类与对象(上)

[本节目标]

- 1.面向过程和面向对象初步认识
- 2.类的引入
- 3.类的定义
- 4.类的作用域
- 5.类的实例化
- 6.类的访问限定符及封装
- 7.类的对象大小的计算
- 8.类成员函数的this指针

1.面向过程和面向对象初步认识

> C语言是**面向过程**的，**关注**的是**过程**，分析出求解问题的步骤，通过函数调用逐步解决问题。
>
> C++是**基于面向对象**的，**关注**的是**对象**，将一件事情拆分成不同的对象，靠对象之间的交互完成。

2.类的引入

> **C语言中，结构体中只能定义变量，在C++中，结构体内不仅可以定义变量，也可以定义函数。**

```
struct Student
{
    void SetStudentInfo(const char* name, const char* gender, int age)
    {
        strcpy(_name, name);
        strcpy(_gender, gender);
        _age = age;
    }

    void PrintStudentInfo()
    {
        cout<<_name<<" "<<_gender<<" "<<_age<<endl;
    }

    char _name[20];
    char _gender[3];
    int _age;
};

```

初始化

打印

```
int main()
{
    Student s;
    s.SetStudentInfo("Peter", "男", 18);
    return 0;
}
```

上面结构体的定义，**在C++中更喜欢用class来代替**

3.类的定义

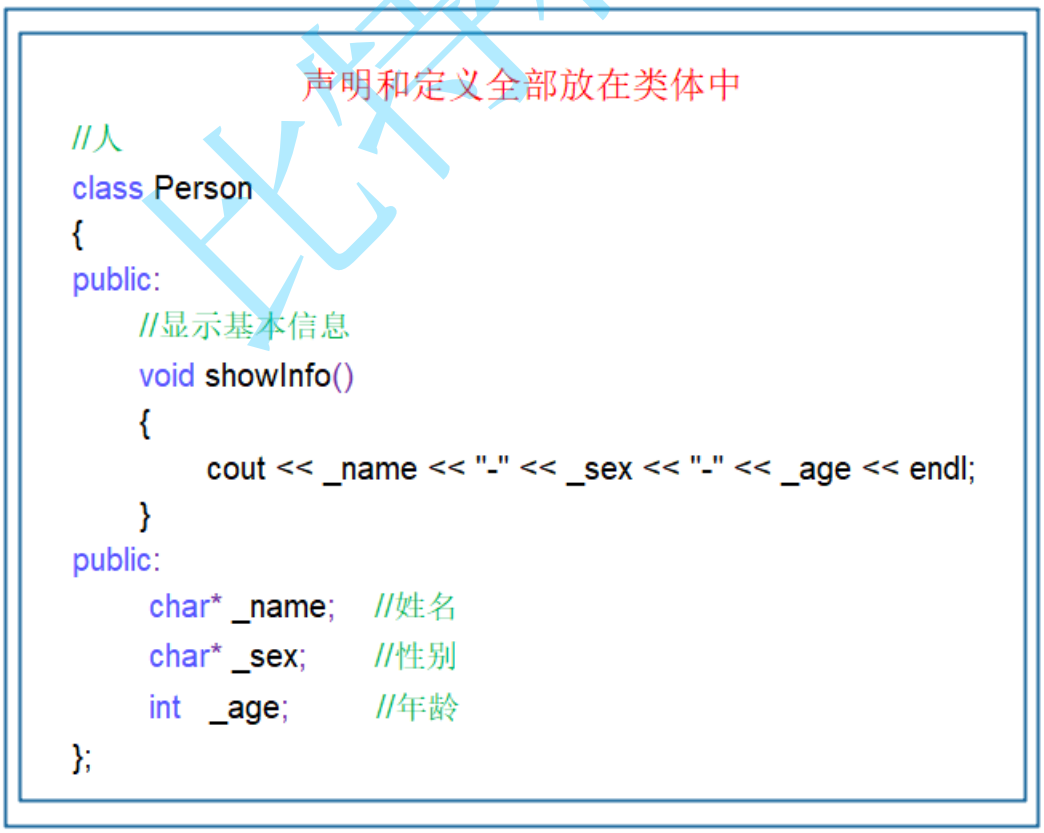
```
class className
{
    // 类体：由成员函数和成员变量组成

}; // 一定要注意后面的分号
```

class为定义类的**关键字，ClassName为类的名字，{}中为类的主体，注意**类定义结束时后面**分号。**

类中的元素称为**类的成员**：类中的**数据**称为**类的属性**或者**成员变量**; 类中的**函数**称为**类的方法**或者**成员函数。**

类的两种定义方式：

1. 声明和定义全部放在类体中，需要注意：成员函数如果**在类中定义**，编译器可能会将其当成**内联函数**处理。



2. 声明放在.h文件中，类的定义放在.cpp文件中

声明放在类的头文件person.h中

```
//人
class Person
{
public:
	//显示基本信息
	void showInfo();
public:
	char* _name;	//姓名
	char* _sex;		//性别
	int  _age;		//年龄
};
```

