继承

Andrew Huang<bluedrum@163.com>

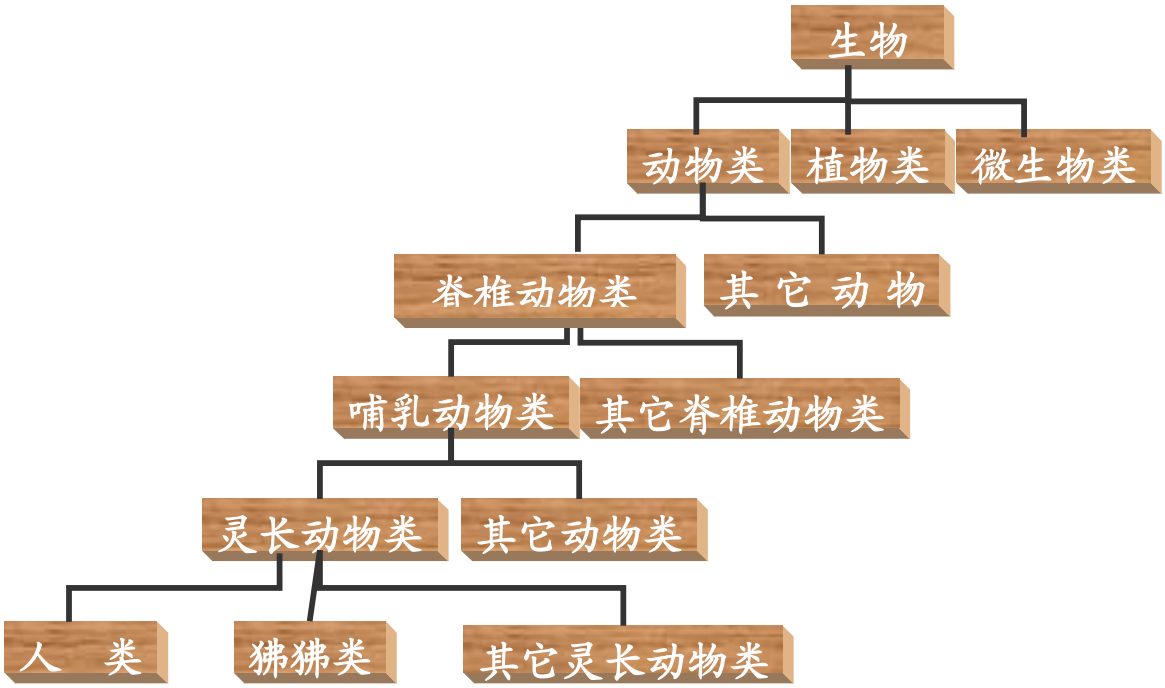
课程内容

- 继承的概念
- 派生类与基类的访问控制
- 派生类的构造和析构

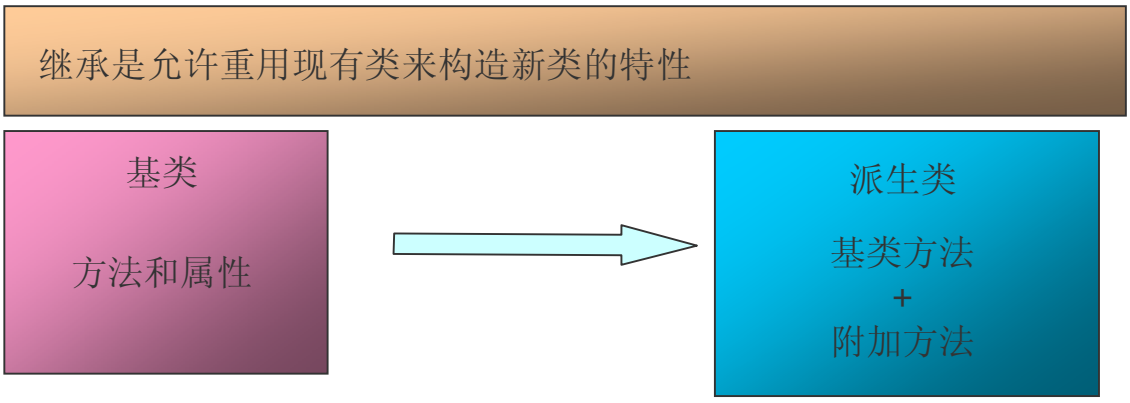
继承的概念

继承的概念

- 自然界广泛的存在继承的概念



C++ 的继承



C++ 继承的优点

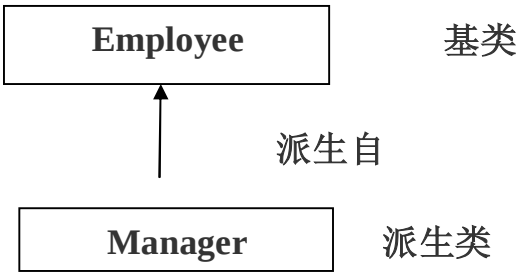
- 通过继承机制，可以利用已有的数据类型来定义新的数据类型。所定义的新的数据类型不仅拥有新定义的成员，而且还同时拥有旧的成员。则已存在的用来派生新类的类为**基类**。由已存在的类派生出的新类称为**派生类**。
- 提供一种健壮的代码重用机制,这样大规模类库就是通过继承组合起来
  - MFC,VCL
- **提供动态绑定功能,修改基类的方法,派生类会自动生效**
- 继承子类无需了解父类的细节

继承的模式

- 一个派生类可以从一个基类派生，也可以从多个基类派生。从一个基类派生的继承称为**单继承**；从多个基类派生的继承称为**多继承**。
- 单一继承最简单,也最为常用.

基类和派生类

- 派生类的声明必须指定基类的名称
  `class Manager : public Employee`
- 任何类都能用作基类
- 基类分为两种类型
  - 直接基类
  - 间接基类



