華中科技大學

第4章　C++的类——基础

许向阳

xuxy@hust.edu.cn

学习内容

華中科技大學

4.1 类的声明及定义

4.2 成员访问权限

4.3 构造函数

4.4 析构函数

4.5 完整的例子

难点

思维的转变：

由习惯C语言的表达过渡到C++的表达

习惯隐含参数this

- 对象构造 与 对象空间分配 的差别
- 对象析构 与 对象空间释放 的差别
- “自动调用”的实现过程
- 以对象引用为参数的构造
- 以对象为参数 与 以对象引用为参数的差别

4.1 类的声明及定义

学生信息管理

姓名、年龄
分数、评价

两个学生
xu 、zhang

结构 ⇒ 类

```
struct Student {
    char    name[10];
    short   age;
    float   score;
    char*   remark;
};

struct Student xu;
struct Student zhang;
```

字段 ⇩ 数据成员

结构变量 ⇩ 对象

在 .cpp 文件中，可以省略 struct，写成 Student xu;

4.1 类的声明及定义

cpp文件中

类：Student

数据成员：

name、age、…

```
struct Student {
    char   name[10];
    short  age;
    float  score;
    char*  remark;
};
```

对象

```
Student xu, zhang;
```

对象指针

```
Student *p = &xu;
```

对象数组

```
Student sarray[100];
```

```
xu.age = 21;
p->age = 21;
sarray[0].age =21;
```

4.1 类的声明及定义

任务1：初始化 xu 的信息

“xuxy”,21, 90, “very …”

任务2：显示 xu 评价

任务3：把 zhang 的信息复制到 xu中

任务4：释放 xu中remark指向的空间

```
void init_xu()
{   strcpy(xu.name, “xuxy”);
    xu.age =21;
    ……
}
```

Q ：如何定义完成这些任务的函数？

```
void init(Student *s, char *name, int age,
                  float score, char *remark);
void display_remark(Student *s);
void assign(Student *dst, Student *src);
void free_remark(Student *s);
```

4.1 类的声明及定义

任务中的**要素**

有动作：初始化、显示、复制、释放空间

有处理对象：xu、zhang

动作 → 函数名	动宾结构
对象 → 参数	更强调动作

要我学习！	VS	我要学习！
主动	VS	被动
动宾结构	VS	主谓结构
以动作为中心	VS	以对象为中心

从 C 到 C++ 的思维转变

4.1 类的声明及定义

任务1：初始化 xu 的信息
“xuxy”,21, 90, “very …”
任务2：显示 xu 评价
任务3：把 zhang 的信息复制到 xu中
任务4：释放 xu中remark指向的空间

从 C 到 C++ 的思维转变

main函数：
任务1：xu, 初始化（你的）信息
“xuxy”,21,90,“very …”
任务2：xu, 显示（你的）评价
任务3：xu, 复制为 zhang的信息
任务4：xu, 释放remark指向的空间

main函数：消息的发出者
xu ：消息的接收者
初始化、显示、… ：消息
值 … ：参数

4.1 类的声明及定义


```
struct Student {
    char   name[10];
    short  age;
    float  score;
    char*  remark;
};
```

```
init(&xu, ……);          初始化 xu …
display_remark(&xu);
assign(&xu, &zhang);
free_remark(&xu);
```

两种表示有何差别？

```
struct Student {
    char   name[10];
    short  age;
    float  score;
    char*  remark;
    void init(char *name, int age, float score, char *remark);
    void display_remark();
    void assign(Student *src);
    void free_remark();
};
```

```
xu.init(……);          xu，初始化…
xu.display_remark();
xu.assign(&zhang);
xu.free_remark( );
```

4.1 类的声明及定义


```
init(&xu, …,21,90,…);
display_remark(&xu);
assign(&xu,&zhang);
free_remark(&xu);

xu.init(…, 21,90,…);
xu. display_remark( );
xu. assign(&zhang);
xu. free_remark();
p->init(……);
sarray[0].init(……);
```

C：更强调操作

init，初始化

assign，复制

C++：

强调对象

听到指令者

消息的接收者

xu，可以初始化、显示评价、复制别人的信息、释放空间

以对象为中心，操作是对象的行为

C：更强调操作

函数参数可以有多种排列顺序，体现不出参数的重要程度。

```
void init(Student *s, char *name, int age, float score, char *remark);
void init(char *name, int age, float score, char *remark, Student *s, );
```

