

构造函数

主讲：陈笑沙

目录

- 4.1 类与对象
- 4.2 构造函数的必要性
- 4.3 构造函数的使用
- 4.4 析构函数
- 4.5 带参数的构造函数
- 4.6 重载构造函数
- 4.7 默认构造函数
- 4.8 类成员初始化
- 4.9 构造对象的顺序

4.1 类与对象

类是 C++ 自定义数据类型——描述数据组织与操作

```
class Point {  
    int x, y; // 数据组织  
public:  
    void set(int ix, int iy) { // 操作  
        x = ix;  
        y = iy;  
    }  
};
```

4.1 类与对象

对象是类的实体

变量是基本类型的实体

```
int a = 3; // 基本类型变量
struct Date { // 结构变量
    int year, month, day;
};
Point w;
w.set(3, 5);
```

4.1 类与对象

C ++是一种混合型程序设计语言

4.1 类与对象

C ++是一种混合型程序设计语言

- 简单数据结构,单纯复合控制语句,函数(模块)设计,构成过程化程序设计

4.1 类与对象

C ++是一种混合型程序设计语言

- 简单数据结构,单纯复合控制语句,函数(模块)设计,构成过程化程序设计
- 以类及对象为数据基础, 以过程化程序设计为框架,构成对象化程序设计

4.1 类与对象

C ++是一种混合型程序设计语言

- 简单数据结构,单纯复合控制语句,函数(模块)设计,构成过程化程序设计
- 以类及对象为数据基础, 以过程化程序设计为框架,构成对象化程序设计
- 以类系多态为数据处理主要对象,以对象化设计为框架,构成面向对象程序设计

4.2 构造函数的必要性

数据实体都有初始化的诉求

```
int a = 3;      // 整型变量初始化
int a; a = 3;  // 整型变量赋初值
double t[] = {1.3, 2.5}; // 数组初始化
struct Date {int year, month, day; };
Date d = {1998, 5, 23}; // 结构体变量初始化
```

4.2 构造函数的必要性

对于对象来说，初始化是一个复杂的问题

```
class Point {  
    int x, y;  
public:  
    void set(int ix, int iy) {  
        x = ix;  
        y = iy;  
    }  
};  
int main() {  
    Point t = {3, 4}; // 错误，直接给私有变量赋值  
    Point d; // 产生未初始化对象  
    d.set(3, 4); // 赋初值，非初始化  
}
```

4.2 构造函数的必要性

如果全部是公有变量，则可以这样操作：

```
1 class Point {  
2     public:  
3         int x, y;  
4     };  
5  
6     int main() {  
7         Point t = {3, 4}; // W  
8         return 0;  
9     }
```

4.2 构造函数的必要性

如果全部是公有变量，则可以这样操作：

```
1 class Point {  
2     public:  
3         int x, y;  
4     };  
5  
6     int main() {  
7         Point t = {3, 4}; // W  
8         return 0;  
9     }
```

4.2 构造函数的必要性

如果全部是公有变量，则可以这样操作：

```
1 class Point {  
2     public:  
3         int x, y;  
4     };  
5  
6     int main() {  
7         Point t = {3, 4}; // W  
8         return 0;  
9     }
```

4.3 构造函数的使用

对象初始化的设计

应在构造对象时完成初始化，无关访问权限

```
class A {  
    int a, b;  
};  
A x = {2, 3}; // OK  
A y{2, 3}; // OK  
A z(2, 3); // OK
```

4.3 构造函数的使用

构造函数的语法：

4.3 构造函数的使用

构造函数的语法：

- 没有返回类型（甚至没有 void）

4.3 构造函数的使用

构造函数的语法：

- 没有返回类型（甚至没有 `void`）
- 名称与类的名称相同（大小写也需一致）

4.3 构造函数的使用

构造函数的语法：

- 没有返回类型（甚至没有 `void`）
- 名称与类的名称相同（大小写也需一致）
- 可以是任意访问权限，不一定非得是 `public`

4.3 构造函数的使用

构造函数的语法：

- 没有返回类型（甚至没有 `void`）
- 名称与类的名称相同（大小写也需一致）
- 可以是任意访问权限，不一定非得是 `public`
- 一个类可以有多个构造函数

