面向对象方法与C++程序设计

第4章　　继承

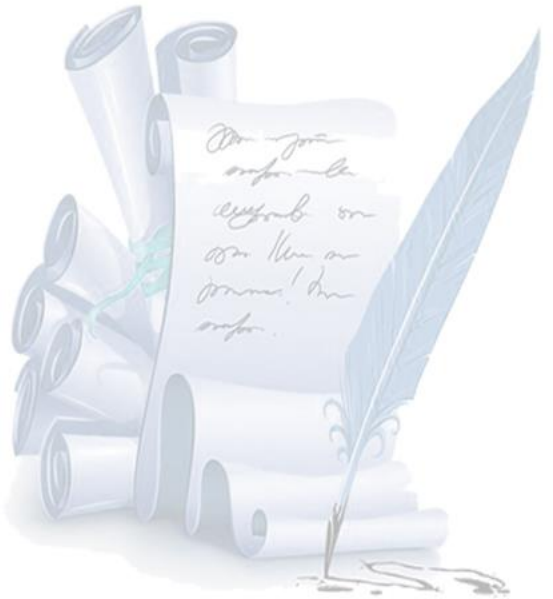
大连理工大学
主讲人-赵小薇

派生类的构造



派生类

从基类继承的数据	⇐	由基类构造函数进行初始化
新增数据	⇐	由派生类构造函数进行初始化



派生类构造函数的定义方式



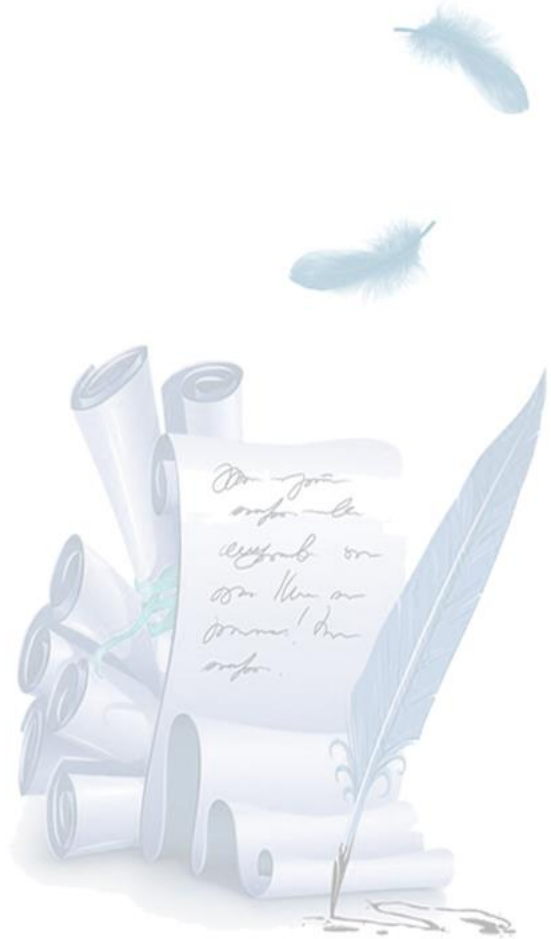
派生类名（参数总表）：基类名（基类构造函数参数表）

```
{
    派生类成员初始化;
}
```

例如

Student(int number, char name[],char sex):Person(name,sex)