定义放在类的实现文件person.cpp中

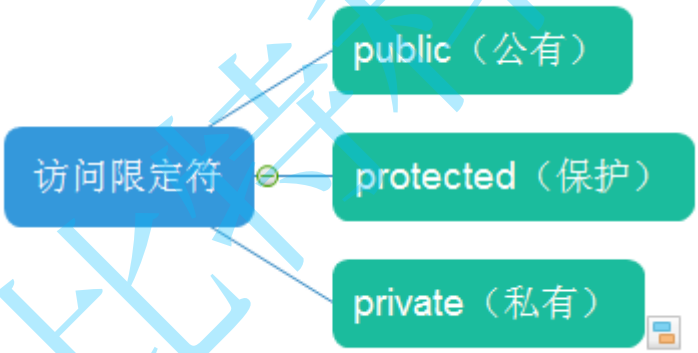
```
#include "person.h"

//显示基本信息，实现：输出 名字 、性别 、年龄
void Person::showInfo()
{
	cout << _name << "-" << _sex << "-" << _age << endl;
}
```

一般情况下，更期望采用第二种方式。

4.类的访问限定符及封装

4.1 访问限定符

C++实现封装的方式：**用类将对象的属性与方法结合在一块，让对象更加完善，通过访问权限选择性的将其接口提供给外部的用户使用。**



【访问限定符说明】

1. public修饰的成员在类外可以直接被访问
2. protected和private修饰的成员在类外不能直接被访问(此处protected和private是类似的)
3. 访问权限**作用域从该访问限定符出现的位置开始直到下一个访问限定符出现时为止**
4. class的默认访问权限为private，struct为public(因为struct要兼容C)

注意：访问限定符只在编译时有用，当数据映射到内存后，没有任何访问限定符上的区别

【面试题】

问题：C++中struct和class的区别是什么?

解答：C++需要兼容C语言，所以C++中struct可以当成结构体去使用。另外C++中struct还可以用来定义类。

和class是定义类是一样的，区别是struct的成员默认访问方式是public，class是struct的成员默认访问方式是private。

4.2 封装

> 【面试题】面向对象的三大特性：**封装、继承、多态。**
>
> 在类和对象阶段，我们只研究类的封装特性，那什么是封装呢？
>
> **封装：将数据和操作数据的方法进行有机结合，隐藏对象的属性和实现细节，仅对外公开接口来和对象进行交互。**
>
> **封装本质上是一种管理**：我们如何管理兵马俑呢？比如如果什么都不管，兵马俑就被随意破坏了。那么我们首先建了一座房子把兵马俑给**封装**起来。但是我们目的全封装起来，不让别人看。所以我们**开放了售票通道**，可以买票突破封装在合理的监管机制下进去参观。类也是一样，我们使用类数据和方法都封装到一下。不想给别人看到的，我们使用protected/private把成员**封装**起来。**开放**一些共有的成员函数对成员合理的访问。所以封装本质是一种管理。