4.3 构造函数的使用

构造函数的语法：

- 没有返回类型（甚至没有 `void`）
- 名称与类的名称相同（大小写也需一致）
- 可以是任意访问权限，不一定非得是 `public`
- 一个类可以有多个构造函数

```
1 class A {  
2     int a, b;  
3 public:  
4     void A(int ia, int ib); // Wrong  
5     construct(int ia, int ib); // Wrong  
6     A(int ia, int ib); // Right  
7 };
```

4.3 构造函数的使用

构造函数的语法：

- 没有返回类型（甚至没有 `void`）
- 名称与类的名称相同（大小写也需一致）
- 可以是任意访问权限，不一定非得是 `public`
- 一个类可以有多个构造函数

```
1 class A {  
2     int a, b;  
3 public:  
4     void A(int ia, int ib); // Wrong  
5     construct(int ia, int ib); // Wrong  
6     A(int ia, int ib); // Right  
7 };
```

4.3 构造函数的使用

构造函数的语法：

- 没有返回类型（甚至没有 `void`）
- 名称与类的名称相同（大小写也需一致）
- 可以是任意访问权限，不一定非得是 `public`
- 一个类可以有多个构造函数

```
1 class A {  
2     int a, b;  
3 public:  
4     void A(int ia, int ib); // Wrong  
5     construct(int ia, int ib); // Wrong  
6     A(int ia, int ib); // Right  
7 };
```

4.3 构造函数的使用

构造函数的语法：

- 没有返回类型（甚至没有 `void`）
- 名称与类的名称相同（大小写也需一致）
- 可以是任意访问权限，不一定非得是 `public`
- 一个类可以有多个构造函数

```
1 class A {  
2     int a, b;  
3 public:  
4     void A(int ia, int ib); // Wrong  
5     construct(int ia, int ib); // Wrong  
6     A(int ia, int ib); // Right  
7 };
```

4.3 构造函数的使用

构造函数的语法：

- 没有返回类型（甚至没有 `void`）
- 名称与类的名称相同（大小写也需一致）
- 可以是任意访问权限，不一定非得是 `public`
- 一个类可以有多个构造函数

```
1 class A {  
2     int a, b;  
3 public:  
4     void A(int ia, int ib); // Wrong  
5     construct(int ia, int ib); // Wrong  
6     A(int ia, int ib); // Right  
7 };
```

4.3 构造函数的使用

例：为 Clock 类添加构造函数

```
class Clock {  
    int hour, minute, second;  
public:  
    Clock(int h, int m, int s) {  
        ...  
    }  
};
```

4 .3 构造函数的使用

实现构造函数： 初始化成员变量

```
Clock(int h, int m, int s) {  
    hour = h;  
    minute = m;  
    second = s;  
}
```

4.3 构造函数的使用

实现构造函数：初始化成员变量

```
Clock(int h, int m, int s) : hour(h), minute(m), second(s) {}
```

4 .3 构造函数的使用

构造函数的调用：

```
1 class Desk {  
2 public:  
3     Desk();  
4 private:  
5     int height, width;  
6 };  
7  
8 Desk::Desk() {  
9     cout << "Constructor of Desk." << endl;  
10    height = 3;  
11    width = 2;  
12 }  
13  
14 Desk desk; // call the constructor
```

4 .3 构造函数的使用

构造函数的调用：

```
1 class Desk {  
2 public:  
3     Desk();  
4 private:  
5     int height, width;  
6 };  
7  
8 Desk::Desk() {  
9     cout << "Constructor of Desk." << endl;  
10    height = 3;  
11    width = 2;  
12 }  
13  
14 Desk desk; // call the constructor
```

4 .3 构造函数的使用

构造函数的调用：

```
1 class Desk {  
2 public:  
3     Desk();  
4 private:  
5     int height, width;  
6 };  
7  
8 Desk::Desk() {  
9     cout << "Constructor of Desk." << endl;  
10    height = 3;  
11    width = 2;  
12 }  
13  
14 Desk desk; // call the constructor
```

4 .3 构造函数的使用

构造函数的调用：

```
1 class Desk {  
2 public:  
3     Desk();  
4 private:  
5     int height, width;  
6 };  
7  
8 Desk::Desk() {  
9     cout << "Constructor of Desk." << endl;  
10    height = 3;  
11    width = 2;  
12 }  
13  
14 Desk desk; // call the constructor
```

