CSDN 学院

C++语言基础

迁者 - 贺利坚

http://blog.csdn.net/sxhelijian/

http://edu.csdn.net

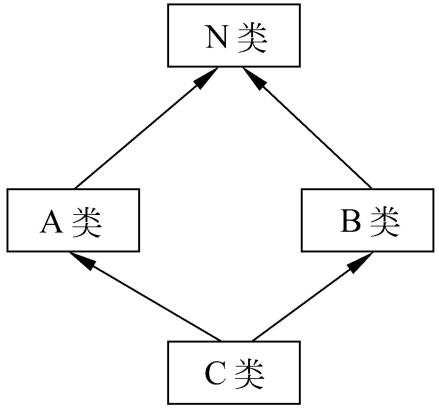
本节主题：

虚基类及应用

二义性(3): 两个基类从同一个基类派生

```
class N
{
public:
   int a;
   void display(){
      cout<<"A::a="<<a<<endl;
   }
};
```

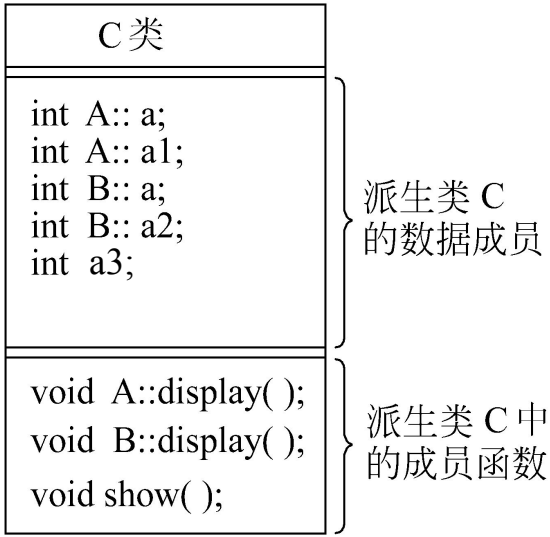
```
class C :public A,public B
{
public :
    int a3;
   void show( ){
       cout<<"a3="<<a3<<endl;
   }
};
```

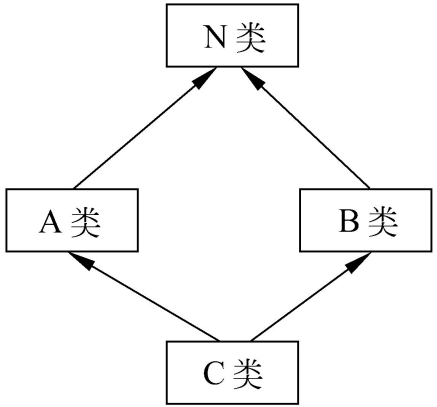


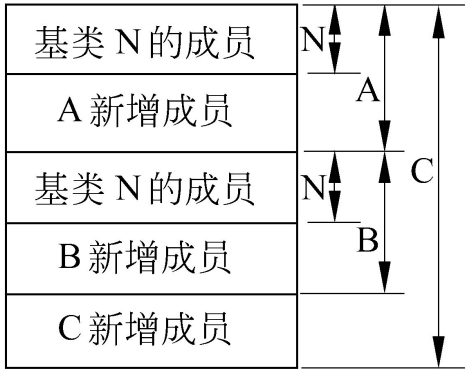

```
class A:public N
{
public:
    int a1;
};
```

```
class B:public N
{
public:
     int a2;
};
```

