

## 第6讲

# 继承与派生

主讲人：赵文彬

## 本章主要内容

- 继承和派生的概念
- 继承方式
  - 公有继承
  - 保护继承\*
  - 私有继承\*
- 派生类的构造与析构

```
//student.h
```

```
#pragma once
```

```
#include <string>
```

```
#include <iostream>
```

```
using namespace std;
```

```
class student
```

```
{public:
```

```
    student(void);
```

```
    ~student(void);
```

```
    void setValues(int n,  
string str, char c);
```

```
    void display();
```

```
protected:
```

```
    int num;string name;
```

```
    char sex;};
```

```
//student.cpp
```

```
#include "student.h"
```

```
student::student(void){}
```

```
student::~~student(void){}
```

```
void student::setValues(int n,  
string str, char c)
```

```
{num = n;name = str; sex = c;  
}
```

```
void student::display()
```

```
{  
    cout << num << " " << name <<  
" " << sex <<  
    endl;  
}
```

```
//postgraduents.h
```

```
#pragma once
```

```
#include "student.h"
```

```
class postgraduate : public  
student
```

```
{
```

```
public:
```

```
    postgraduate(void);
```

```
    ~postgraduate(void);
```

```
    void setAdvisor(string  
str){advisor = str;}  
    string getAdvisor(){return  
advisor;}
```

```
private:
```

```
    string advisor;
```

```
};
```

```
//postgraduate.cpp
```

```
#include "postgraduate.h"
```

```
postgraduate::postgraduate(void)
```

```
{
```

```
}
```

```
postgraduate::~~postgraduate(voi  
d)
```

```
{
```

```
}
```

研究生是学生，具有(继承)学生的所有属性  
研究生有自己的导师

# 学生和研究生

```
//main.cpp
```

```
#include "postgraduent.h"
```

```
void main()
```

```
{
```

```
    postgraduent xq;
```

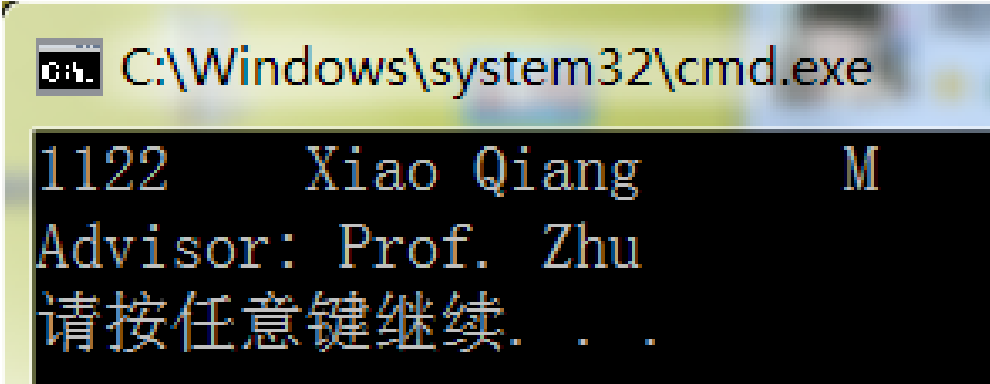
```
    xq.setValues(1122, "Xiao Qiang", 'M');
```

```
    xq.setAdvisor("Prof. Zhu");
```

```
    xq.display();
```

```
    cout << "Advisor: " << xq.getAdvisor() << endl;
```

```
}
```