4 .3 构造函数的使用

构造函数的调用：

```
1 class Desk {  
2 public:  
3     Desk();  
4 private:  
5     int height, width;  
6 };  
7  
8 Desk::Desk() {  
9     cout << "Constructor of Desk." << endl;  
10    height = 3;  
11    width = 2;  
12 }  
13  
14 Desk desk; // call the constructor
```

4.4 析构函数

类设计中若存在指针成员,则构造函数会申请堆内存
赋给该指针,成员函数就可共享该资源

```
class A {  
    int* aa;  
    int num;  
public:  
    A(int n) {  
        num = n;  
        aa = new int[n];  
        // delete[] aa; ???  
    }  
};
```

4.4 析构函数

4.4 析构函数

- 有时我们在释放类对象的时候，可能需要做一些收尾工作，但也可能会因为忘记调用这些做收尾工作的函数而出现问题，这个怎么解决？

4.4 析构函数

- 有时我们在释放类对象的时候，可能需要做一些收尾工作，但也可能会因为忘记调用这些做收尾工作的函数而出现问题，这个怎么解决？
- C++也为我们考虑了，与构造函数相对的，提供了析构函数专门用于处理对象销毁时候的清理工作。

4.4 析构函数

析构函数没有返回类型，没有参数，函数名是在类名前加“~”。析构函数将会在对象的生存期结束后被自动调用。

析构函数也可以是私有的，但是不常见。

4.4 析构函数

```
1 class A {  
2     int* aa;  
3     int num;  
4 public:  
5     A(int n) {  
6         num = n;  
7         aa = new int[n];  
8     }  
9     ~A() { // 不能有参数  
10        delete[] aa;  
11    }  
12};
```

4.4 析构函数

```
1 class A {  
2     int* aa;  
3     int num;  
4 public:  
5     A(int n) {  
6         num = n;  
7         aa = new int[n];  
8     }  
9     ~A() { // 不能有参数  
10        delete[] aa;  
11    }  
12};
```

4.4 析构函数

```
1 class A {  
2     int* aa;  
3     int num;  
4 public:  
5     A(int n) {  
6         num = n;  
7         aa = new int[n];  
8     }  
9     ~A() { // 不能有参数  
10        delete[] aa;  
11    }  
12};
```

4.5 带参数的构造函数

通过向构造函数传入参数，可以指定成员变量的值。

```
class Student {
    std::string name;
    char gender;
    double gpa;
public:
    Student(const std::string& name, char gender, double gpa):
        name(name), gender(gender), gpa(gpa) {}
};

Student stu("Eric", 'M', 4.5);
```

4.5 带参数的构造函数

```
1 class Teacher {
2     std::string name;
3     uint8_t age;
4 public:
5     Teacher(const std::string& name, uint8_t age) {
6         this->name = name;
7         this->age = age;
8     }
9 }
10
11 class Student {
12     std::string name;
13     char gender;
14     double gpa;
15     Teacher t;
```

4.5 带参数的构造函数

```
1 class Teacher {
2     std::string name;
3     uint8_t age;
4 public:
5     Teacher(const std::string& name, uint8_t age) {
6         this->name = name;
7         this->age = age;
8     }
9 }
10
11 class Student {
12     std::string name;
13     char gender;
14     double gpa;
15     Teacher t;
```

4.5 带参数的构造函数

```
1 class Teacher {
2     std::string name;
3     uint8_t age;
4 public:
5     Teacher(const std::string& name, uint8_t age) {
6         this->name = name;
7         this->age = age;
8     }
9 }
10
11 class Student {
12     std::string name;
13     char gender;
14     double gpa;
15     Teacher t;
```

4.5 带参数的构造函数

```
6     .name = name,
7     this->age = age;
8 }
9 };
10
11 class Student {
12     std::string name;
13     char gender;
14     double gpa;
15     Teacher t;
16 public:
17     Student(const std::string& name, char gender, double gpa
18             name(name), gender(gender), gpa(gpa) {
19         t(name, 30); // Wrong
20     }
21 }
```

4.5 带参数的构造函数

```
7         this->age = age;
8     }
9 }
10
11 class Student {
12     std::string name;
13     char gender;
14     double gpa;
15     Teacher t;
16 public:
17     Student(const std::string& name, char gender, double gpa)
18         name(name), gender(gender), gpa(gpa) {
19             t(name, 30); // Wrong
20     }
21 };    Teacher t,
```