按理，s 比其他参数更富有信息量，应该在更突出的位置！

在 C 程序中，没有办法突出 s 的核心地位！

4.1 类的声明及定义

C++：更强调对象（消息的接收者，即听到指令的人）

操作 可以看成是对象能够的任务，对象的行为。

xu， 初始化信息……；

zhang, 初始化信息……

4.1 类的声明及定义

日常生活中表达方式的差异

XX号，YY栋，喻园小区，华中科技大学，洪山区，武汉市

武汉市，洪山区，华中科技大学，喻园小区，YY栋，XX号

2025年10月1日

1号/10月/2025年

Xu Xiangyang

Xiangyang Xu

我和你

You and I

核心要素不变，但表达的顺序不同

强调的重点不同， 动作 or 完成动作的人

函数的实现

```
void  init(Student *s,const char* name, short age,
                float score, const char* remark)
{  int len;
      strcpy(s->name, name);
      s->age = age;
      s->score = score;
      len = strlen(remark);
      s->remark = (char *)malloc(len+1);
      strcpy(s->remark, remark);
}
```

- 不应该有 s->remark = remark; why?
- 为 s->remark 申请新空间，其之前指向的空间怎么办?
- 对于数组 s->name，不能直接赋值 s->name=name;
  有语法错误！不可指定数组类型 char[10]

4.1 类的声明及定义

C++ 的写法

```
struct Student {
    char  name[10];          short age;
    float score;             char* remark;
    void init(const char* name1, short age1, float score1, const
char* remark1)
   {   int len;
       strcpy(name, name1);
       age = age1;              score = score1;
       len = strlen(remark1);
       remark = (char *)malloc(len + 1);
       strcpy(remark, remark1);
   }
};
```

Q: C中的init函数,通过指针s, s->name来访问结构的字段。

C++ 中，如何知道 name 是访问哪个对象的name?

```
xu.init("xuxy",20, 100,  "excellent");
zhang.init("zhangwj",22, 95,  "good");
```


```
void init(const char* name1, short age1,……)
```

Q: 参数名能不能与数据成员名 同名?

```
struct Student {
    char  name[10];          short age;
    float score;             char* remark;
    void init(const char* name, short age, float score, const char*
remark)
   {   int len;
       strcpy(this->name, name);
       this->age = age;              this->score = score;
       len = strlen(remark);
       this->remark = (char *)malloc(len + 1);
       strcpy(this->remark, remark);
   }
};
```

隐含第一参数 this : Student * const this;

成员函数在类内定义

4.1 类的声明及定义

成员函数 在类内申明
在类外定义

```
struct Student {
    char  name[10];
    short age;
    float score;
    char* remark;
    void init(const char* name, short age, float score, const char*
remark);
};
void Student::init(const char* name, short age, float score, const
char* remark)
{   int len;
    strcpy(this->name, name);
    this->age = age;                    // Student::age=age;
    this->score = score;                // (*this).score = score;
    len = strlen(remark);
    this->remark = (char *)malloc(len + 1);
    strcpy(this->remark, remark);
}
```

4.1 类的声明及定义

Tips:

```
#define _CRT_SECURE_NO_WARNINGS
        // 在 VS2019中，用 strcpy_s 代替了 strcpy.
        // 若仍然使用 strcpy, 则加上该语句，编译才不报错

#include <stdio.h>
#include <string.h>
#include <malloc.h>
```

Q: 为什么 加 const?

```
  init(const char* name, short age, float score,
const char* remark);
     xu.init(“xuxy”,…);
     xu.init(buf,……);
```

4.1 类的声明及定义

类保留字： class、struct或union可用来声明和定义类。

类的定义：

```
class 类型名{
[private:]
    私有成员声明或定义;
protected:
    保护成员声明或定义;
public:
    公有成员声明或定义;
};
```

类的实现： 类的函数成员的实现，即定义类的函数成员。

类的申明： class 类型名;

4.1 类的声明及定义

总结：

- 类 将 数据成员、函数成员封装在一起
- （非静态的）函数成员有**隐含第一参数**

  类名 * const this;

Any Questions?

4.2 成员访问权限

4.2 成员访问权限

类保留字： class、struct或union可用来声明和定义类。

类的定义：

```
class 类型名{
[private:]
    私有成员声明或定义;
protected:
    保护成员声明或定义;
public:
    公有成员声明或定义;
};
```

4.2 成员访问权限

private：私有成员

仅可被本类的函数成员访问

不能被派生类、其它类和普通函数访问

protected：受保护成员

可被本类和派生类的函数成员访问

不能被其它类函数成员和普通函数访问

public：公有成员

可被任何函数成员和普通函数访问

class定义的类缺省访问权限为private

struct和union定义的类缺省访问权限为public。

4.2 成员访问权限


```
class Student
{
public:  char  name[10];
private:
          short  age;
          float score;
public:  char  *remark;
public:
          Student(const char *name, short age, float score,
                    const char * remark);
          Student( );
          void modifyScore(float score)   //声明与定义
          {
                    Student::score=score;
          }
          void printStudentInfo( );                    // 在体外定义
};
```

