清華大學
Tsinghua University

第9章 访问控制与函数重载

清华大学计算机系 韩文弢

教学内容

1. 访问控制
2. 函数重载
3. 存储管理

1、访问控制

类的接口与实现

- 抽象与封装
  - ☺ 将类的本质行为和它的具体实现分开
  - ☺ 外部观点：类的对外接口
  - ☺ 内部观点：类的具体实现

public: 任何地方都可访问（的成员变量和方法）
protected: 类及其子类的函数可访问，但对象.成员不行
private: 只能在类内部访问
未指定(默认): 等价于**private**

抽象

从外部视角来观察

清華大學 Tsinghua University

银行账户


在线客服
收件箱
安全提示
ICBC 中国工商银行
个人网上银行
金融@家
[redacted] 您好 地区：上海
系统公告：T行推出iPhone手机银行服务
搜索
现在时间
欢迎页面 我的账户 定期存款 通知存款 公益捐款 转账汇款 e卡服务 网上贷款 网上挂失 养老金 在线财务管理
T行理财 网上汇市 网上基金 网上国债 网上贵金属 银证业务 个人外汇业务 网上期货 网上保险 跨国理财 银商银权转账
网上预约 缴费站 信用卡服务 金融超市 网上商城 工银信使 电子银行注册 银行卡服务 安全中心 客户服务 分行特色
银商银权转账
交易市场产品
我的交易市场
查询出入金状态
银行端余额如下：
银行结算账户：[redacted]
户名：[redacted]
银行结算账户余额：1,879.39元
最新管理账户余额：2,000.00元
银行控制的可取余额：-
交易市场端余额如下：
交易市场名称：上海航运运价交易有限公司
会员交易资金账号：[redacted]
总余额：101,000.00元
可用余额：101,000.00元
可取余额：101,000.00元

BankAccount类

```
class BankAccount{
public:
    string number;          //帐号
    double balance;         //余额
    string password;        //密码
    ......
};
```

直接访问成员变量

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    BankAccount account;
    account.balance = 1000000;
    cout << "passwaord is: " <<
        account.password << endl;
    return 0;
}
```

BankAccount类

```
class BankAccount{
private:
    string number;              //帐号
    double balance;             //余额
    string password;            //密码
public:
    void deposit(double money){
        balance += money;
    }
    void withdraw(double money){
        balance -= money;
    }
    void resetPassword(string pwd){
        password = pwd;
    }
};
```

getXXX()：查询类

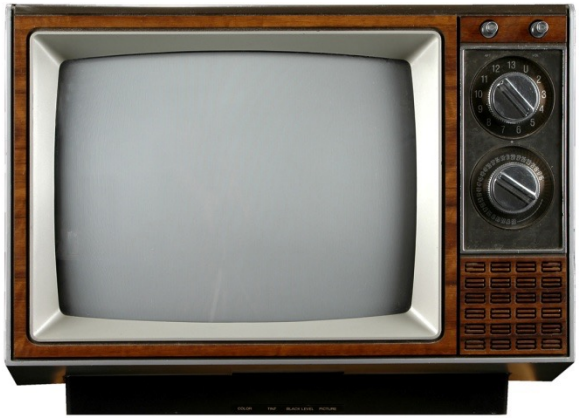
setXXX()：修改类

不能直接访问

```
int main( )
{
    BankAccount account;
    account.deposit(1000000);
    string pwd = "abc123";
    account.resetPassword(pwd);
    return 0;
}
```

抽象与封装

访问控制的另一个目的：抽象与封装。


TV类

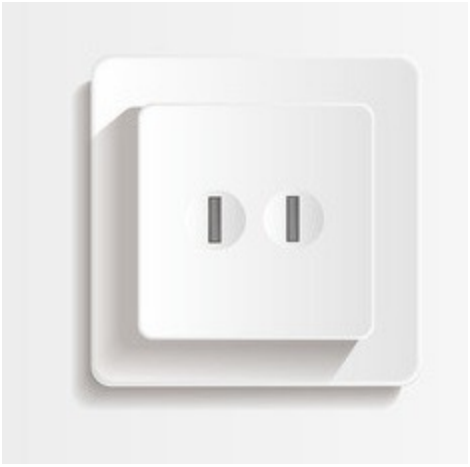
```
class TV{
private:     // private members, 内部实现
    ......