4.5 带参数的构造函数

4.5 带参数的构造函数

如果构造函数仅有一个参数，也可以这样初始化：

4.5 带参数的构造函数

如果构造函数仅有一个参数，也可以这样初始化：

```
className obj = parameter;
```

4.5 带参数的构造函数

如果构造函数仅有一个参数，也可以这样初始化：

```
className obj = parameter;
```

为了防止这种情况，可以进行如下设置：

4.5 带参数的构造函数

如果构造函数仅有一个参数，也可以这样初始化：

```
className obj = parameter;
```

为了防止这种情况，可以进行如下设置：

```
1 class A{
2     int a;
3 public:
4     explicit A(int n):a(n){} // 注意新的关键字
5 };
6
7 int main() {
8     A a = 1; // Wrong
9     return 0;
10 }
```

4.5 带参数的构造函数

如果构造函数仅有一个参数，也可以这样初始化：

```
className obj = parameter;
```

为了防止这种情况，可以进行如下设置：

```
1 class A{
2     int a;
3 public:
4     explicit A(int n):a(n){} // 注意新的关键字
5 };
6
7 int main() {
8     A a = 1; // Wrong
9     return 0;
10 }
```

4.5 带参数的构造函数

如果构造函数仅有一个参数，也可以这样初始化：

```
className obj = parameter;
```

为了防止这种情况，可以进行如下设置：

```
1 class A{  
2     int a;  
3 public:  
4     explicit A(int n):a(n){} // 注意新的关键字  
5 };  
6  
7 int main() {  
8     A a = 1; // Wrong  
9     return 0;  
10 }
```

4.5 带参数的构造函数

如果构造函数仅有一个参数，也可以这样初始化：

```
className obj = parameter;
```

为了防止这种情况，可以进行如下设置：

```
1 class A{
2     int a;
3 public:
4     explicit A(int n):a(n){} // 注意新的关键字
5 };
6
7 int main() {
8     A a = 1; // Wrong
9     return 0;
10 }
```

4.5 带参数的构造函数

4.5 带参数的构造函数

- 题外话：函数的字符串参数应该是什么类型？

4.5 带参数的构造函数

- 题外话：函数的字符串参数应该是什么类型？
 - `const char*`

4.5 带参数的构造函数

- 题外话：函数的字符串参数应该是什么类型？
 - `const char*`
 - `const std::string&`

4.5 带参数的构造函数

- 题外话：函数的字符串参数应该是什么类型？
 - `const char*`
 - `const std::string&`
- 一般而言，`const char*` 性能更好，但是不安全。

4.5 带参数的构造函数

- 题外话：函数的字符串参数应该是什么类型？
 - `const char*`
 - `const std::string&`
- 一般而言，`const char*` 性能更好，但是不安全。
- 大部分情况，不必要如此极致地考虑性能。

4.5 带参数的构造函数

- 题外话：函数的字符串参数应该是什么类型？
 - `const char*`
 - `const std::string&`
- 一般而言，`const char*` 性能更好，但是不安全。
- 大部分情况，不必要如此极致地考虑性能。
- 一般情况选 `const std::string&`。

4.5 带参数的构造函数

- 题外话：函数的字符串参数应该是什么类型？
 - `const char*`
 - `const std::string&`
- 一般而言，`const char*` 性能更好，但是不安全。
- 大部分情况，不必要如此极致地考虑性能。
- 一般情况选 `const std::string&`。
- C++17 之后，可以使用 `std::string_view`。

4.5 带参数的构造函数

4.5 带参数的构造函数

- 有时构造函数参数与成员变量名称相同，则可以：

4.5 带参数的构造函数

- 有时构造函数参数与成员变量名称相同，则可以：
 - 使用 `this` 指针

4.5 带参数的构造函数

- 有时构造函数参数与成员变量名称相同，则可以：
 - 使用 `this` 指针
 - 使用初始化成员列表

4.5 带参数的构造函数

- 有时构造函数参数与成员变量名称相同，则可以：
 - 使用 `this` 指针
 - 使用初始化成员列表
- `this` 指针是类成员函数中指向当前对象实例的隐含指针，用于访问对象的成员。