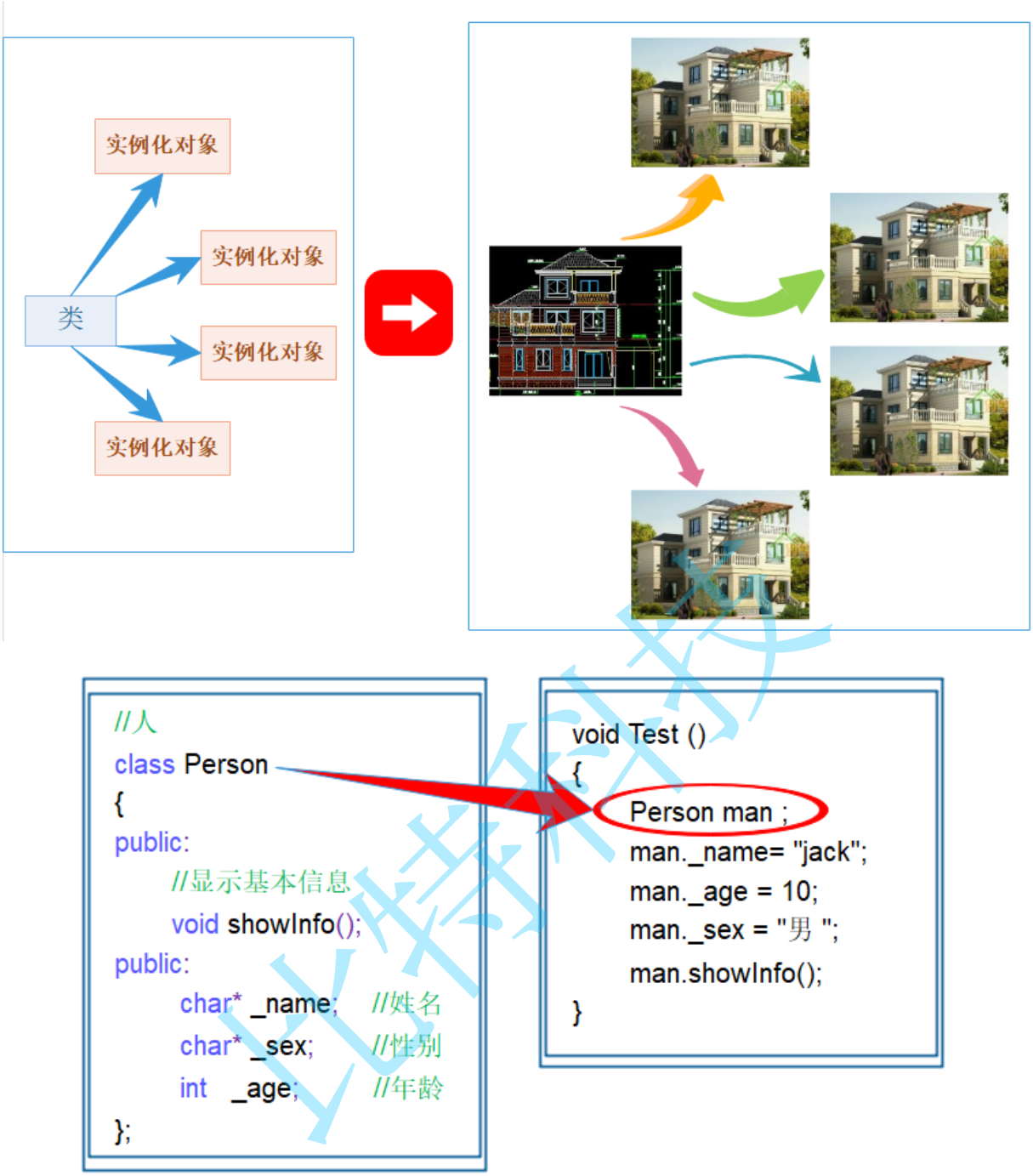
5.类的作用域

> **类定义了一个新的作用域**，类的所有成员都在类的作用域中。在类体外定义成员，需要使用 :: 作用域解析符指明成员属于哪个类域。

```
class Person
{
public:
  void PrintPersonInfo();
private:
  char _name[20];
  char _gender[3];
  int  _age;
};

// 这里需要指定PrintPersonInfo是属于Person这个类域
void Person::PrintPersonInfo()
{
     cout<<_name<<" "_gender<<" "<<_age<<endl;
}
```

6.类的实例化

> **用类类型创建对象的过程，称为类的实例化**
>
> 1. **类只是**一个**模型**一样的东西，限定了类有哪些成员，定义出一个类**并没有分配实际的内存空间**来存储它
> 2. 一个类可以实例化出多个对象，**实例化出的对象 占用实际的物理空间，存储类成员变量**
> 3. 做个比方。**类实例化出对象就像现实中使用建筑设计图建造出房子，类就像是设计图**，只设计出需要什么东西，但是并没有实体的建筑存在，同样类也只是一个设计，实例化出的对象才能实际存储数据，占用物理空间