C:\Windows\system32\cmd.exe

1122 Xiao Qiang M

Advisor: Prof. Zhu

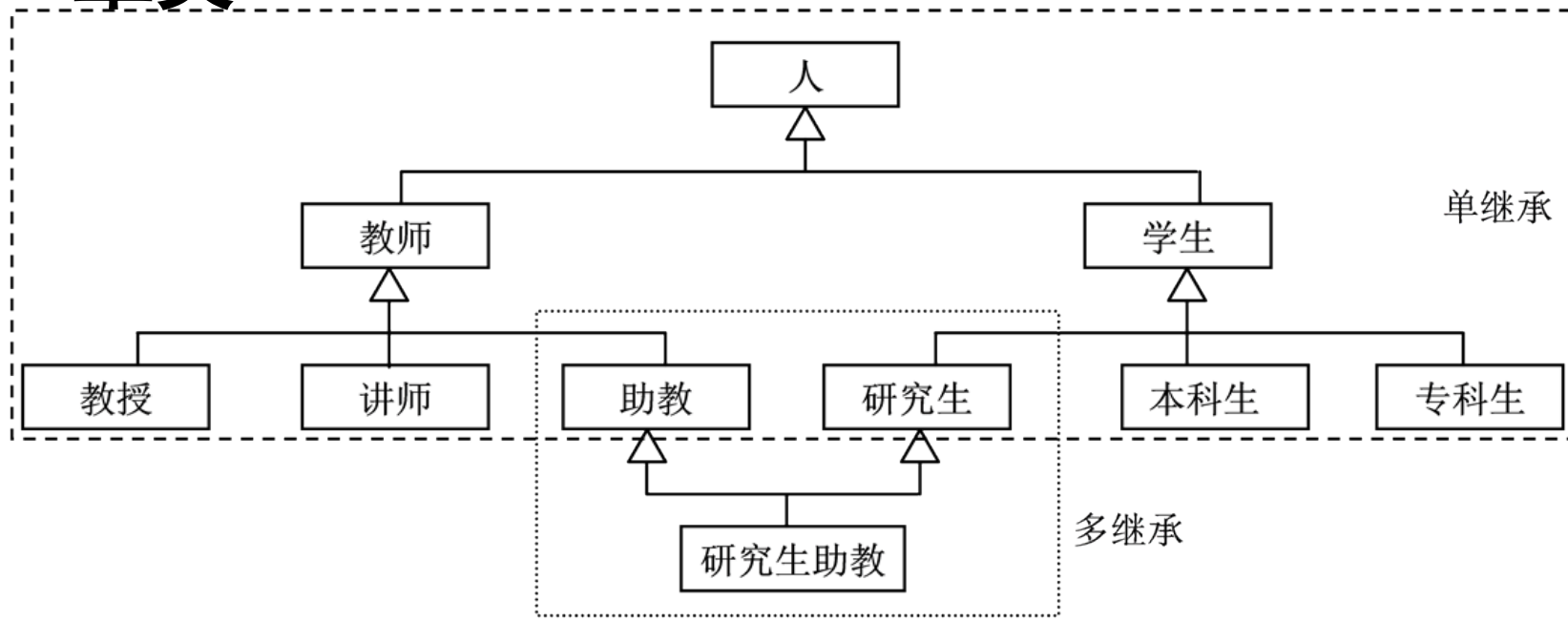
请按任意键继续. . .

# 继承与派生的概念

- **继承(Inheritance)机制**：在C++中可以利用已有的类来定义新的类，新类将拥有原有类的全部属性
- **原有的类被称为基类(base class)或父类(super class)**，student类
- **新产生的类被称为派生类(derived class)或子类**，postgradudent类

# 继承与派生的概念

- **单继承**：只有一个父类
- **多继承**：有多个父类
- **多层继承**：继承分多个层次，子类，父类，祖辈类...



# 派生类定义的语法

```
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  
};
```



# 派生类定义的语法

```
class postgraduent : public student
{
public:
    postgraduent(void);
    ~postgraduent(void);
    void setAdvisor(string str){advisor = str;}
    string getAdvisor(){return advisor;}
private:
    string advisor;
};
```

# 继承方式

- **继承方式**指定了派生类成员对于从基类继承来的成员的**访问权限**。
- 继承方式有三种：**public: 公有继承**；**private: 私有继承**；**protected: 保护继承**。

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

## 公有继承

- 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。

# 多边形

```
#include <iostream>
using namespace std;
class CPolygon {
protected:
    int width, height;
public:
    void set_values (int a, int b)
    { width=a; height=b; }
};
class CRectangle: public CPolygon {
public:
    int area ()
    { return (width * height); }
};
```

```
class CTriangle: public CPolygon {
public:
    int area ()
    { return (width * height / 2); }
};
int main () {
    CRectangle rect;
    CTriangle trgl;
    rect.set_values (4,5);
    trgl.set_values (4,5);
    cout << rect.area() << endl;
    cout << trgl.area() << endl;
    return 0;
}
```

C:\Windows\system32\cmd.exe

20

10

Press any key to continue . . .

