# 类

范 懿

# 目录

# 类的动机（I）

## 例

- 学校管理中需要处理学生这种类型，
- 银行管理中需要账号这种类型，
- 汽车生产中需要汽车这种类型

## 大型程序设计

实际可用的程序，比如

- 物流管理系统，
- 超市销售系统等，

都是功能复杂的程序，变量和过程的交互非常复杂。

# 类的动机（II）

- 代码封装在大型程序设计中有重要意义。
- 松散耦合使程序员着眼于局部，编程更直观，差错更易找。

## 面向过程编程的缺陷

- 全局变量太容易被读写，一旦出错非常难找
- 传递数值的时候，如果仅依赖于函数头，那么
  - 参数列表很长；
  - 频繁复制参数导致效率不高。

要防止数据污染，也要确保效率和可读——面向对象成了必要。

## 例

一个保险箱，如果只有少数几个人知道密码，

- 那么东西丢失之后嫌疑人就好找；
- 如果很多人都知道，那嫌疑人就很难找。

# 从面向过程到面向对象

下面以洗衣为例。

## 面向过程

- 通电、通水、设置参数
- 加水、旋转
- 启动和停止电机
- 停转、出水

## 面向对象

- 水电开关与洗衣机交互
- 电机与开关、转筒交互
- 转筒与进出水口交互
- 控制系统与面板交互

- 面向过程关注的是动词——以上帝视角观察全局
- 面向对象关注的是名词——以层级视角观察周围

☞人非上帝，层级视角更方便。

# 面向对象的基本步骤

- 把系统划分为若干模块
- 确定各个模块的功能
- 确定各个模块分别与哪些其他模块交互
- 确定交互的信息和方式（单向/双向，读/写）

☞面向对象的基本单元——模块（也叫作对象）。

# 创建对象的步骤

实际系统中，对象往往有很多相似之处，逐个创建费时费力。

**例**

- 在一个学校中，学生的个数成千上万
- 在一台汽车中，轮胎有好几个
- 在银行管理中，分支银行就有好几家

因此，我们需要先创建一个模板，然后根据模板创建对象。

# 类和对象的关系 I

**例**

1. 人类是一个“类”；张三、李四、王五、徐六、陈七都是这个类里面的“对象”
2. 鸟类是一个“类”：笼子里的那只鹦鹉，家里看门的那只鹅，昨天才熬好的那只鹰，刚刚送信来的白鸽
3. 兽类是一个“类”：动物园里的这只老虎、那只狮子、这只豹子、那匹狼

类是抽象的，是统称，是概括。对象是类里面的个体或者实体。

# 类和对象的关系 II

☞类给出属性和行为模式，对象给出具体的属性值和行为。

## 例

1. 人有年龄：张三 20 岁，李四 21 岁
2. 人有性别：张三是男，李四是女
3. 人会说话：张三说“四川话”，李四说“潮汕话”
4. 人要吃东西：张三吃花椒，李四吃沙茶酱

## 从类到对象

1. 这是一个从一般到具体的过程
2. 只要给出属性值和行为的具体化即可

# 类的实现

使用关键字`class`

**定义**

一个类定义了一种新的类型，从而把数据和代码连接在一起。

这个类型用于创建类的对象。

☞类和结构体在语法上相像。

```
1  #define SIZE 100
2
3  // 这创建一个堆栈类。
4
5  class Stack {
6      private:
7          int stck[SIZE];
8          int tos;
9      public:
10         void init();
11         void push(int i);
12         int pop();
13 };
14
15 /*这将被用于创建一个堆栈。*/
```

# 类的成分

```
class Stack {
  private:
    int stck[SIZE];
    int tos;
  public:
    void init();
    void push(int i);
    int pop();
};
```

类可能包含private部分，也可能包含public部分。

**例**

- stck 和 tos 是private，它们不能被该类以外的函数访问。
- 这是实现**封装**的一种方式——通过设置private来严格控制对某些数据的访问。
- 也可以有私有函数，它们只能被类内的函数调用。

# 类的成分

```
class Stack {
  private:
    int stck[SIZE];
    int tos;
  public:
    void init();
    void push(int i);
    int pop();
};
```