派生类C中成员的情况





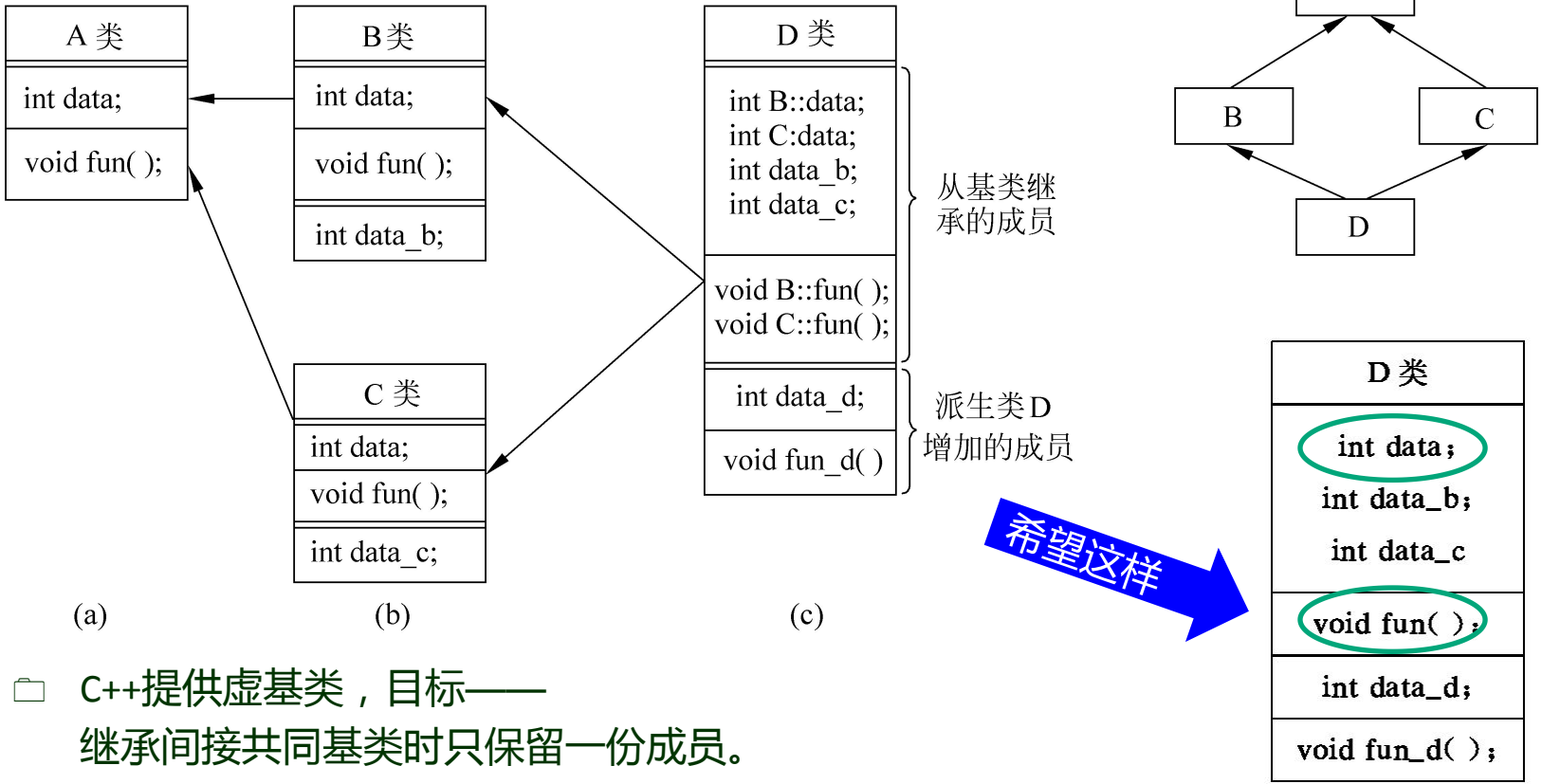


A::a和B::a是N类成员的拷贝
A::a和B::a占用不同的空间

```
//合法访问
c1.A::a=3;
c1.B::a=5;
c1.B::display();
```

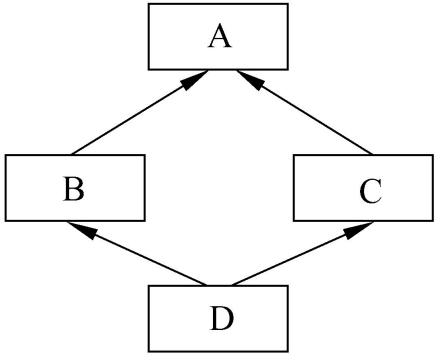
```
//非法访问
c1.a=3;
c1.display( );或
c1.N::a=3;
c1.N::display();
```

