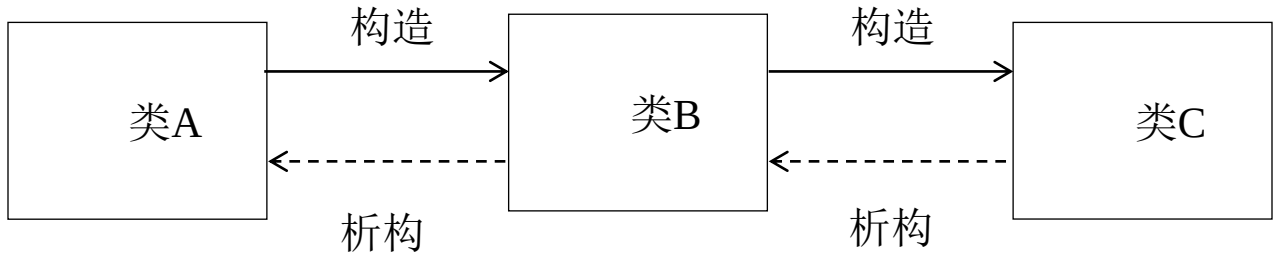
6.4.1 派生类构造函数

- 基类名1(参数表1),...,基类名m (参数表m)称为**基类成员的初始化表**。
- 成员对象名1(成员对象参数表1),...,成员对象名n(成员对象参数表n) 为**成员对象的初始化表**。
- 基类成员的初始化表与成员对象的初始化表构成派生类构造函数的**初始化表**。
- 在派生类构造函数的**参数总表**中，需要给出基类数据成员的初值、成员对象数据成员的初值、新增一般数据成员的初值。
- 在参数总表之后，列出需要使用参数进行初始化的基类名、成员对象名及各自的参数表，各项之间使用逗号分隔。
- 基类名、对象名之间的次序无关紧要，它们各自出现的顺序可以是**任意的**。在生成派生类对象时，程序首先会使用这里列出的参数，调用基类和成员对象的构造函数。

6.4 派生类的构造与析构

6.4.2 派生类的析构

派生类析构的顺序与构造的顺序正好相反，即在构造过程中先构造的，在析构过程中后析构。



6.4 派生类的构造与析构

6.4.3 单继承的构造

- 派生类从基类继承了成员，因此先要对基类成员进行初始化，避免发生未初始化就使用的错误。调用基类构造函数时，有以下两种方式：
  - 显示调用

在派生类的构造函数中，列出基类的带参构造函数，为基类的构造函数提供参数，格式如下：

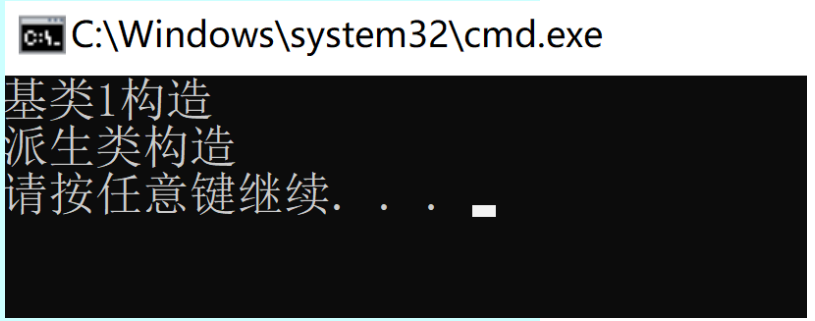
派生类名::派生类名(派生类参数列表):基类名(基类参数列表)

6.4 派生类的构造与析构

6.4.3 单继承的构造

【例4】派生类显示调用基类构造函数的示例代码

```
/*****************************************
*****************************************/
class Base {
public :
int n ;              //基类成员变量
Base(int i) : n(i)       //基类构造函数
{   cout <<“基类” <<n<< “构造” << endl ;  }
} ;
class Derived : public Base{
public:
Derived(int i) : Base(i)    //派生类构造函数
{   cout<<“派生类构造”<<endl;  }
} ;
//test code
int main()
{
        Derived Obj(1);       //派生类生成对象obj，传参数1
        return 0;
}
```



6.4 派生类的构造与析构

6.4.3 单继承的构造

➢ 隐式调用

在派生类的构造函数中，如果不列出基类构造函数，则派生类构造函数会自动调用基类的默认构造函数，即无参的构造函数，但是如果基类显示定义了带参构造函数后，系统不会给其分配默认的无参构造函数，此时编译器会报错。如例4的继承类修改为：

```
class Derived : public Base{
public:
Derived(int i)     //未显示调用基类构造函数
{   cout<<"派生类构造"<<endl;  }
};
```