7.类对象模型

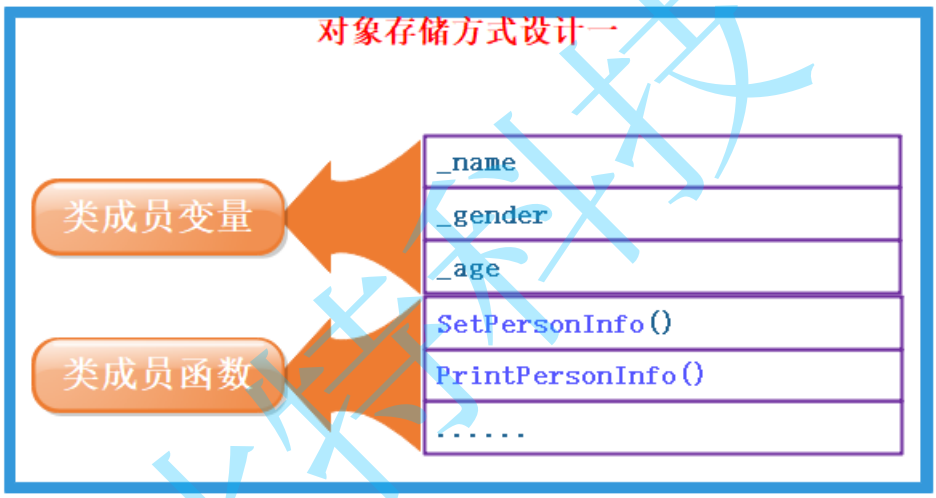
7.1 如何计算类对象的大小

```cpp
class A
{
public:
  void PrintA
  {
      cout<<_a<<endl;
  }
private:
  char _a;
};
```

问题：类中既可以有成员变量，又可以有成员函数，那么一个类的对象中包含了什么？如何计算一个类的大小？

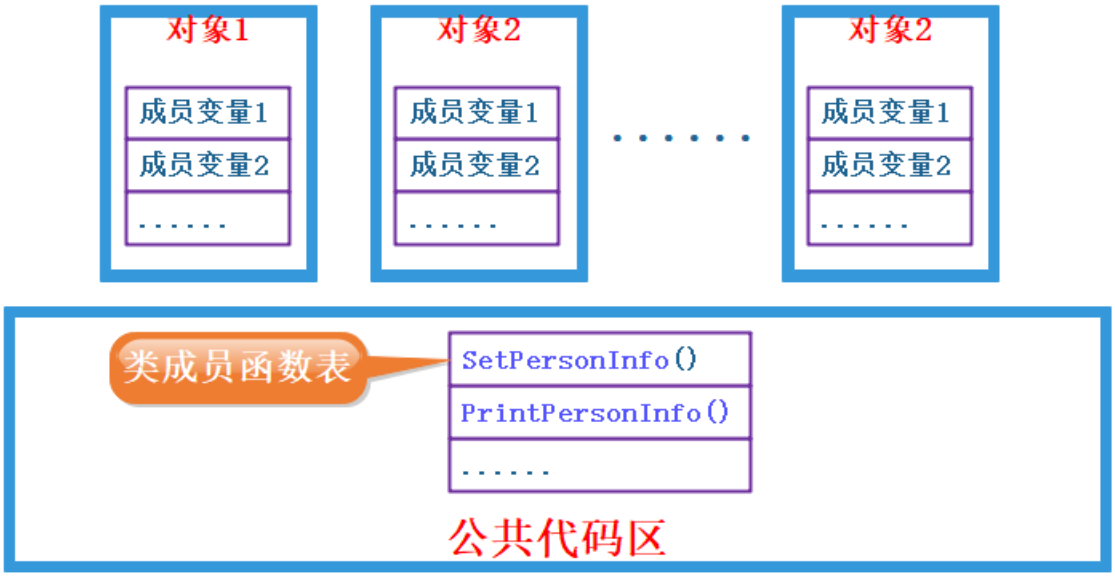
7.2 类对象的存储方式猜测

- **对象中包含类的各个成员**



缺陷：每个对象中成员变量是不同的，但是调用同一份函数，如果按照此种方式存储，当一个类创建多个对象时，**每个对象中都会保存一份代码，相同代码保存多次，浪费空间。那么如何解决呢?**

- **只保存成员变量，成员函数存放在公共的代码段**



问题：对于上述两种存储方式，那计算机到底是按照那种方式来存储的?