Q：私有成员有哪些？

公有成员有哪些？

受保护成员？

4.2 成员访问权限

```
int main(int argc, char* argv[])
{    Student  stu1, stu2;
     strcpy_s(stu1.name,"xu" );       // name 为 public 成员
     stu1.remark = (char *)malloc(10);
     strcpy_s(stu1.remark, 10,"123456789");
     stu1.score = 99;                 // score 为 private 成员
//   error C2248: 'score' : cannot access private member declared in class
'Student'
      return 0;
}
```

strcpy_s ：有两种函数形式

```
errno_t strcpy_s(char (&strDestination)[size],
                 const char *strSource);
errno_t strcpy_s(char *strDestination,
      size_t numberOfElements,const char *strSource);
```

第一个参数差别：有长度的字符数组 vs 字符指针

4.2 成员访问权限

- private、protected和public 标识每一区间的访问权限
- private、protected和public 可以多次出现；
- 同一区间内可以有数据成员、函数成员和类型成员；
- 习惯上按类型成员、数据成员和函数成员分开；
- 成员可以任意顺序出现；
- 函数成员的实现既可放在类体外，也可在类体中；
- 若函数成员在类的定义体外实现，
  使用“类名::”指明该函数成员所属的类；
- 类的定义体花括号后要有分号作为结束标志。

4.2 成员访问权限

Q：一般情况下，数据成员应该定义成什么权限？

应该是私有的，实现信息的隐藏。

通过公有接口，访问对象的数据成员.

```
class Student
{
private:
    char  name[10];  short age;   float score;   char  *remark;
public:
    Student(const char *name, short age, float score,
                const char * remark);
    Student( );
    void modifyScore(float score) ;
    float getScore( );
    void printStudentInfo();
};
```

4.2 成员访问权限

DeepSeek：C++类定义中，使用私有化数据成员，公有化接口，有哪些好处？

坚持“**私有数据，公有接口**”的原则，是编写**健壮、灵活、易维护、易协作**的C++代码的基石。它有效地在类的作者和使用者之间建立了一道“**防火墙**”，使得软件项目能够更好地应对变化和复杂性。

4.2 成员访问权限

私有数据，公有接口

- **数据完整性：**通过验证逻辑确保对象状态始终有效。
- **实现隐藏：**改变内部实现不影响现有代码，降低耦合。
- **抽象：**用户只关心接口（做什么），不关心实现（怎么做）。
- **可控性：**可轻松添加日志、通知、延迟计算等副作用。
- **访问控制：**可精细控制每个成员的读、写权限。
- **易于调试：**数据修改路径明确，易于追踪和调试。

4.3 构造函数

Student xu： 定义对象；

xu.init(“xuxy”,20,100, “excellent”); 对象空间初始化

Q：在定义一个对象时，除了给对象分存储空间外，能否同时对空间中的内容初始化？

Deepseek：C++中，在定义对象时，就初始化(构造函数)，有什么好处？

在定义对象时进行初始化是 C++ 中的核心编程原则，它提高了代码的健壮性、安全性和可维护性，是现代 C++ 开发中的必备实践。

4.3 构造函数

在定义对象时，就初始化(构造函数)，有什么好处?

- **保证对象始终处于有效状态**：通过构造函数初始化，可以确保对象一旦被创建，其所有成员都已经被设置为有意义的值。这样可以避免出现未初始化的成员，从而防止使用未定义的值导致的错误。
- **提供必要的初始化逻辑**：有些对象的成员可能需要进行复杂的初始化，比如申请资源（内存、文件句柄等）、检查参数有效性等。构造函数可以封装这些逻辑，确保初始化正确完成。
- 避免忘记初始化：如果依赖用户先创建对象再调用一个初始化函数，可能会忘记调用。构造函数则自动调用，避免了这种错误。
- **const成员和引用成员的初始化**：const成员和引用成员必须在构造函数初始化列表中进行初始化，因为它们一旦初始化后就不能再赋值。因此，构造函数是初始化这些成员的唯一途径。

4.3 构造函数

- **继承和多态**：基类部分的初始化需要通过构造函数完成。同时，通过构造函数的调用顺序，可以正确初始化基类和成员子对象。
- **资源管理**：如果类管理资源（如动态内存、文件、锁等），构造函数可以获取资源，并通过RAII（资源获取即初始化）机制，确保资源在对象生命周期结束时被释放（通过析构函数）。这是C++中管理资源的重要 idiom。
- **简化代码**：通过构造函数，可以将初始化代码集中在一个地方，而不是分散在多个初始化函数中，使代码更清晰、更易于维护。
- **支持多种初始化方式**：通过重载构造函数，可以提供多种初始化对象的方式，增加类的灵活性。
- **异常安全**：如果构造函数中发生异常，可以通过异常机制处理，避免创建无效的对象。