- 因而，派生类生成对象时是先调用基类构造函数再调用派生类构造函数。

6.4 派生类的构造与析构

6.4.3 单继承的构造

➢ 隐式调用

当出现成员对象时，该类的构造函数要包含对成员的初始化。如果构造函数的成员初始化列表没有对成员对象初始化，则使用成员对象的默认构造函数。

【例5】派生类构造顺序

```
class A {
private:
        int nTa;
public:
        A(int ta);
};
class C{
private:
  int nTc;
public:
        C(int tc);
};
class B : public A{
private:
        int nTb;
        C obj1, obj2;
public:
        B(int ta,int tc1, int tc2, int tb);
};
```

```
B::B(int ta,int tc1, int tc2, int tb):A(ta), obj1(tc1), obj2(tc2), nTb(tb)
{
…
}
```

6.4 派生类的构造与析构

6.4.3 单继承的构造

- 单继承的情况下，派生类构造函数调用的一般顺序为：
  - 调用基类构造函数
  - 调用成员对象的构造函数，并且应该按照其在当前派生类中的定义顺序调用
  - 调用派生类自己的构造函数
  - 若继承（派生）层次有多层，则派生类构造函数的调用顺序将从最上层的基类开始，由上而下到最下层的派生类，依次按上面的一般调用顺序进行调用

6.5 多继承

6.5.1 多继承的构造与析构

多继承（multiple inheritance，MI）是指派生类具有两个或两个以上的直接基类（direct class）。

- 多继承时派生类构造函数执行的一般次序如下：
  - 调用各基类构造函数；各基类构造函数调用顺序按照基类被继承时声明的顺序，从左向右依次进行。
  - 调用内嵌成员对象的构造函数；成员对象的构造函数调用顺序按照它们在类中定义的顺序依次进行。
  - 调用派生类的构造函数；

☞注意:

- 在继承层次图中，处于同一层次的各基类构造函数的调用顺序取决于定义该派生类时所指定的各基类的先后顺序，与派生类构造函数定义时初始化表中所列的各基类构造函数的先后顺序无关。
- 对同一个基类，不允许直接继承两次。

6.5 多继承

6.5.1 多继承的构造与析构

- 多继承析构时，系统会先调用派生类的析构函数，析构派生类新增的普通成员；再调用成员对象所属类的析构函数，析构派生类中的成员对象；最后调用基类的析构函数，析构从基类继承的成员。

6.5 多继承

6.5.2 二义性问题

- 对多继承来说，由于继承自不同的基类，而被继承的基类之间原本没有关联，因此其内部定义时自成体系，没有考虑过是否同名的问题，这时，如果作为派生类用同一个成员标识符去调用，很可能会遇到几个基类都有同名成员可以被调用的情况，这就造成了编译器无法确定要调用到底是哪个成员。这种由于多继承引起的对某类的某成员进行访问时，不能唯一确定的情况，就称为二义性问题。

6.5 多继承

6.5.2 二义性问题

【例6】多继承的二义性问题

假设有一个汽车类和一个船类，现要求定义一个水陆两用车类。由于水陆两用车既具备车的特征，又有船的特征，因此可考虑让它继承汽车类和船类两个基类

```
#include <iostream>
using namespace std;
class Car
{
public:
    Car(int p, int s)
    {
            power = p;
            seat = s;
    }
    void Show()
    {
            cout<<"汽车动力:"<<power<<", 座位数： "<<seat<<endl;
    }
private:
    int power;//马力
    int seat;//座位
};
```

6.5 多继承

6.5.2 二义性问题

【例6】多继承的二义性问题（续）

```
class Boat
{
public:
    Boat(int c, int s)
    {
            capacity = c;
            seat = s;
    }
    void Show()
    {
            cout<<"船载重量： "<< capacity <<", 座位数： "<<seat<<endl;
    }
private:
    int capacity;//载重量
    int seat;//座位
};
```

6.5 多继承

6.5.2 二义性问题

【例6】多继承的二义性问题（续）

```
//水陆两栖车
class AAmobile:public Car, public Boat
{
public:
   AAmobile( int power, int cseat, int capacity, int bseat ):Car(power, cseat),Boat(capacity, bseat) { }
    void ShowAA()
    {
            cout<<"水陆两用车："<<endl;
            Car::Show();
            Boat::Show();
    }
};
int main()
{
    Car NormalCar(200,2);
    Boat NormalBoat(1000,2);
    AAmobile CoolCar(500,5,2000,4);
    NormalCar.Show();
    NormalBoat.Show();
    //CoolCar.Show();       //二义性
    CoolCar.ShowAA();
    return 0;
}
```