直接基类和间接基类

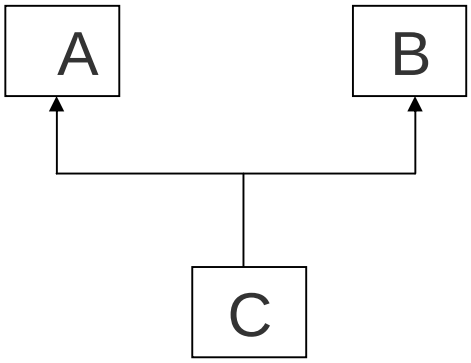
- 直接基类

```
class A
{};
class B : public A          //A 是 B 的直接基类
{};
```

- 间接基类

```
class A
{};
class B : public A
{};
class C : public B          //A 是 C 的间接基类
{};
```

多重继承



```
class A
{…};
class B
{…};
class C :public A, public B
{…};
```

继承基本格式

单继承的定义格式为：

```
class  派生类名 ： 继承方式  基类名
   {
          派生类新定义成员
   };
```

多继承的定义格式为：

```
class  派生类名 ： 继承方式1  基类名1，
                   继承方式2  基类名2，…
   {
          派生类新定义成员
    };
```

规定基类成员在派生类中的访问权限

public:公有派生
private:私有派生
protected:保护派生

访问控制

访问控制

- 派生类的函数
  - 能够访问基类的保护和公有成员
- 派生类的对象
  - 公有派生的类的对象能够访问基类的公有成员
  - 私有和保护派生的类的对象不能访问基类的任何成员