# 说明

- 公有继承是最常用的一种继承方式，此时由于基类的私有成员不能在子类中访问，一般地将基类的私有成员设为protected类型。

## 保护继承

- 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。
- 基类的私有成员在派生类中不能被直接访问。

## 私有继承

- 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。
- 基类的私有成员在派生类中不能被直接访问。
- 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。

```
//date.h
```

```
#pragma once
```

```
#include <iostream>
```

```
using namespace std;
```

```
class date
```

```
{
```

```
public:
```

```
    date(void);
```

```
    date(int y, int m, int d);
```

```
    ~date(void);
```

```
    void displayDate();
```

```
protected:
```

```
    int year,month,day;
```

```
};
```

```
//date.cpp
```

```
#include "date.h"
```

```
date::date(void)
```

```
{
```

```
date::date(int y, int m, int d)
```

```
{
```

```
    year = y;
```

```
    month = m;
```

```
    day = d;
```

```
}
```

```
date::~date(void) {}
```

```
void date::displayDate()
```

```
{
```

```
    cout << "Date: " << year << "-"
```

```
<< month << "-" << day << endl;
```

```
}
```



```
//time.h
```

```
#pragma once
```

```
#include <iostream>
```

```
using namespace std;
```

```
class time
```

```
{
```

```
public:
```

```
    time(void);
```

```
    time(int h, int m, int s);
```

```
    ~time(void);
```

```
    void displayTime();
```

```
protected:
```

```
    int hour,minute,second;
```

```
};
```

```
//time.cpp
```

```
#include "time.h"
```

```
time::time(void) {}
```

```
time::time(int h, int m, int s)
```

```
{
```

```
    hour = h;
```

```
    minute = m;
```

```
    second = s;
```

```
}
```

```
time::~~time(void) {}
```

```
void time::displayTime()
```

```
{
```

```
    cout << "Time: " << hour << ":"  
<< minute << ":" << second
```

```
    << endl;
```

```
}
```

## 日期时间类



```
//DateTime.h
```

```
#pragma once
```

```
#include "date.h"
```

```
#include "time.h"
```

```
class dateTime : public date, public time
```

```
{
```

```
public:
```

```
    dateTime(void);
```

```
    dateTime(int y, int mon, int d, int h, int m, int s) :
```

```
        date(y,mon,d), time(h,m,s){}
```

```
    ~dateTime(void);
```

```
    void displayDateAndTime();
```

```
};
```

# 日期时间类

```
//DateTime.cpp
```

```
#include "dateTime.h"
```

```
dateTime::dateTime(void) {}
```

```
dateTime::~~dateTime(void) {}
```

```
void dateTime::displayDateAndTime()
```

```
{
```

```
    cout << "DateAndTime: ";
```

```
    cout << year << "-" << month << "-" << day
```

```
    << " ";
```

```
    cout << hour << ":" << minute << ":" <<
```

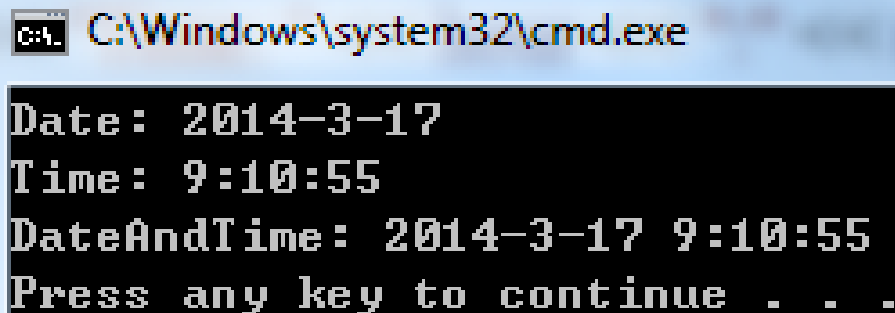
```
    second;
```

```
    cout << endl;
```

```
}
```

# 日期时间类

```
//main.cpp
#include "dateTime.h"
void main()
{
    dateTime DT(2014,3,17,9,10,55);
    DT.displayDate();
    DT.displayTime();
    DT.displayDateAndTime();
}
```