同名覆盖（ CoolCar.Show() ）

```
C:\Windows\system32\cmd.exe
汽车动力:200, 座位数：2
船载重量：1000, 座位数：2
水陆两用车：
汽车动力:500, 座位数：5
船载重量：2000, 座位数：4
请按任意键继续. . .
```

X

```
error C2385: 对“Show”的访问不明确
  note: 可能是“Show”(位于基“Car”中)
  note: 也可能是“Show”(位于基“Boat”中)
```

6.6 类型兼容

类型兼容是指在公有派生的情况下，一个派生类对象可以作为基类的对象来使用的情况。类型兼容又称为类型赋值兼容或类型适应。

- 在C++中，类型兼容主要指以下三种情况：
  - 派生类对象可以赋值给基类对象
  - 派生类对象可以初始化基类的引用，或者说基类引用可以直接引用派生类对象
  - 派生类对象的地址可以赋给基类指针，或者说基类指针可以指向派生类对象

6.6 类型兼容

【例7】兼容性代码示例

```
#include<iostream>
using namespace std;
class A
{
private:
  int na;
public:
A( int na )
{  this->na = na;  }
void Show ( )
{  cout << na<< endl;}
};
class B : public A
{
private:
  int nb;
public:
B ( int na, int nb ) : A( na )
{  this->nb = nb;  }
void Show ( )
{  cout <<nb<< endl;  }
};
```

```
int main ( )
{
  //对象调用所属类的成员函数
                    A a (1);
                    a.Show ( );            //1
                    B b( 200,200 );
                    b.Show ( );          //200
                    cout << endl;
              //子类对象调用父类的成员
               b.Show ( );            //200
               cout << endl;

  //基类指针指向不同对象
                    A *pa = NULL;
                    pa = &a;              //基类指针指向基类对象
                    pa-> Show ( );          //1
               pa = &b;                   //基类指针指向派生类对象
               pa->Show( );               //200
               cout<<endl;
 //派生类对象赋值给基类对象
    a = b;
    a.Show ( );            //200
    cout<<endl;
    //基类引用派生类对象
    A &ra = b;
    ra.Show ( );           //200
                    return 0;
}
```

6.6 类型兼容

【例8】兼容性代码示例2（基于例7修改，A和B一致，增加Display外部函数，修改main函数

```
void Display(A aobj)
{
aobj.Show();
}
int main()
{
A a(1);
B b(200,200);
Display(a);
Display(b);
return 0;
}
```

- ✓ C++提供的类型兼容规则使编译器可以自动在派生类和基类之间进行隐式的类型转换，使派生类对象也可以调用成功，无需手动增加重载代码
- ✓ 可以隐式地将基类和派生类的对象、指针、引用进行类型转换，使得基类的指针（对象\引用）可以访问不同的派生类对象
- ✓ 只能访问派生类中从基类继承到的成员，不能访问派生类的新增成员