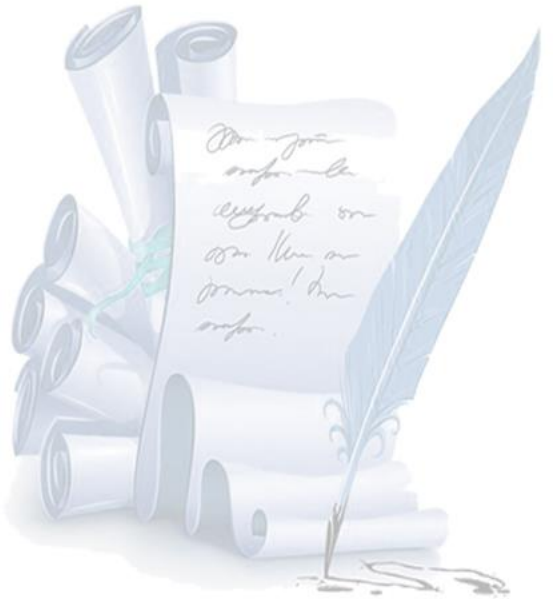
{……}



举例

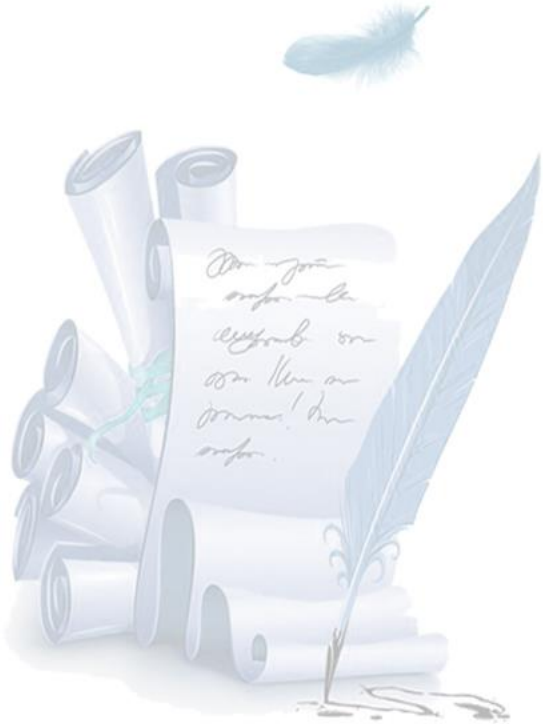


```
class Person {
protected:
        char name[10];                  // 姓名
        char sex;                       // 性别
public:
        // 基类构造函数
        Person(char name[],char sex):sex(sex){
                cout<<"基类构造函数运行"<<endl;
                strcpy(this->name,name);
        }
};
```





```
class Student : public Person{
private:
        int number;                                    // 学号
public:
        // 派生类构造函数
        Student(int number, char name[],char sex):
          Person(name,sex){
                cout<<"派生类构造函数运行"<<endl;
              this->number = number;
        }
        void print(){   ...}                          //输出学生信息
};
```



- 将派生类的构造函数改成下面的形式，更简洁。

```
Student(int number, char name[],char sex):
    Person(name,sex),number(number){    }
```

- 可不可以把构造函数中number和Person初始化的声明顺序交换呢？交换后改成下面的形式：

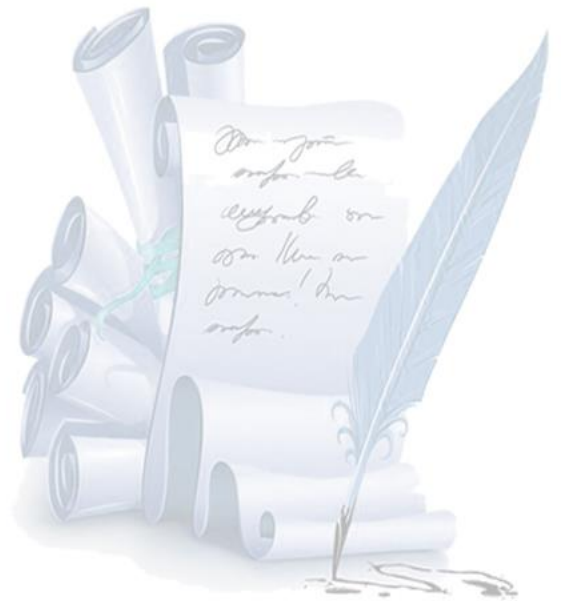
```
Student(int number, char name[],char sex):
    number(number),Person(name,sex){    }
```