在public之外后定义的变量或者函数都能在程序各处被访问。

☞良好的编程习惯要求你限制 public 变量的使用。

建议把所有变量设为private，并通过public函数访问它们。

# 成员变量和成员函数

## 成员变量

函数 `init()`，`push()` 以及 `pop()` 被称作成员函数，因为它们是 `Stack` 类的一部分。

## 成员变量

变量 `stck` 和 `tos` 被称作成员变量（或称数据成员）。

☞一个对象形成了代码和数据直接的连接。

成员函数可以访问同一个类内部的私有成员。

## 例

`init()`，`push()` 以及 `pop()` 可以访问 `stck` 和 `tos`。

# 创建对象

类的名字成为一个新的数据类型声明字。

```
Stack myStack;
```

**实例**

声明一个对象的时候，我们在创建该类的一个实例（Instance）。

**例**

`myStack` 是 `Stack` 的一个实例。

也可以像声明结构体变量那样，在声明类的时候同时声明对象。

# 类的使用

一个类创造一种新的类型，该类型可用于创建相应的对象

| 类 | 对象 |
| --- | --- |
| `int` | n |

表: 从使用来看相当于整型和某个相应类型变量的关系

- 类是逻辑上的一种抽象，不占用内存
- 对象是一个实体，占用一定的内存

# 类的定义的一般形式

```
class class-name {
  private:
    private data and functions
  public:
    public data and functions
} object name list;
```

object name list 可以为空

成员函数都必须在类中给出原型的声明。

# 成员函数的实现

实现成员函数时，我们要告知编译器是哪个类的成员函数。

```
void Stack::push(int i)
{
  if(tos == SIZE) {
    cout << "Stack is full.\n";
    return;
  }
  stck[tos] = i;
  tos++;
}
```

☞通过在函数名前面添加类名和双冒号实现。

:: 称为生存域归结符（Scope Resolution Operator）。

# 成员的提及

## 在一个类的外部访问其成员

对象名字 + 点运算符 + 成员名——适用于数据成员和函数成员

以下调用 stack1 中的 init() 函数

## 例

```
Stack stack1, stack2;
stack1.init();
```

上述片段创建两个对象 stack1 和 stack2，并初始化 stack1

- stack1 和 stack2 是两个独立的对象
- 初始化 stack1 并不会导致 stack2 也被初始化

# 成员的提及

## 在类的内部

成员函数可以直接调用另一个成员函数，或访问成员变量

```
1  #include <iostream>
2
3  using namespace std;
4
5  #define SIZE 100
6
7  // 创建一个堆栈类
8  class Stack
9  {
10 private:
11     int stck[SIZE];
12     int tos;
13 public:
14     void init();
15     void push(int i);
16     int pop();
17 };
18
19 void Stack::init()
20 {
21     tos = 0;
```

```
22 }
23
24 void Stack::push(int i)
25 {
26     if(tos == SIZE) {
27         cout << "Stack is full.\n";
28         return;
29     }
30     stck[tos] = i;
31     tos++;
32 }
33
34 int Stack::pop()
35 {
36     if(tos == 0) {
37         cout << "Stack underflow.\n";
38         return 0;
39     }
40     tos--;
41     return stck[tos];
42 }
```

```
43
44 int main()
45 {
46     Stack stack1, stack2; // 创建两个堆栈对象
47     stack1.init();
48     stack2.init();
49     stack1.push(1);
50     stack2.push(2);
51     stack1.push(3);
52     stack2.push(4);
53     cout << stack1.pop() << " ";
54     cout << stack1.pop() << " ";
55     cout << stack2.pop() << " ";
56     cout << stack2.pop() << "\n";
57     return 0;
58 }
59
60 /*
61 示例输出
62 3 1 4 2
63 */
```

```
64
65 /*一个对象的私有成员只能被该对象的成员函数访问。
66 比如stack1.tos = 0;不能被前面的main()函数访问，
67 也就不能被放在前面的main()函数中。
68 */
```

# 类的一般声明

```
class class-name {
    private data and functions
  access-specifier:
    data and functions
  access-specifier:
    data and functions
  // ...
  access-specifier:
    data and functions
   }object-list;
```