protected访问控制说明符

- 保护部分类似于私有部分

- 只能被其所属类的成员访问
- 不能被类外部的对象或函数访问
- 区别只有在派生类中才会表现出来

派生类的三种继承方式

- 在定义类于需要标明继承方式，分别是公有继承，私有继承和保护继承

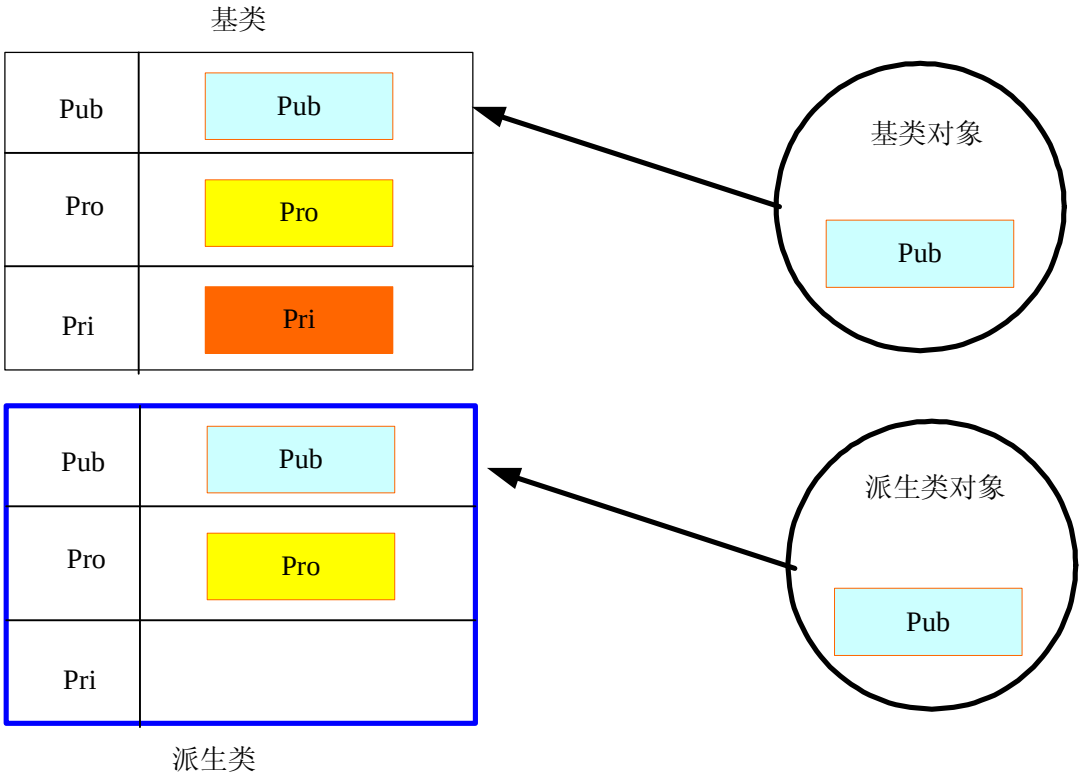
```
class A
{ };
class B : public A          //b 从 A 公有继承
{ };
class C :private A //C 从 A 私有继承
{}
class D:protected A//D 从 A 保护继承
```

- 如无例外,C++ 绝大部分是属于公有继承

公有继承

- **公有继承**时，派生类的对象只可访问基类中的公有成员，不能访问其它成员。派生类的成员函数可以访问基类中的公有成员和保护成员，不可访问其私有成员。
- **其特点**是：基类的**公有成员**和**保护成员**作为派生类的成员时，都**保持原有状态**，而**私有成员**仍是**私有**。

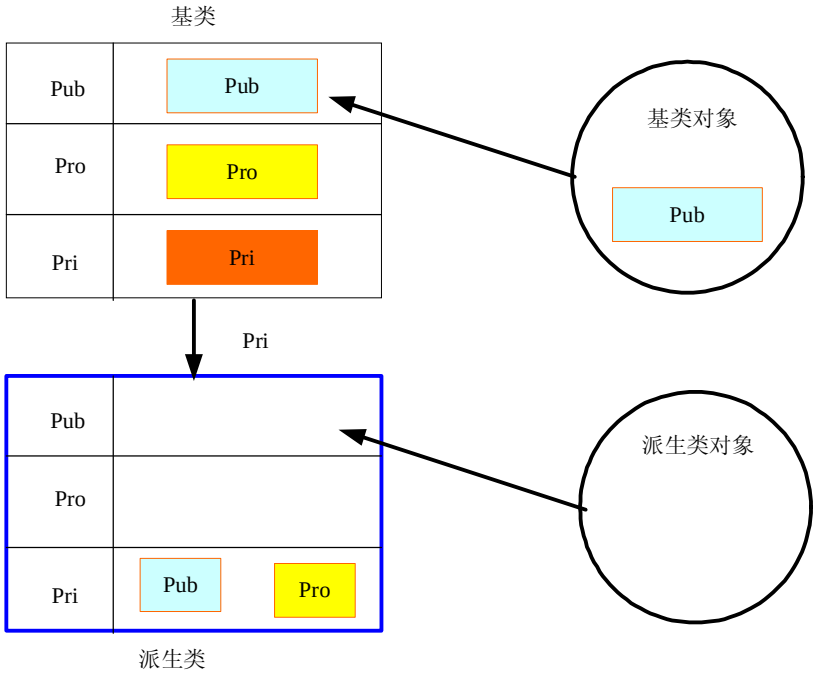
公有继承



私有继承

- **私有继承时，基类的成员只能由直接派生类访问，而无法再往下继承。**
- 其**特点**是：基类的**公有成员**和**保护成员**都作为派生类的**私有成员**，并且**不能**被这个**派生**类的子类所**访问**