交换后是否会影响初始化的顺序？派生类构造函数初始化的顺序是怎样的？

组合单继承的构造函数举例

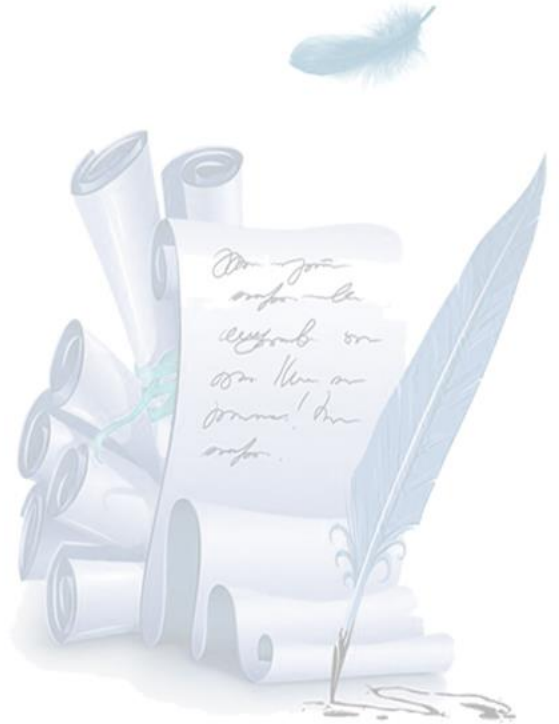


```
class Person {  ......  };
class School {
private:
        char name[10];                          // 校名
        char city[10];                          // 所在城市
public:
        School(char name[],char city[]){    // 构造函数
                strcpy(this->name,name);
                strcpy(this->city,city);
        }
        void print(){                           // 输出学校信息
                cout<<"school name:"<<name<<endl;
                cout<<"city:"<<city<<endl;     }
    };
```





```
class Student : public Person{
private:
    int number;                                    // 学号
    School school;                          // 学校（内嵌对象）
public:
    Student(int number,char name[],char sex,char s_name[],char city[])
          :Person(name,sex),school(s_name,city),number(number){ }
    void print(){
      cout<<"Student ID:"<< number<<endl;
      cout<<"name:"<< name<<endl;
      cout<<"sex:"<< sex<<endl;
      school.print();                  // 调用内嵌对象的公有函数
     }
};
```

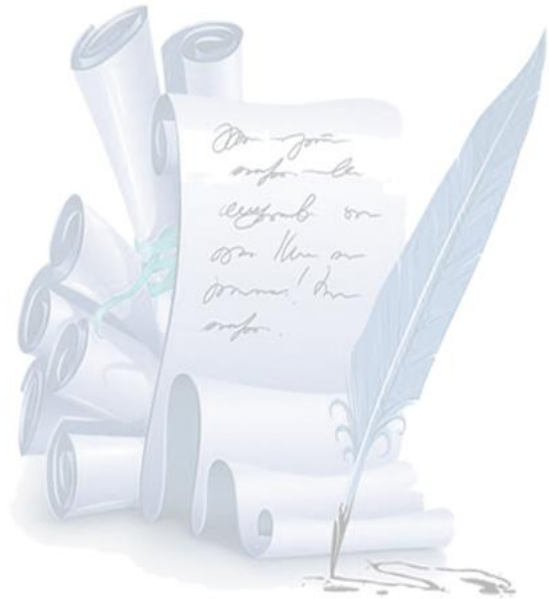




```
//主函数
int main(){
        Student s(1001,"Li Ming",'M',"DLUT","dalian");
        s.print();
        return 0;
}
```

程序运行结果如下：

```
Student ID:1001
name:Li Ming
sex:M
school name:DLUT
city:dalian
```





派生类参数

基类参数列表

内嵌对象参数列表

Student(int number,char name[],char sex,char s_name[],char city[])

:Person(name,sex),school(s_name,city),number(number)

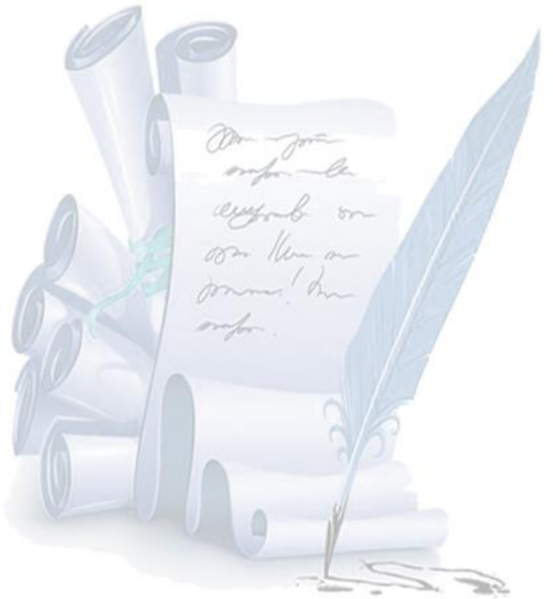
派生类构造函数执行顺序——

“**先祖先，再客人，后自己**”