6.7 虚基类

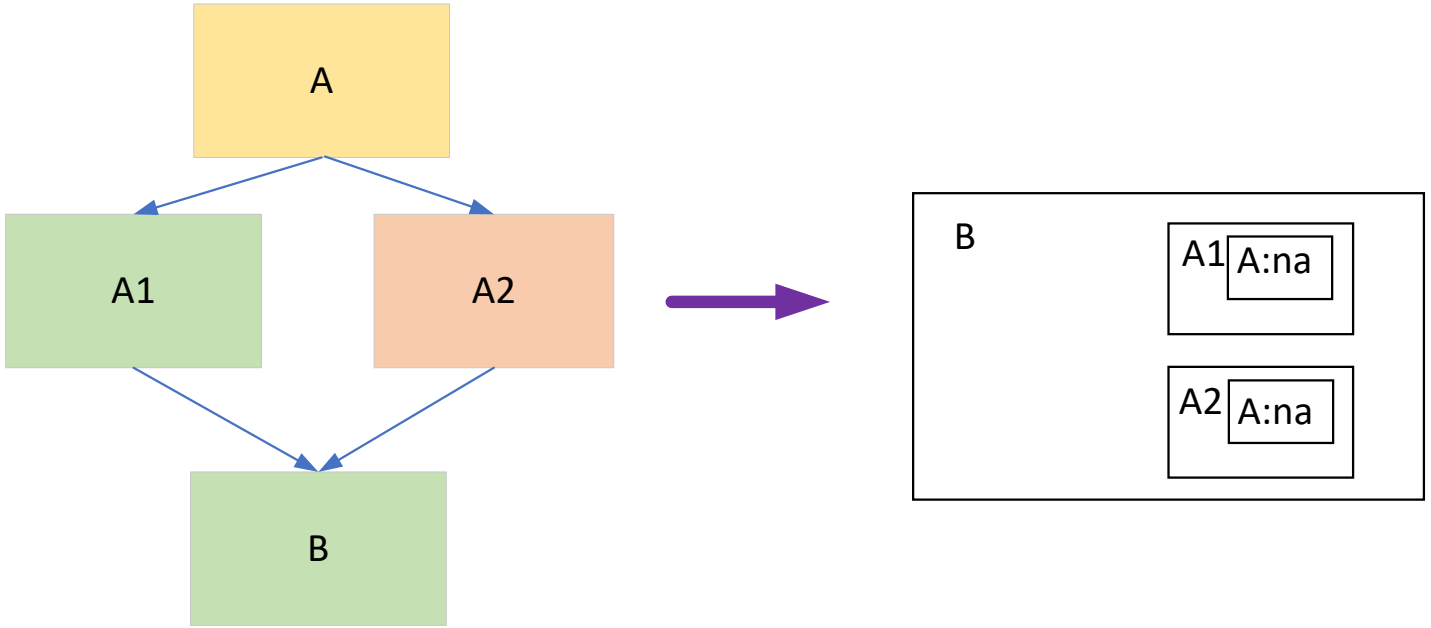
- 6.5节中讲到的多继承中多个基类有同名成员时带来的二义性问题称为直接二义性问题，可以通过同名覆盖等方法解决
- 在多继承中，当派生类的两个或以上直接基类都是从一个共同的基类派生出来的，那么这些直接基类中就都包含从上层基类继承来的相同成员。当派生类生成了对象，则从不同直接基类中继承的同名成员在内存中就同时拥有多个副本（拷贝），这种情况称为间接二义性。

【例9】间接二义性代码示例

```
#include<iostream>
using namespace std;
//基类A
class A {
public:
        int na;
        A(int a) {  na = a;  }
} ;
//一级派生类
class A1 :public A
{public:  A1(int a) : A(a){ };  } ;
class A2 :public A
{ public:  A2(int a) : A(a){ };  } ;
//二级派生类
class B:public A1,public A2
{ public:  B(int a) : A1(a) , A2(a) { };  } ;
```

6.7 虚基类

例9的继承关系图:



- 如果生成一个B的对象b,通过b访问na时将会发生间接二义性问题

6.7 虚基类

6.7.1 虚基类的定义

虚基类并不是完全有别于普通类而独立存在的一种类的定义形式，它是在派生类的定义中，

用关键字virtual对继承方式进行修饰得到的，定义格式如下：

```
class 派生类名: virtual 继承方式 基类名
```

说明：

- 在某基类的继承方式前使用virtual关键字时，说明该基类为此派生类的虚基类。
- virtual关键字只限定紧跟其后的一个类。
- 派生类声明虚基类后，其效果对后续的派生一直有效。