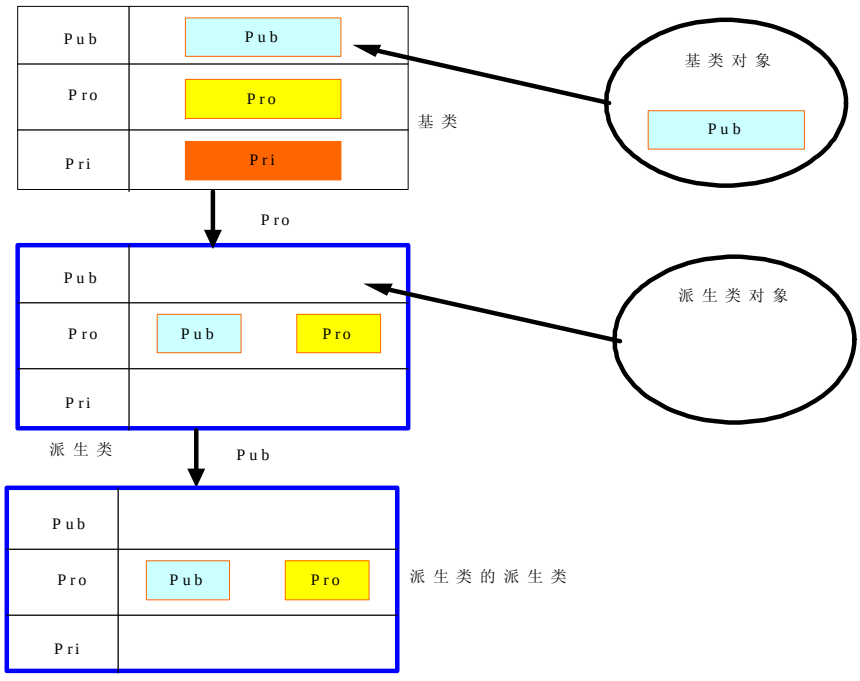
私有继承



保护继承

- 其**特点**是：基类的所有**公有成员**和**保护成员**都作为派生类的**保护成员**，并且只能被它的派生类**成员函数**或**友元**访问，基类的**私有成员**仍是**私有**的。

保护继承



不同继承方式对派生类的访问影响

基类成员	公有继承	私有继承	保护继承
公有	公有	私有	保护
保护	保护	私有	保护
私有	不被继承	不被继承	不被继承

访问控制

- 派生类不能访问基类的私有成员
- 公有继承不改变基类成员的访问级别
- 类成员总是能够被它们自己的类的方法访问
- 继承类能访问基类的公有或保护成员
- 公有成员可以在任何地方被访问
- **派生类如何访问基类的私有数据?**
  - 只能通过基类的保护或公有方法访问

派生类的函数

派生类的构造

- 当创建派生类对象时,总是先执行基类的构造,再执行派生类的构造.
- 如果在派生类的构造没有指明哪一个基类构造,则使用缺省默认的构造函数

派生类的构造函数

- 从理论讲,派生类构造前应该先保证基类被构造出来.所以C++在派生类构造时,要求调用基类构造函数.
- 如果没有特别机制,在C++在对象的嵌入的子对象只能采用缺省构造机制.如果需要带参数的构造机制,也需要用一些特别的办法
- 所以C++提供下列派生类的格式

```
<派生类名>(<派生类构造函数总参数表>):
                        <基类构造函数>(<参数表1>),
                        <子对象名>(<参数表 2>).

{
        <派生类中数据成员初始化>
};
```

```
class CEngineer{
   public:
     CEngineer(double capacity){}
};
class CWheels{
   public:
     CWheels(int count){}
};
//一个轿车包含轮胎和引擎,这是组合关系
class CCar{
public:
  int height;
  int width;
  CCar(int w) :Engineer(0.8),Wheels(4)
{width = w;}
  CEngineer Engineer;
  CWheels Wheels;
};
```

```
class CQQ:public CCar{
   public:
     CQQ(int w);
};

CQQ::CQQ(int w):
CCar(w )
{
   height=160;
}

CQQ qq(80);
```