我们再通过对下面的不同对象分别获取大小来分析看下

```
// 类中既有成员变量，又有成员函数
class A1 {
public:
    void f1(){}
private:
    int _a;
};

// 类中仅有成员函数
class A2 {
public:
   void f2() {}
};

// 类中什么都没有---空类
class A3
{};
```

sizeof(A1) : _____ sizeof(A2) : _____ sizeof(A3) : _____

结论：一个类的大小，实际就是该类中"成员变量"之和，当然也要进行内存对齐，注意空类的大小，空类比较特殊，编译器给了空类一个字节来唯一标识这个类。

7.3 结构体内存对齐规则

1. 第一个成员在与结构体偏移量为0的地址处。
2. 其他成员变量要对齐到某个数字（对齐数）的整数倍的地址处。

注意：对齐数 = 编译器默认的一个对齐数 与 该成员大小的较小值。

VS中默认的对齐数为8

3. 结构体总大小为：最大对齐数（所有变量类型最大者与默认对齐参数取最小）的整数倍。
4. 如果嵌套了结构体的情况，嵌套的结构体对齐到自己的最大对齐数的整数倍处，结构体的整体大小就是所有最大对齐数（含嵌套结构体的对齐数）的整数倍。

【面试题】

1. 结构体怎么对齐？ 为什么要进行内存对齐
2. 如何让结构体按照指定的对齐参数进行对齐
3. 如何知道结构体中某个成员相对于结构体起始位置的偏移量
4. 什么是大小端？如何测试某台机器是大端还是小端，有没有遇到过要考虑大小端的场景

8.this指针

8.1 this指针的引出

我们先来定义一个日期类Date

```
class Date
{
public :
    void Display ()
    {
        cout <<_year<< "-" <<_month << "-"<< _day <<endl;
    }

    void SetDate(int year , int month , int day)
    {
        _year = year;
        _month = month;
        _day = day;
    }
private :
    int _year ; // 年
    int _month ; // 月
    int _day ; // 日
};

int main()
{
    Date d1, d2;
    d1.SetDate(2018,5,1);
    d2.SetDate(2018,7,1);
    d1.Display();
    d2.Display();
    return 0;
}
```

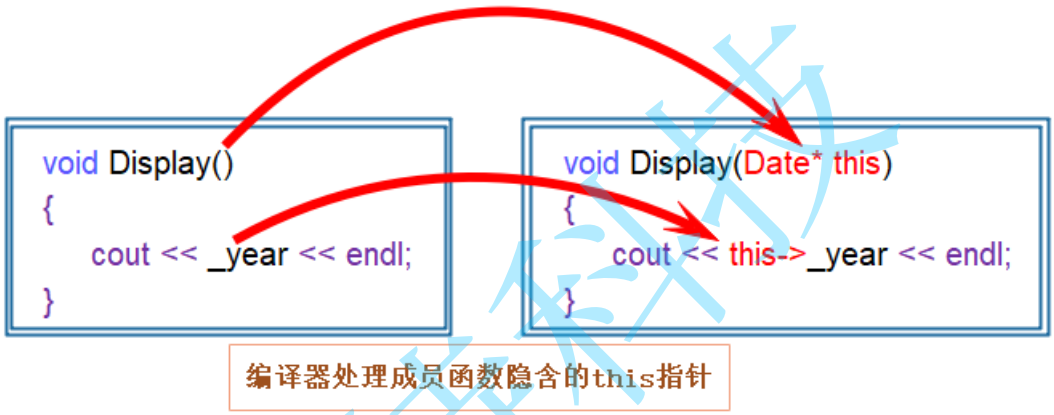
对于上述类，有这样的一个问题：

Date类中有SetDate与Display两个成员函数，函数体中没有关于不同对象的区分，那当s1调用SetDate函数时，该函数是如何知道应该设置s1对象，而不是设置s2对象呢？

C++中通过引入this指针解决该问题，即：**C++编译器给每个“非静态的成员函数”增加了一个隐藏的指针参数，让该指针指向当前对象(函数运行时调用该函数的对象)，在函数体中所有成员变量的操作，都是通过该指针去访问。只不过所有的操作对用户是透明的，即用户不需要来传递，编译器自动完成。**

8.2 this指针的特性

1. this指针的类型：类类型* const
2. 只能在“成员函数”的内部使用
3. **this指针本质上其实是一个成员函数的形参**，是对象调用成员函数时，将对象地址作为实参传递给this形参。所以**对象中不存储this指针。**
4. **this指针是成员函数第一个隐含的指针形参，一般情况由编译器通过ecx寄存器自动传递，不需要用户传递**



【面试题】

1. this指针存在哪里?
2. this指针可以为空吗?

```
// 1.下面程序能编译通过吗?
// 2.下面程序会崩溃吗？在哪里崩溃
class A
{
public:
    void PrintA()
    {
        cout<<_a<<endl;
    }

    void Show()
    {
        cout<<"Show()"<<endl;
    }
private:
    int _a;
};

```

```
int main()
{
    Date* p = NULL;
    p->PrintA();
    p->Show();
}
```