4.3 构造函数

Q：空间中的内容来源有哪些？

- 无参数的构造函数
- 以简单类型为参数的构造函数
- 以对象为参数的构造函数　　复制构造
- 以临时对象为参数的构造函数（移动构造）

4.3 构造函数

- 可以有多个构造函数；　（也可以不定义构造函数）
- 函数名与类名相同；
- 多种参数，以支持重载；
- 无返回类型；
- 只能自动调用；
- 构造函数并不申请 对象本身的空间；
- 只是做空间初始化等工作。

构造函数示例

```
class Student {
  private:
    char  name[10];          short age;
    float score;             char* remark;
  public:
    Student(const char* name, short age, float score,
const char* remark);
};
Student::Student(const char* name, short age, float score, const char*
remark)
{   int len;
    strcpy_s(this->name, name);
    this->age = age;
    this->score = score;
    len = strlen(remark)+1;
    this->remark = (char *)malloc(len);
    strcpy_s(this->remark, len, remark);
}
```

构造函数示例

```
class Student {
  private:
    char  name[10];      short age;
    float score;         char* remark;
  public:
    Student(char* name, short age, float score, char* remark);
    Student(const char* name, short age, float score,
            const char* remark);
};
Student::Student(const char* name, short age, float score, const char*
remark)
{   int len;
    cout << "general construct " << endl;
    strcpy_s(this->name, name);
    this->age = age;          this->score = score;
    len = strlen(remark)+1;
    this->remark = (char *)malloc(len);
    strcpy_s(this->remark,len, remark);
}
```

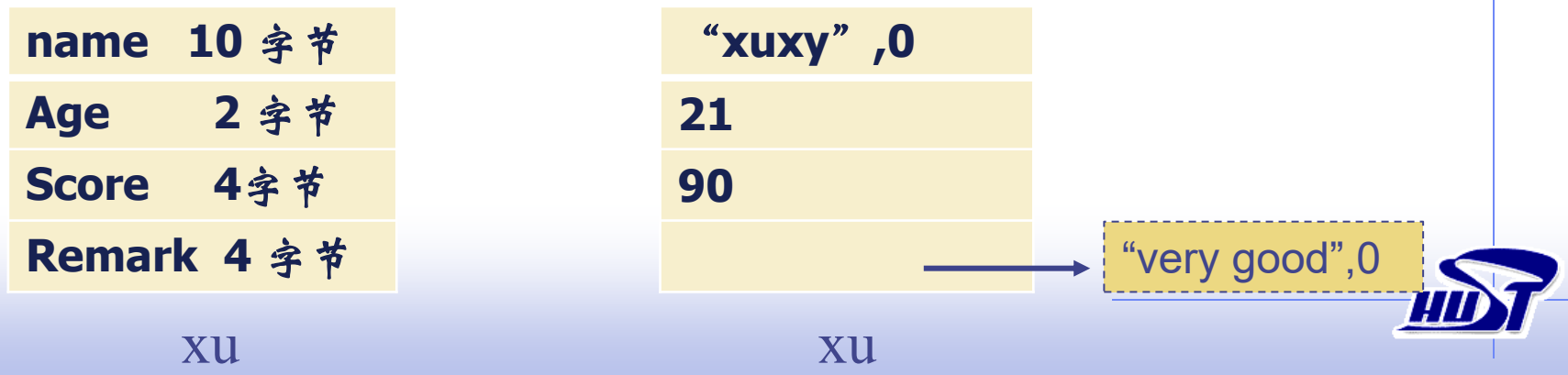
4.3 构造函数

```
Student   xu = { "xuxy", 21, 90, "very good" };
Student   xu1 ( "xuxy", 21, 90, "very good" );
Student   xu2 =Student( "xuxy", 21, 90, "good" );
```

Q: sizeof(Student)= 20;

sizeof(xu) =?