4.5 带参数的构造函数

```
class Point {  
    int x, y;  
public:  
    Point(int x, int y) : x(x), y(y) {}  
};
```

4.5 带参数的构造函数

```
class Point {  
    int x, y;  
public:  
    Point(int x, int y){  
        this->x = x;  
        this->y = y;  
    }  
};
```

4.5 带参数的构造函数

以下两种方式调用构造函数是等价的：

```
Point p(3, 4); // 传统方式
Point p = {3, 4}; // 如果构造函数为explicit，则编译失败
Point p{3, 4}; // 推荐的方式，c++11以后的方式，与函数区分
```

4.6 重载构造函数

4.6 重载构造函数

- C++ 成员函数可重载, 构造函数同样也可重载

4.6 重载构造函数

- C++ 成员函数可重载, 构造函数同样也可重载
- 一个类可以提供多个构造函数, 即构造函数的重载。

4.6 重载构造函数

- C++ 成员函数可重载, 构造函数同样也可重载
- 一个类可以提供多个构造函数, 即构造函数的重载。
- 重载的目的是为了满足不同的初始化需要。

4.6 重载构造函数

```
1 class Clock
2 {
3     private:
4         int hour, minute, second;
5     public:
6         Clock(int h, int m, int s);
7         Clock();
8         Clock(const std::string& timestr);
9     };
10
11 int main( )
12 {
13     Clock clock1{23, 12, 0};
14     Clock clock2{};
15     Clock clock3{"14:45:32"};
```

4.6 重载构造函数

```
1 class Clock
2 {
3     private:
4         int hour, minute, second;
5     public:
6         Clock(int h, int m, int s);
7         Clock();
8         Clock(const std::string& timestr);
9     };
10
11 int main( )
12 {
13     Clock clock1{23, 12, 0};
14     Clock clock2{};
15     Clock clock3{"14:45:32"};
```

4.6 重载构造函数

```
2 i
3 private:
4     int hour, minute, second;
5 public:
6     Clock(int h, int m, int s);
7     Clock();
8     Clock(const std::string& timestr);
9 }
10
11 int main()
12 {
13     Clock clock1{23, 12, 0};
14     Clock clock2{};
15     Clock clock3{"14:45:32"};
16 }
```

4.6 重载构造函数

该如何实现以下拥有多种构造方式的类？

```
int main()
{
    Date date1{2000, 3, 4};
    Date date2{2000, 3};
    Date date3{2000};
    Date date4{};
    return 0;
}
```

4.6 重载构造函数

方案一（重载）

```
class Date{
    int year, month, day; // 默认private
public: // 以下四个重载函数，每个对应一种构建对象方式
    Date();
    Date(int d);
    Date(int m, int d);
    Date(int y, int m, int d);
};

Date::Date(){ year=1900; month=1; day=1; }
Date::Date(int y){ month=4; day=d; year=1996; }
Date::Date(int y, int m){ month=m; day=1; year=1900; }
Date::Date(int y, int m, int d){ month=m; day=d; year=y; }
```

4.6 重载构造函数

方案二 (C++11)

```
class Date {  
    int year, month, day;  
public:  
    Date(int year, int month, int day) : year(year), month(month)  
    Date(int year, int month) : Date(year, month, 1) {}  
    Date(int year) : Date(year, 1, 1) {}  
    Date() : Date(1900, 1, 1) {}  
};
```

4.6 重载构造函数

方案三（默认参数，推荐）

```
class Date
{
    int year, month, day;
public:
    Date(int year = 1900, int month = 1, int day = 1)
        : year(year), month(month), day(day)
    {
    }
};
```

4.7 默认构造函数

4.7 默认构造函数

- 默认构造函数不是默认参数构造函数

4.7 默认构造函数

- 默认构造函数不是默认参数构造函数
- 创建对象一定需要构造函数

4.7 默认构造函数

- 默认构造函数不是默认参数构造函数
- 创建对象一定需要构造函数
- 默认构造函数：类中若无构造函数定义，系统会默认一个无参构造函数，以完成创建对象的使命

4.7 默认构造函数

- 默认构造函数不是默认参数构造函数
- 创建对象一定需要构造函数
- 默认构造函数：类中若无构造函数定义，系统会默认一个无参构造函数，以完成创建对象的使命
- 程序员只要定义构造函数(不管几个)，则系统不再默认构造函数