C:\Windows\system32\cmd.exe

```
Date: 2014-3-17
Time: 9:10:55
DateAndTime: 2014-3-17 9:10:55
Press any key to continue . . .
```

# 派生类的构造函数

子类不能继承父类的构造函数

如何在派生类  
中完成基类成  
员的初始化？

# 女儿和儿子

```
#include <iostream>
using namespace std;
class mother {
public:
    mother ()
    { cout << "mother\n"; }
};
```

```
class daughter : public mother {
public:
    daughter ()
    { cout << "daughter\n"; }
};
```

```
class son : public mother {
public:
    son ()
    { cout << "son\n"; }
};
int main ()
{
```

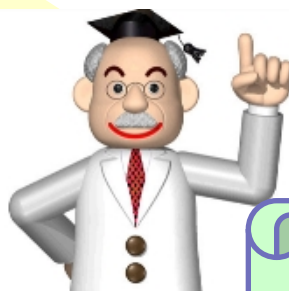
C:\Windows\system32\cmd.exe

```
mother
daughter
mother
son
Press any key to continue . . .
```

# 派生类的构造函数

子类不能继承父类的构造函数

如何在派生类  
中完成基类成  
员的初始化？



在执行派生类的构造函数时，调用基类的构造函数

```
//postgraduent.h
```

```
#pragma once
```

```
#include "student.h"
```

```
class postgraduent : public student
```

```
{
```

```
public:
```

```
    postgraduent(void);
```

```
    postgraduent(int n, string str1, char c, string str2) :
```

```
    student(n,str1,c), advisor(str2){}
```

```
    ~postgraduent(void);
```

```
    void setAdvisor(string str){advisor = str;}
```

```
    string getAdvisor(){return advisor;}
```

```
private:
```

```
    string advisor;
```

```
}
```



# 研究生类的构造函数

```
//postgraduent.cpp
```

```
#include "postgraduent.h"
```

```
postgraduent::postgraduent(void){  
}
```

```
postgraduent  
{
```

```
//main.cpp
```

```
#include "postgraduent.h"
```

```
void main()
```

```
{
```

```
    postgraduent hui(29,"Hui Wang", 'M',"Prof. Su");
```

```
    hui.display();
```

```
    cout << "Advisor: " << hui.getAdvisor() << endl;
```

```
}
```



C:\Windows\system32\cmd.exe

29 Hui Wang M

Advisor: Prof. Su

Press any key to continue . . .

## 派生类的构造函数

注意：这里的  
参数是实参而  
不是形参

派生类构造函数的一般形式：

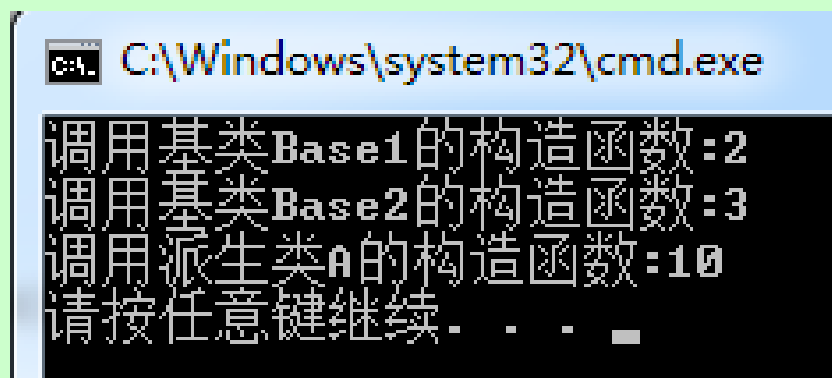
```
派生类构造函数名(总参数表)(: 基类构造函数名(参数表))  
{  
    派生类中新增数据成员初始化  
}
```