- 分配了 20个字节的空间 （可认为是编译时，预留空间）
- 自动调用构造函数，对20个节进行初始化 （执行语句时）



4.3 构造函数


```
Student  xu = { "xuxy", 21, 90, "very good" };

Student  zhang;   // 没有合适的默认构造函数
                  // 此时需要无参的构造函数
```

在一个构造函数都没有时，Student zhang; 是正确的。

在无构造函数时，编译器提供了默认的无参数构造函数。

当类中定义了构造函数时，就不再提供无参数的构造函数。

4.3 构造函数


```
class Student {
  private:
    char  name[10];      short age;
    float score;         char* remark;
  public:
    Student(char* name, short age, float score, char* remark);
    Student(const char* name, short age, float score,
            const char* remark);
    Student();
};
Student::Student( )     //无参构造函数
{
    strcpy_s(name,"");
    remark = NULL;
}
```

4.3 构造函数

Q： 能否以一个对象来构造另一个对象？

```
Student  xu = { "xuxy", 21, 90, "very good" };
Student  zhang = xu;
Student  yang(xu);
```

Q： 什么时候会用一个对象来构造另一个对象？

- 定义一个对象时：Student zhang = xu;
- 以对象为参数，调用相应的函数，进行参数传递时：

  如 void myfun(Student a){ …}； myfun(xu);

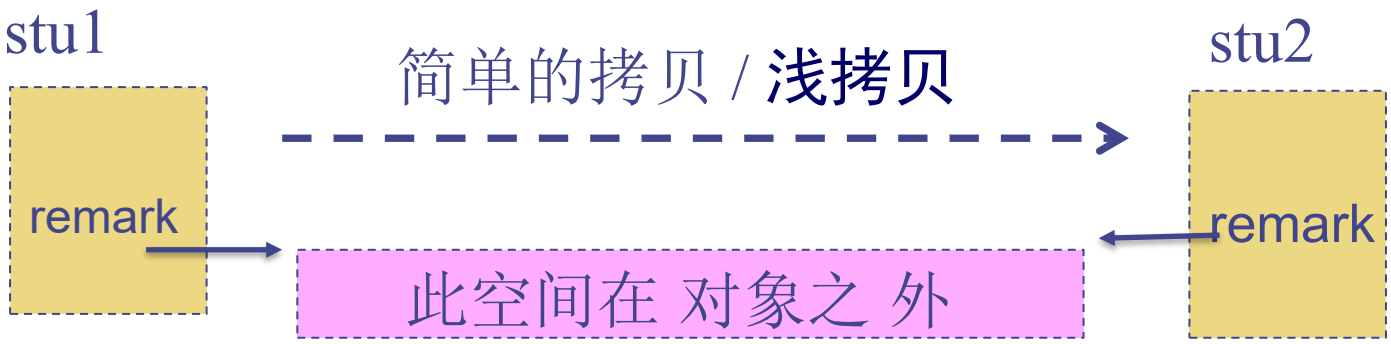
  参数传递等同 Student a=xu;
- 以对象为返回结果，执行相应的return 语句时：

  如 Student myfun(……){ return 对象** };

默认的以对象有址引用为参数的构造函数

Student stu2(stu1);

简单地将 stu1中的内容都拷贝到stu2的中。

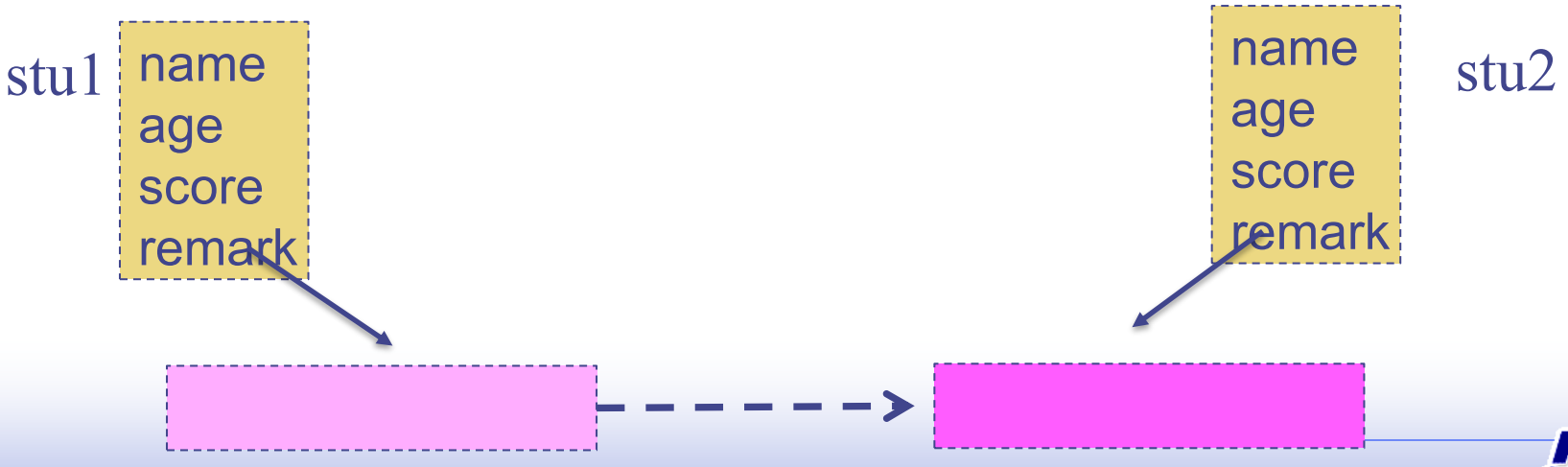


Q：浅拷贝存在什么问题？如何解决问题？

为实现深拷贝，自己编写以对象有址引用为参数的构造函数

```
Student(const Student  &a)
{  ……}

Student stu2(stu1);
```