虚基类(virtual base class)的提出

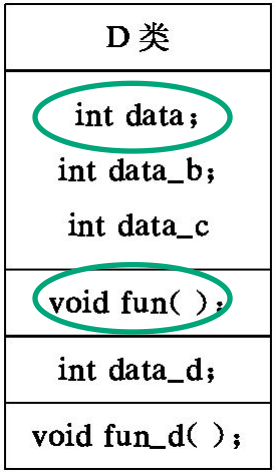


- C++提供虚基类，目标——
  继承间接共同基类时只保留一份成员。

虚基类定义的一般形式




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



语法：class 派生类名: **virtual** 继承方式 基类名
效果：当基类通过多条派生路径被一个派生类继承时，该派生类只继承该基类一次。

虚基类的初始化

- 通过构造函数的初始化表对虚基类进行初始化。例如

```
class A
{
    A(int i) { }
    …
};
```

```
class D: public B, public C
{
    D(int n):A(n),B(n),C(n) { }
    …
};
```

```
class B: virtual public A
{
    B(int n):A(n) { }
    …
};
```

```
class C: virtual public A
{
    C(int n):A(n) { }
    …
};
```

- 规定: 在最后的派生类中不仅要负责对其直接基类进行初始化，还要负责对虚基类初始化。
- C++编译系统只执行最后的派生类对虚基类的构造函数的调用，而忽略虚基类的其他派生类(如类B和类C) 对虚基类的构造函数的调用，这就保证了虚基类的数据成员不会被多次初始化。

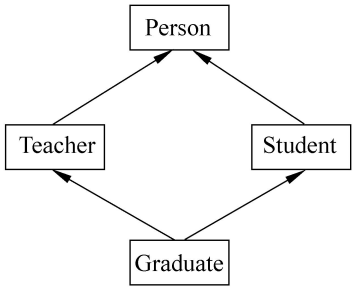
虚基类应用举例

```
class Person
{
public:
   Person(char *nam,char s,int a)
   {
      strcpy(name,nam);
      sex=s;
      age=a;
   }
protected:
   char name[20];
   char sex;
   int age;
};
```

```
class Graduate:public Teacher,public Student
{
public:
   Graduate(char *nam,char s,int a,char *t,float sco,float w):
      Person(nam,s,a),Teacher(nam,s,a,t),Student(nam,s,a,sco),wage(w) {}
   void show( )
   {
      cout<<"name:"<<name<<endl;
      cout<<"age:"<<age<<endl;
      cout<<"sex:"<<sex<<endl;
      cout<<"score:"<<score<<endl;
      cout<<"title:"<<title<<endl;
      cout<<"wages:"<<wage<<endl;
   }
private:
   float wage;
}; //工资
```

```
class Teacher:virtual public Person
{
public:
   Teacher(char *nam,char s,int a,char *t):Person(nam,s,a)
   {
      strcpy(title,t);
   }
protected:
   char title[10];
};
```

```
class Student:virtual public Person
{
public:
   Student(char *nam,char s,int a,float sco):
      Person(nam,s,a),score(sco) {}
protected:
   float score;
};
```




```
int main( )
{
   Graduate grad1("Wang",'f',24,"assistant",89.5,1234.5);
   grad1.show( );
   return 0;
}
```

谨慎使用多重继承

- 使用多重继承时要十分小心，否则会经常出现二义性问题。
- 许多专业人员认为
  - 不提倡在程序中使用多重继承
  - 只有在比较简单和不易出现二义性的情况或实在必要时才使用多重继承
  - 能用单一继承解决的问题就不要使用多重继承
- 有些面向对象的程序设计语言(如Java，Smalltalk) 并不支持多重继承。

CSDN 学院

THANKS

本课程由 迁者-贺利坚 提供

CSDN网站：www.csdn.net
企业服务：http://ems.csdn.net/
人才服务：http://job.csdn.net/
CTO俱乐部：http://cto.csdn.net/
高校俱乐部：http://student.csdn.net/
程序员杂志：http://programmer.csdn.net/

CODE平台：https://code.csdn.net/
项目外包：http://www.csto.com/
CSDN博客：http://blog.csdn.net/
CSDN论坛：http://bbs.csdn.net/
CSDN下载：http://download.csdn.net/