## 派生类的构造函数

```
#include <iostream>
using namespace std;
class mother {
public:
    mother ()
    { cout << "mother\n"; }
};
class daughter : public
mother {
public:
    daughter():mother ()
    { cout << "daughter\n"; }
};
```

```
class son : public mother {
public:
    son():mother ()
    { cout << "son\n"; }
};
int main ()
{
    daughter cynthia;
    son daniel;
    return 0;
}
```

```
#include<iostream>
using namespace std;
class Base1
{ public :
    Base1( int i ) { cout << "调用基类Base1的构造函数:" << i << endl ; }
};
class Base2
{ public:
    Base2( int j ) { cout << "调用基类Base2的构造函数:" << j << endl ; }
};
class A : public Base1, public Base2
{ public :
    A( int a, int b, int c, int d ) : Base1(b), Base2(c), b1(a), b2(d)
    { cout << "调用派生类A的构造函数:" << a+b+c+d << endl; }
private :
    int b1 ;
    int b2 ;
};
int main()
{ A obj( 1, 2, 3, 4 ); }
```



```
C:\Windows\system32\cmd.exe
调用基类Base1的构造函数:2
调用基类Base2的构造函数:3
调用派生类A的构造函数:10
请按任意键继续. . .
```

```
#include<iostream>
```

```
using namespace std;
```

```
class Base1
```

```
{ public
```

```
    E
```

```
};
```

```
class
```

```
{ public
```

```
    E
```

```
};
```

```
class A public Base1, public Base2
```

```
{ public :
```

```
    A( int a, int b, int c, int d ) : Base2(c), Base1(b) b1(a), b2(d)
```

```
    { cout << "调用派生类A的构造函数:" << a+b+c+d << endl; }
```

```
private :
```

```
    int b1 ;
```

```
    int b2 ;
```

```
};
```

```
int main()
```

```
{ A obj( 1, 2, 3, 4 ); }
```

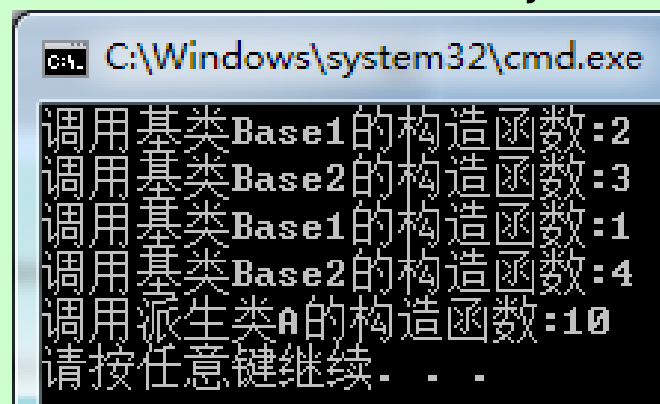
**多继承中派生类中基类构造函数的调用顺序由派生类声明中基类出现的顺序决定。**

```
C:\Windows\system32\cmd.exe
```

```
调用基类Base1的构造函数:2
调用基类Base2的构造函数:3
调用派生类A的构造函数:10
请按任意键继续. . .
```

```
#include<iostream>
using namespace std;
class Base1
{ public :
    Base1( int i ) { cout << "调用基类Base1的构造函数:" << i << endl ; }
};
class Base2
{ public:
    Base2( int j ) { cout << "调用基类Base2的构造函数:" << j << endl ; }
};
class A : public Base1, public Base2
{ public :
    A( int a, int b, int c, int d ) : Base2(c), Base1(b), b1(a), b2(d)
    { cout << "调用派生类A的构造函数:" << a+b+c+d << endl; }
private :
    Base1 b1 ;
    Base2 b2 ;
};
int main()
{ A obj( 1, 2, 3, 4 ); }
```

子对象



```
C:\Windows\system32\cmd.exe
调用基类Base1的构造函数:2
调用基类Base2的构造函数:3
调用基类Base1的构造函数:1
调用基类Base2的构造函数:4
调用派生类A的构造函数:10
请按任意键继续. . .
```

# 派生类的构造函数

派生类构造函数的一般形式：

```
派生类构造函数名(总参数表)： 基类构造函数名(参数表 ①)  
(, 子对象名(参数表)  
{ ②  
    派生类中新增数据成员初始化  
} ③
```

## 派生类的构造函数

析构函数的调用顺序正好与构造函数相反。



## 小结

- 继承与派生的概念
- 派生类成员的访问属性
- 派生类的构造函数和析构函数