以对象有址引用为参数的构造函数

```
Student(const  Student   &a)
{   ……}
Student  stu2(stu1);
```

参数传递：const Student &a=stu1;

Q：为什么以对象参数的构造函数形式不是
Student(Student a);

Q：为什么以对象参数的构造函数形式不是
Student(Student &a); 而要在参数上加 const?

4.3 构造函数

Q：Student(const Student &a)，为什么要加 const?

场景1 ：

```
const Student unchange = { "XXXX",100,100,"unchang" };
Student xu = unchange;
```

编译器报错：无法从 const Student 转换为 Student。

本质： **Student &a=常量对象；**

场景2 ：

```
Student CreateStudent()     // 普通函数，非Student的函数成员
{  Student temp("temp", 0, 0, NULL);
   return temp;
}
Student ppqq = CreateStudent();
```

编译器报错：Student 没有适当的复制构造函数。
无法从 Student 转换为 Student。

参数传递等同 Student &a = 临时对象；

临时对象可视为不可修改的。无const时，可以通过 a 来修改！

4.3 构造函数

华中科技大学

对象复制构造示例

```
Student::Student(const Student& s)
{
        cout << "construct : object parameter" << endl;
        strcpy_s(name, s.name);
        age = s.age;
        score = s.score;
        if (s.remark == NULL)
                remark = NULL;
        else {
                int len = strlen(s.remark) + 1;
                remark = (char*)malloc(len);
                strcpy_s(remark,len, s.remark);
        }
}
```

4.3 构造函数


```
void Student::compareAgeWithOther(Student s)
{
        cout << name << " age " << age << " ";
        if (age < s.age) cout << " < ";
        else if (age == s.age) cout << " = ";
        else  cout<< " > ";
        cout << s.name << " age " << s.age << endl;
}

int main()
{       Student  xu = { "xu",20,80,"good" };
        Student  yang("yang", 21, 90, "excellent");
        xu.compareAgeWithOther(yang);
        return 0;
}
```

以对象为参数

参数传递：
Student s = yang;
对象复制构造

4.3 构造函数

对象复制构造

类名(const 类名 &src);

- 为什么加 const ?
  被复制的对象可以是常对象、临时对象。

- 为什么要使用 引用 & ?
  参数传递的是指针。

- 何时自动调用 对象复制构造?
  用一个对象定义一个另一个对象;
  参数为对象; 返回结果为对象。

4.3 构造函数

移动构造　类名(类名 &&src);

对象复制构造　类名(const 类名 &src);

移动构造 ：无址引用为参数的构造函数

```
Student:: Student(const Student& s)
{…
     if (s.remark == NULL)       remark = NULL;
        else {  int len = strlen(s.remark) + 1;
                remark = (char*)malloc(len);
                strcpy_s(remark,len, s.remark);
        }
}
```

以对象有址引用、无址引用为参数的构造函数的差别！

```
Student::Student(Student&& s)
{   cout << "&& construct : temp object parameter" << endl;
    strcpy_s(name, s.name);
    age = s.age;
    score = s.score;
    remark = s.remark;      // 直接接管临时(局部)对象指针指向的空间
    s.remark = NULL;
}
```

构造函数及示例

```
Student(const char* name, short age, float score,
        const char* remark);
Student();
Student(const Student& s);
Student(Student && s);
```

```
Student xu("xuxy", 21, 90, "very good");
Student yang;
Student ma(xu);
const Student unchange = { "XXXX",100,100,"unchang" };
Student  pq = unchanged;
Student ppqq = CreateStudent();
```

```
Student CreateStudent()     // 普通函数，非Student的函数成员
{  Student temp("temp", 0, 0, NULL);
   return temp;
}
```

4.3 构造函数

构造函数 ：并不分配对象的体内空间，而是初始化对象！
可为指针类型的成员申请指向的空间，并初始化！

- 当类中未定义构造函数时，编译器提供默认的无参构造函数；
- 当类中定义了构造函数时，不再提供默认的无参构造函数；
- 编译器提供默认的以对象有址引用为参数的构造函数