初始化基类

初始化内嵌对象

初始化派生类成员



多重继承派生类构造函数

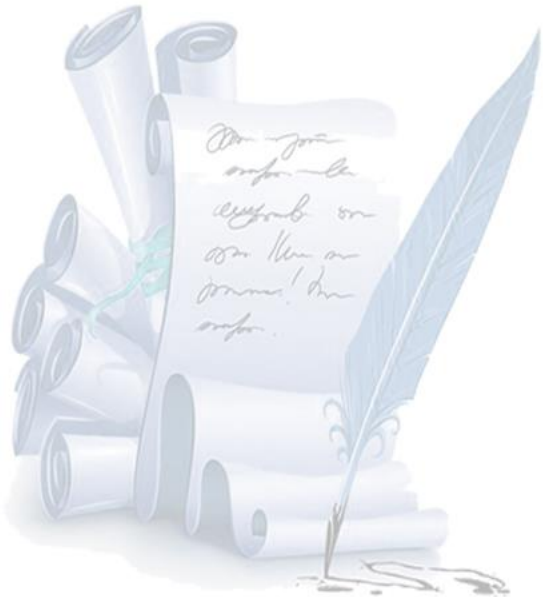


派生类名（参数总表）：基类1名（基类1构造函数参数表），基类2名（基类2构造函数参数表），……，基类n名（基类n构造函数参数表）
{派生类新增数据成员初始化；}

- 影响多重继承基类构造函数调用顺序的是派生类定义时的继承列表。Assistant类定义时的继承列表如下：

```
class Assistant: public Student, public Teacher
```

- 这决定了Assistant类构造函数的调用顺序一定是先Student，再Teacher。



派生类构造函数的执行顺序

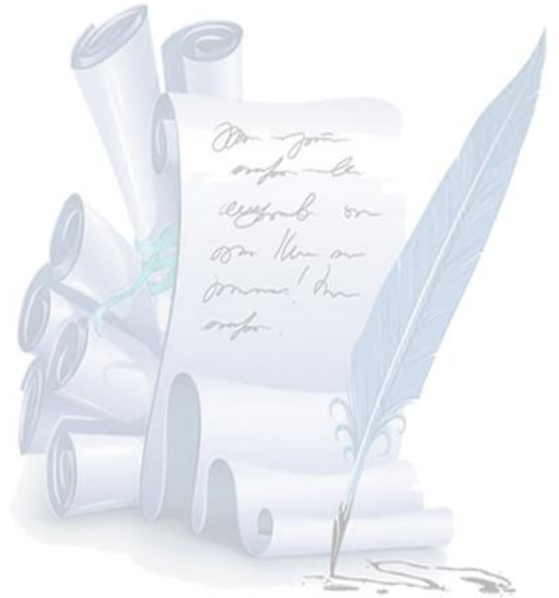


1 调用基类的构造函数，如有多个基类，则按照它们被继承的顺序依次调用。

2 调用内嵌对象的构造函数，如果有多个，则按照它们在类的数据成员声明中的先后顺序依次调用。

3 执行派生类的构造函数体中的内容。

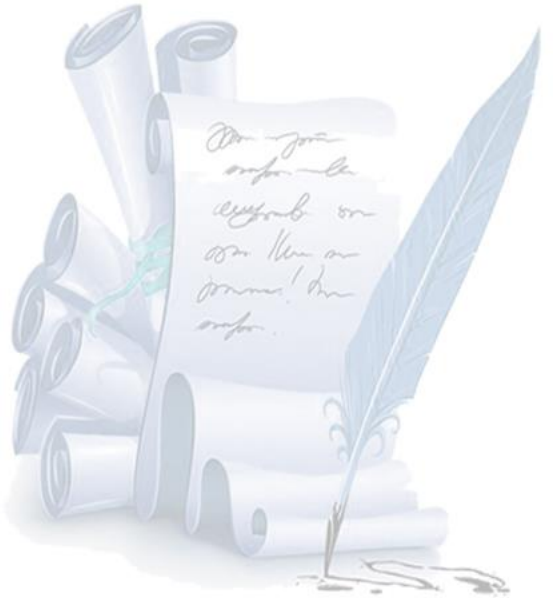
- 如果派生类的构造函数没有显示声明其基类和其内嵌对象的构造方式，那么系统按照“默认”方式对它们进行初始化，也就是调用它们的默认构造函数，如果基类或者内嵌类不具有这样的构造函数，那么就会出现编译错误。