析构函数调用

当一个派生类对象撤消时，由于**析构函数也不能被继承**，因此在执行派生类的析构函数时，基类的析构函数也将被调用。析构函数调用的顺序与构造函数相反；即先执行派生类的析构函数撤消派生类自身，再撤消其成员，最后，再执行基类的析构函数撤消基类。

函数覆盖

- 派生类的函数覆盖基类的同名函数
  - 通过派生类的对象调用时，执行派生类的函数
  - 用基类的对象调用时，执行基类的函数
- 派生类的成员函数要调用基类的同名函数，必须使用作用域解析操作符: :

```
class A{
    void print() { cout<< “a”; }
};
class B{
    void print() {
       A::print();
       cout << “b”; };
};
```

多重继承

多重继承

- 是指派生类具有多个基类，派生类与每个基类之间的关系仍可看作是一个单继承。
- **定义格式在开始已经给出。**
- **如：**

```
class A {¡-};
class B {¡-};
```

```
class C :public A,public B
  {…};
```

- 根据继承的规定，派生类C的成员包含了基类A中成员和基类B中成员及该类本身的成员。

多重继续实例

例：

```
#include< iostream.h>
class B1 { public:
   B1(int I) {b1=I; cout<<“constructor B1.”<<I<<endl; }
   void print(){cout<<b1<<endl;}
private:        int b1;
                    };//定义基类 B1
class B2 { public:
   B2(int I) {b2=I; cout<<“constructor B2.”<<I<<endl; }
   void print(){cout<<b2<<endl;}
private:        int b2;
                    };//定义基类 B2
class B3 { public:
   B3(int I) {b3=I; cout<<“constructor B3.”<<I<<endl; }
   int getb3{return b3;}
private:        int b3;
                    };//定义基类 B3
class A：public B2,public B1 {                  //定义派生类 A
      public:
      A(int i,int j,int k,int m):B1(i),B2(j),bb(k) //派生类构造函数
        {a=m; cout<<“constructor A.”<<l<<endl; }
     void print()
        {
          B1::print();          B2::print();//调用不同基类中的同名函数
          cout<<a<<“,”<<bb.getb3()<<endl;
        }
private:
        int a;
        B3 bb;      //定义子对象 bb
          };
void main ()
{          A aa(1,2,3,4);
           aa.print();
   }
```

运行结果：

```
           constructor B2.2
           constructor B1.1
           constructor B3.3
           constructor A.4
           1
           2
           4 ，  3
```

多重继承的问题

- 缺点：
  - 1．基类函数同名发生二义性冲突

- 2．运行效率低下
- 3．类结构复杂．

- 为此，很多开发语言取消了多重继承的语法
  - 如Java,Delphi 就只有单继承
- 任何用多重继承的设计都可以转换成其它设计
  - 如主从的类组合，例如CQQ

对象组合

对象组合

- 是指对象中嵌套对象.与继承一样,也是一种应用很广的设计方法
- 以象棋棋谱为例设计一个类
  - 双向链表结构或类作为棋谱成员
- C++可以从结构里继承出一个类

组合对象的构造

- 在父类的构造函数里调用组合的构造函数
- 或在构造函数,初始化相关数据.

```
<派生类名>(<派生类构造函数总参数表>):
<子对象名>(<参数表 2>).

{
        <派生类中数据成员初始化>
};
```

课堂练习

- 从CDate 继承下一新的子类CDateTime，加入时间支持
  - 1．用三个整数表示小时，分钟和秒
  - 2．加入显示完整时间函数
  - 3．加入设置时间函数
  - 4．加入增加N 秒的操作符重载
  - 5．实现函数，可以参考 afx.h 和 VC98/MFC/src/timecore.cpp 关于CTime 的类实现