4.7 默认构造函数

- 默认构造函数不是默认参数构造函数
- 创建对象一定需要构造函数
- 默认构造函数：类中若无构造函数定义，系统会默认一个无参构造函数，以完成创建对象的使命
- 程序员只要定义构造函数(不管几个)，则系统不再默认构造函数
- 默认构造函数一定是无参构造函数,无参构造函数可自定义

4.7 默认构造函数

- 默认构造函数不是默认参数构造函数
- 创建对象一定需要构造函数
- 默认构造函数：类中若无构造函数定义，系统会默认一个无参构造函数，以完成创建对象的使命
- 程序员只要定义构造函数(不管几个)，则系统不再默认构造函数
- 默认构造函数一定是无参构造函数,无参构造函数可自定义
- 实际上，也存在与之对应的默认析构函数

4.7 默认构造函数

```
1 class Date
2 {
3 };
4
5 int main()
6 {
7     Date date{}; // ok
8     Date date2; // ok
9     Date date3(); // ok
10    return 0;
11 }
```

4.7 默认构造函数

```
1 class Date
2 {
3 };
4
5 int main()
6 {
7     Date date{}; // ok
8     Date date2; // ok
9     Date date3(); // ok
10    return 0;
11 }
```

4.7 默认构造函数

```
1 class Date
2 {
3 };
4
5 int main()
6 {
7     Date date{}; // ok
8     Date date2; // ok
9     Date date3(); // ok
10    return 0;
11 }
```

4.7 默认构造函数

```
1 class Date
2 {
3 };
4
5 int main()
6 {
7     Date date{}; // ok
8     Date date2; // ok
9     Date date3(); // ok
10    return 0;
11 }
```

4.7 默认构造函数

```
1 class Date
2 {
3 };
4
5 int main()
6 {
7     Date date{}; // ok
8     Date date2; // ok
9     Date date3(); // ok
10    return 0;
11 }
```

4.7 类成员初始化

4.7 类成员初始化

- 构造函数总是先创建对象空间，然后执行构造函数体语句

4.7 类成员初始化

- 构造函数总是先创建对象空间，然后执行构造函数体语句
- 类中若含对象成员，则构造函数创建对象空间完成时，调用对象成员的无参构造函数，然后执行构造函数体语句

4.7 类成员初始化

- 构造函数总是先创建对象空间，然后执行构造函数体语句
- 类中若含对象成员，则构造函数创建对象空间完成时，调用对象成员的无参构造函数，然后执行构造函数体语句
- 对象成员若没有无参构造函数，则应采用成员初始化列表方式

4.7 类成员初始化

```
1 class Teacher {
2     std::string name;
3     uint8_t age;
4 public:
5     Teacher(const std::string& name, uint8_t age) {
6         this->name = name;
7         this->age = age;
8     }
9 }
10
11 class Student {
12     std::string name;
13     char gender;
14     double gpa;
15     Teacher t;
```

4.7 类成员初始化

```
1 class Teacher {
2     std::string name;
3     uint8_t age;
4 public:
5     Teacher(const std::string& name, uint8_t age) {
6         this->name = name;
7         this->age = age;
8     }
9 }
10
11 class Student {
12     std::string name;
13     char gender;
14     double gpa;
15     Teacher t;
```

4.7 类成员初始化

```
6         this->name = name;
7         this->age = age;
8     }
9 }
10
11 class Student {
12     std::string name;
13     char gender;
14     double gpa;
15     Teacher t;
16 public:
17     Student(const std::string& name, char gender, double gpa
18             name(name), gender(gender), gpa(gpa), t("no name", 100)
19             }
20 };
21 }
```

4.7 类成员初始化

```
6         this->name = name;
7         this->age = age;
8     }
9 }
10
11 class Student {
12     std::string name;
13     char gender;
14     double gpa;
15     Teacher t;
16 public:
17     Student(const std::string& name, char gender, double gpa)
18         : name(name), gender(gender), gpa(gpa), t("no name", 100)
19     {}
20 }
21     Teacher t,
```

4.9 构造对象的顺序

成员以其在类中声明的顺序构造

```
class A {  
    int num, age; // 先构造num, 后构造age  
public:  
    A(int n) : age(n), num(age + 1) {} // 与此处顺序无关  
};
```

例子：约瑟夫环问题