  类名(const 类名 &A);

  该函数实现对象A 的简单拷贝——浅拷贝
- 编译器提供默认的以对象无址引用为参数的构造函数

  类名(类名 &&A);

  该函数实现对象A 的简单拷贝——浅拷贝

4.3 构造函数

Q: 默认的无参构造函数， 完成什么功能？

定义数据成员时给出了初值，会使用该函数完成初始化。

- 当类中定义了构造函数时，不再提供默认的无参构造函数；

Q: 如果此时希望有无参构造函数，并且完成与编译器提供的默认无参构造函数相同的功能， 怎么办？

即，要求编译器生成默认的无参构造函数！

构造函数名( ) = default;

Q: 若有 *构造函数名*( ) = default; ，又写了一个无参构造函数，会出现什么情况？

编译器报错：已定义的成员函数

4.3 构造函数

- 编译器提供默认的以对象有址引用为参数的构造函数

  即　复制构造　　类名(const 类名 &A);

  该函数实现对象A 的简单拷贝——浅拷贝

Q: 若不希望用一个对象来构造另外一个对象，也就是不但自己不写“以对象有址引用为参数的构造函数”，也不希望编译器提供默认的以对象为参数的构造函数，怎么办 ？

```
类名(const 类名 &A) = delete;
```

4.3 构造函数

记住

- 编译器提供默认的：

无参构造函数	类名( )
以对象有址引用为参数的构造函数	类名(const 类名 &A)
以对象无址引用为参数的构造函数	类名(类名 &&A)

- 若一定要编译器提供默认的构造函数：

```
类名( ) = default;
类名(const 类名 &A) = default;
类名(类名 &&A) = default;
```

- 一定不让定义（使用）的构造函数：

```
…… = delete;
```

- 自己编写了相应的构造函数，编译器就不再提供默认的构造函数

4.3 构造函数

Q：从语言的角度，构造函数是否一定要与类名相同？

C++中，构造函数与类名相同。

python中，构造函数为 __init__(…)

Q：成员函数中，对象参数是否一定要以隐藏的形式出现？

C++中，隐藏的对象参数为 this。

python中，显示给出对象参数 self。

Q：定义构造函数有什么好处？

对象一降生，就应该拥有相应的属性值；

避免出现未初始化的情况；

表达更简单，定义对象时自动调用构造函数。

4.4 析构函数

Q: Student 类中有指针 remark，
何时、如何释放 remark 指向的空间？

```
Student::~Student()
{   cout << "deconstruct :" << name << endl;
    if (remark) {
        free(remark);
        remark = NULL;
     }
}
```

在一个对象生命周期结束时，会自动调用析构函数。由析构函数释放 remark 指向的空间。

自动：就是编译器生成析构函数的调用语句。

4.4 析构函数

- 只有一个析构函数；
- 函数名与类型名相同，前面加 ~；
- 无返回；
- 无参数；
- 可以写析构函数的调用语句；
- **析构函数并不释放 对象本身的空间**；
  只是做释放体外空间等工作。

```
~Student( );
```

4.4 析构函数

在一个对象生命周期结束时，会自动调用析构函数。

Q: 一个对象的生命周期，何时结束？

对象的生命周期结束时间取决于其创建方式。

- 对于自动对象，是作用域结束；（局部对象，参数对象）
- 对于静态对象，是程序结束；
- 对于动态对象，是delete操作；
- 对于临时对象，一般是表达式结束 （函数返回对象）。

4.4 析构函数

Q: 析构函数，要完成一些什么工作?

```
class IntArray {
{  private:  int len;  int *data;
   public:   ……
      ~IntArray( )  { … }
      void show( ) { 显示所有元素 }
};
```

```
IntArray::~IntArray( ) {
    if (data)  free(data);
    len=0;
    data = null;
};
```

不只是释放空间，还应该
len= 0; data=null;

```
IntArray   t (5);
t. ~IntArray( ) ;
t. show( );
```

4.5 完整示例

一个完整的例子

Student.h	定义类型
Student.cpp	函数成员的定义
Main.cpp	主程序

重点理解：

- 构造函数的自动调用
- 对象定义与构造函数的对应关系
- 析构函数的自动调用
- 函数中定义对象的析构顺序

4.5 完整示例

一个完整的例子　　Student.h

```
class Student {
private:  char  name[10];
          short age;
          float score;
          char* remark;
public:
    Student(const char* name, short age, float score,
            const char* remark);
    Student();
    Student(const Student& s);
    void displayRemark();
    void displayName();
    int getAge();
    Student& operator =(const  Student& s);
    ~Student();
};
```