6.7 虚基类

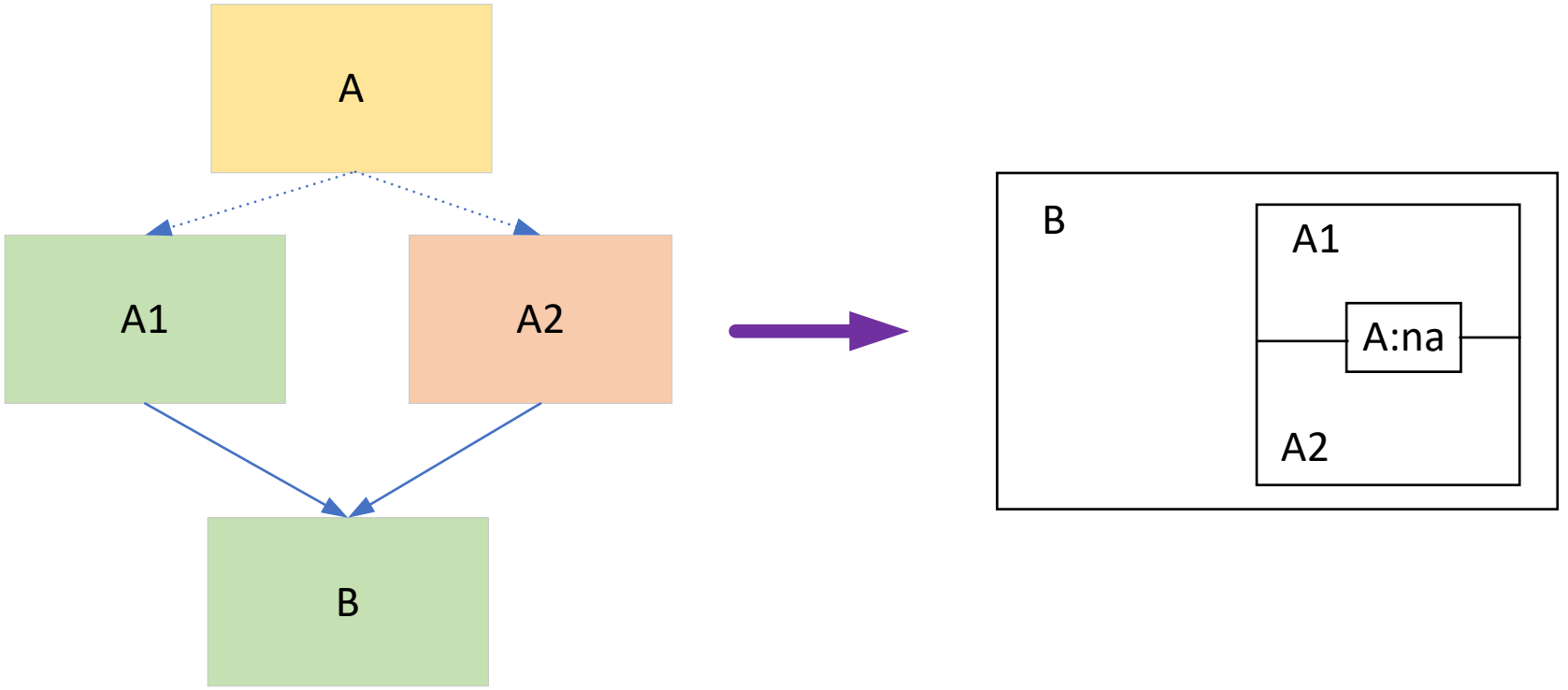
6.7.1 虚基类的定义

【例10】虚基类代码示例

```
//基类A
class A {
public:
        int na;
        A ( int a = 0 ) { na = a; }
};
//一级派生类
class A1 :virtual public A
{
public:
        A1(int a) : A(a) { }
};
class A2 : virtual public A
{
public:
        A2(int a) : A(a) { }
};
//二级派生类
class B : public A1,public A2
{
 public:
  B(int a) : A1(a) , A2(a) { }
};
```

6.7 虚基类

6.7.1 虚基类的定义



- ✓ 虚基类继承时基类必须有默认构造函数，如果没有则编译出错，因而例10中基类构造函数要设置参数默认值

6.7 虚基类

6.7.2 虚基类的构造

- 存在虚基类时，对象构造函数的调用顺序与一般情况有所不同，一般主要遵循以下规律:
  - 虚基类的构造函数在非虚基类构造函数之前调用
  - 若同一个继承层次中包含多个虚基类，则按照他们的声明次序依次调用虚基类的构造函数。
  - 若虚基类是有非虚基类派生的，则遵守先调用基类构造函数，再调用派生类构造函数的规则。

【例11】虚基类构造顺序代码示例

```
class A1 {...};
class A2 {...};
class B1 : public A2,virtual public A1 {...};
class B2:public A2, virtual public A1 {...};
class C : public B1, virtual public B2 {...};
```

C实例化一个对象，构造函数的调用顺序为：A1→A2→B2→A2→B1→C

6.7 虚基类

6.7.3 虚基类的构造与析构

- 虚基类的构造函数调用分三种情况：

(1) 虚基类没有定义构造函数

程序自动调用系统缺省的构造函数来初始化派生类对象中的虚基类子对象。

(2) 虚基类定义了缺省构造函数

程序自动调用自定义的缺省构造函数和析构函数。

(3) 虚基类定义了带参数的构造函数

直接或间接继承虚基类的所有派生类，都必须在构造函数的初始化表中列出对虚基类的初始化。

- 与单继承析构顺序与构造顺序相反的情况类似，虚基类的析构顺序也与其构造顺序相反，先构造的后析构，后构造的先析构。