派生类的析构函数



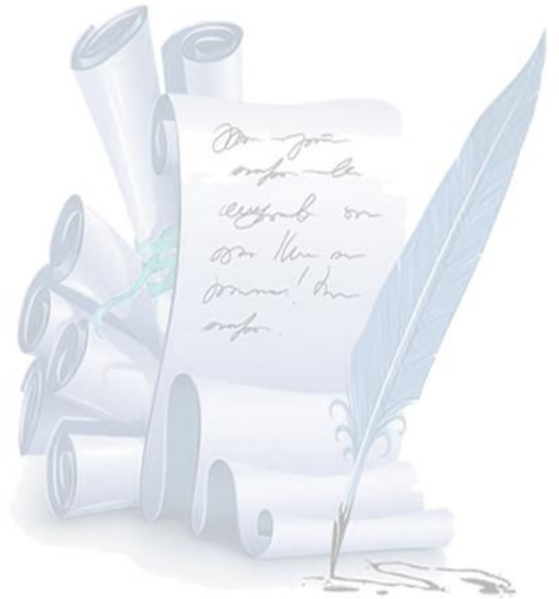
- 在析构的的时候，派生类的析构函数依次执行了自身的析构函数、内嵌对象的析构函数和基类的析构函数，其执行的顺序与其构造函数执行的顺序正好严格相反。



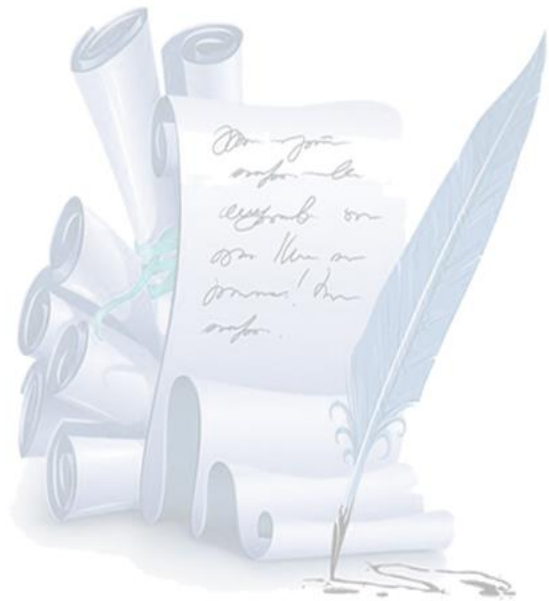
举例



```
class Student{
    ……
        // 析构函数
        ~Student(){cout<<"Student类析构"<<endl;}
    ……
};
class Teacher{
     ……
        // 析构函数
        ~Teacher(){ cout<<"Teacher类析构("<<t_name<<")"<<endl;}
    ……
};
```



```
class Assistant: public Student, public Teacher{
private:
        float salary;                   // 薪金
        Teacher teacher;                // 助课教师
public:
        Assistant(int number,int age, char name[],char
sex,char title[],float salary,int t_age,char t_name[],char
t_title[]):Student(number,name,sex),Teacher(age,name,title),t
eacher(t_age,t_name,t_title),salary(salary){
                cout<<"Assistant类构造"<<endl;
        }
        ~Assistant(){cout<<"Assistant类析构"<<endl;}
        void print(){ … } // 输出助教信息
};
```





```
int main(){
        Assistant a(1001,18,"Li Ming",'M',"assistant",987.65f,45,"Wang Feng","Professor");
        return 0;
}
```

程序运行的结果如下：

```
Student类构造
Teacher类构造(Li Ming)
Teacher类构造(Wang Feng)
Assistant类构造
Assistant类析构
Teacher类析构(Wang Feng)
Teacher类析构(Li Ming)
Student类析构
```