4.5 完整示例

一个完整的例子　Student.cpp

```
#include "Student.h"
#include <iostream>
using namespace std;
Student::Student(const char* name, short age, float score,
                 const char* remark)
{   cout << "general construct" << endl;
    int len;
    if (name) {
        len = strlen(name);
        if (len <= 9)  strcpy_s(this->name, name);
        else { cout << "name is too long;  cut  string" << endl;
               memcpy(this->name, name, 9);
               this->name[9] = 0;  }
    }
    else  this->name[0] = 0;
    (*this).age = age;
    Student::score = score;
    if (remark) {  len = strlen(remark)+1;
                   this->remark = (char*)malloc(len);
                   strcpy_s(this->remark,len, remark);
    } else this->remark = NULL;
}
```

4.5 完整示例

一个完整的例子　Student.cpp

```
Student::Student()
{
    cout << "no parameter construct" << endl;
    name[0] = 0;
    age = 0;
    score = 0;
    remark = NULL;
}

Student::~Student()
{
   cout << "deconstruct :" << name << endl;
   if (remark) {
      free(remark);
      remark = NULL;
    }
}
```

4.5 完整示例

一个完整的例子　Student.cpp

```
void Student::displayRemark()
{
    cout<<"remark : "<<  remark<<endl;
}

void Student::displayName()
{
    cout<< “name : ”<<  name<<endl;
}

int Student::getAge( )
{
   return age;
}
```

4.5 完整示例

一个完整的例子　main.cpp

```
# include "Student.h"
#include <iostream>
using namespace std;
int main()
{
    Student xu("xuxy123456789012", 21, 90, "very good student");
    Student zhang("zhang", 22, 95, 0);
    Student yang;
    Student li("lishi", 20, 85, "good");
    Student ma(li);
    xu.displayName();
    cout << "zhang . score = " << zhang.getScore() << endl;
    int  x = zhang.getAge();
    cout << "zhang . age = " << x << endl;
    return 0;
}
```

4.5 完整示例

运行结果

```
general construct
name is too long;  cut  string
general construct
no parameter construct
general construct
construct : object parameter
name : xuxy12345
zhang . score = 95
zhang . age = 22
deconstruct :lishi
deconstruct :lishi
deconstruct :noname
deconstruct :zhang
deconstruct :xuxy12345
```

```
Student xu("xuxy123456789012",
  21, 90, "very good student");
Student zhang("zhang", 22, 95, 0);

Student yang;
Student li("lishi", 20, 85, "good");

Student ma(li);
xu.displayName();
cout << "zhang . score = " <<
          zhang.getScore() << endl;
int  x = zhang.getAge();
cout <<"zhang . age = "<< x << endl;
return 0;
```

测验

- 类是如何定义的？
- 访问权限有哪几种？
- 构造函数的定义形式？
- 构造函数的种类有哪些？
- 编译器默认提供的构造函数有哪三个？
- 什么时候，编译器不再提供默认的构造函数？
- 默认的以对象为参数的构造函数，实现什么功能？
- 如何保留或者废除默认的构造函数？
- 为什么要保留或者废除默认的构造函数（应用场景）？
- 自动调用各种构造函数的对应的场景是什么？
- 浅拷贝是什么意思？移动构造是什么意思？

测验

- 析构函数的定义形式？
- 数据成员、函数成员如何访问？
- 隐含第一参数 this 是什么含义？
- 一个函数成员如何调用另一个函数成员？
- 一个函数中，以对象为参数 与 以对象引用为参数，有何差别? 参数传递的过程是什么？
- 对于函数参数，若不允许在函数中修改，加const，有什么好处？
- 对象的空间是何时分配、何时释放的？

关键字和符号

- class
- private、protected、public
- default 、delete
- const
- &、&&、~、::

作业

- 建立一个教师对象数组（N个对象，#define N 3）

教师的数据成员如下

```
class Teacher {
private:
        char  *name;          short age;     ……
};
```

公有函数成员有：

- 简单类型参数的构造函数、复制构造函数、移动构造函数；
- 要能够使用编译器默认的无参构造函数；
- 析构函数，释放name指向的空间；
- 修改教师信息的函数、显示教师信息的函数；

完成的功能包括：各个教师修改自己的信息；依次显示教师数组中各教师的信息；

另外定义几个教师对象，完成对三个构造函数的测试。

提交的作业中包含源程序，以及程序运行截图。

作业

- 特别提示

  对于移动构造，一般应定义一个函数，其返回为一个对象；

  再定义一个对象，用返回的临时对象来构造。

还有一种特殊用法：

Teacher xu(…..);

Teacher xu_bak = move(xu); 此时，也是调用移动构造。

但要注意，执行移动构造后， xu 中的name 指针为空。