object-list 是可选的。如果存在，它声明该类的对象。

access-specifier 可以是：private，protected 或 public。

# 外部访问权限

## 例

一台洗衣机中，

- 接排水口、电插头、盖子和按钮是外部可触摸的——公有
- 电动机和线路是在外部不可触摸的——私有

公有是我们有访问权限的，私有是我们没有访问权限的。

## 访问权限的动因

- 防止发生危险
- 让洗衣机变得易用

# 类的一般声明

类中的成员被默认为是`private`，只能被类中的其他成员访问。

`public`访问声明字允许函数或者变量被任何代码访问。

`protected`访问声明字在继承机制中有用。

某个访问声明字一旦被启用，作用范围会一直到另一个访问声明字被启用，或者类的结束。

可以随意切换访问声明字——比如可以为了声明某些成员而切换到`public`模式然后又切换回`private`模式。

```cpp
#include <iostream>
#include <cstring>

using namespace std;

class Employee
{
    char name[80]; // 默认是private
public:
    void putname(char *n); // 这两函数public
    void getname(char *n);
private:
    double wage; // 现在再次private
public:
    void putwage(double w); // 回到public
    double getwage();
};

void Employee::putname(char *n)
{
    strcpy(name, n);
```

```
22 }
23
24 void Employee::getname(char *n)
25 {
26     strcpy(n, name);
27 }
28
29 void Employee::putwage(double w)
30 {
31     wage = w;
32 }
33
34 double Employee::getwage()
35 {
36     return wage;
37 }
38
39 int main()
40 {
41     Employee ted;
42     char name[80];
```

```
43      ted.putname("Ted Jones");
44      ted.putwage(75000);
45      ted.getname(name);
46      cout << name << " makes $";
47      cout << ted.getwage() << " per year.";
48      return 0;
49  }
```

# 编程规范

对编译器而言，多次使用同一个访问声明字并没有带来区别。

**例**

在同一个类中，大部分程序有都只会使用一个`private`，一个`protected`，和一个`public`部分。

```
class Employee {
    char name[80];
    double wage;
  public:
    void putname(char *n);
    void getname(char *n);
    void putwage(double w);
    double getwage();
  };
```

# 访问权限

## 成员函数

在一个类的内部声明的函数称为该类的成员函数。

成员函数可访问类中的任何成员，包括`private`成员。

## 成员变量

如果一个变量是类的成分（elements），那么该变量称为成员变量（member variables）、数据成员（data members）或者实例变量（instance variables）。

☞ 类中任何成员都可以访问任何成员。

# 类成员需要满足的条件

- 非static的成员变量不能被初始化。
- 不允许有任何成员是正在被定义的该类的对象。
- 不允许有任何成员被定义成auto，extern或者 register。

# 使用建议

所有的数据成员都被定为`private`。

一个需要在全局范围内被频繁访问的变量应该被定为`public`。

`public`变量可以在程序的任何部分被直接访问。

访问`public`变量的方式——对象名 + 点运算符 + 变量名

```cpp
#include <iostream>

using namespace std;

class MyClass
{
    public:
        int i, j, k; // 在整个程序中可被访问
};

int main()
{
    MyClass a, b;
    a.i = 100; // 访问i, j, k是可以的
    a.j = 4;
    a.k = a.i * a.j;
    b.k = 12; // 记住: a.k 和 b.k 是不同的
    cout << a.k << " " << b.k;
    return 0;
}
```

# 结构体与类

class与struct在语法上相似。

C++ 中的 struct 已经被扩展为指定一个类的另一种方式。

class和struct唯一的区别是：

- struct中的成员默认是public
- 而class中的默认为private。

一个结构体定义了一个类。

```
1  // 使用结构体来定义类
2  #include <iostream>
3  #include <cstring>
4
5  using namespace std;
6
7  struct MyStr {
8      void buildstr(char *s); // 公有
9      void showstr();
10 private: // 现在是私有
11     char str[255];
12 } ;
13
14 void MyStr::buildstr(char *s)
15 {
16     if(!*s) *str = '\0'; // 初始化字符串
17     else strcat(str, s);
18 }
19
20 void MyStr::showstr()
21 {
```

```
22      cout << str << "\n";
23  }
24  int main()
25  {
26      MyStr s;
27      s.buildstr(""); // 初始化
28      s.buildstr("Hello ");
29      s.buildstr("there!");
30      s.showstr();
31      return 0;
32  }
33  /*示例输出
34  Hello there!
35  */
```

## 另一种写法

```
class MyStr {
    char str[255];
  public:
    void buildstr(char *s); // 公有
    void showstr();
} ;
```

# 为何有 struct 和 class 两个几乎等价的关键字

- C 已经提供了`struct`用于把数据结合起来，C++ 允许它有成员函数只是一小步
- 有利于把现有的 C 程序移植到 C++
- 保持对 C 兼容的同时，允许`class`继续演变

## 建议

- 需要用类的时候使用`class`，
- 需要用 C 形式的结构体时使用`struct`