public:     // public methods, 对外接口
    void PowerOn() {...}
    void PowerOff() {...}
    void changeChannel(int channel){...}
    void increaseChannel(){...}
    void decreaseChannel(){...}
    void increaseVolume(){...}
    void decreaseVolume(){...}
    ......
}
```

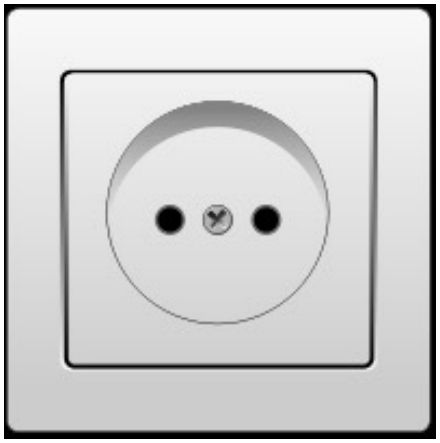
教学内容

1. 访问控制
2. 函数重载
3. 存储管理

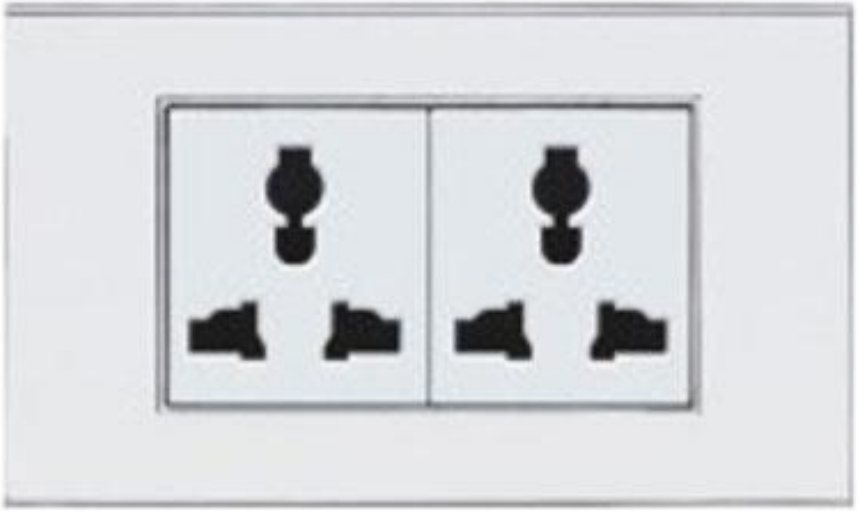
函数重载(overload)

解决方案？





解决方案！



其他类型转换为字符串

C语言的实现方法：

```
//把一个整数value转换为字符串string
char *itoa( int value, char *string,int radix);

//把一个实数value转换为字符串
char *fcvt(double value, int ndigit,
           int *decpt, int *sign);

//把一个无符号长整型数value转换为任意进制的字符串
char *ultoa(unsigned long value, char *string,
            int radix);
```

C++11的实现方法

std::to_string(int i) : 将 int 变量 i 转换成字符串

std::to_string(unsigned u) : 将 unsigned 变量 u 转换成字符串

std::to_string(long l) : 将 long 变量 l 转换成字符串

std::to_string(unsigned long l):将无符号long变量转换成字符串

std::to_string(float f) : 将 float 变量 f 转换成字符串

std::to_string(double d) : 将 double 变量 d 转换成字符串

小狗阿黄

```
class Dog {
public:
    void bark() {
        cout << "汪汪汪!" << endl;
    }
    void bark(bool injured) {
        if(injured) {
            cout << "呜咽..." << endl;
        }
    }
```

小狗阿黄(2)

```
    void bark(int mood){
        if(mood == 0)              // normal
            cout << "汪汪汪！" << endl;
        else if(mood == 1)         //happy
            cout << "旺！旺旺！" << endl;
        else if(mood == 2)         //sad
            cout << "呜—呜" << endl;
    }
};
void main(){
    Dog ahuang;
    ahuang.bark();
    ahuang.bark(true);
    ahuang.bark(1);
}
```

重载的条件(1)

- 成员方法重名，但参数个数不同，或者参数的类型不同，或者顺序不同
- 情形1：参数个数不同

```
class Calculation{
public:
    void sum(int a,int b) { cout << (a+b); }
    void sum(int a,int b,int c){ cout << (a+b+c); }
};
void main( ){
    Calculation obj;
    obj.sum(10, 10, 10);
    obj.sum(20, 20);
}
```

重载的条件(2)

- 情形2：参数的数据类型不同

```
class Calculation2{
public:
    void sum(int a,int b) { cout << (a+b); }
    void sum(double a, double b){ cout << (a+b); }
};

void main(String args[]){
    Calculation2 obj;
    obj.sum(10.5, 10.5);
    obj.sum(20, 20);
}
```

重载的条件(3)

- 情形3：参数的数据类型的顺序不同

```
class Calculation3{
public:
    void sum(int a, double b){ cout << (a+b); }
    void sum(double b, int a){ cout << (a+b); }
};

void main(String args[]){
    Calculation3 obj;
    obj.sum(10, 10.5);
    obj.sum(20.5, 20);
}
```

重载的条件(4)

- 若参数个数相同、各个参数的数据类型和顺序也相同，但变量名不同？

```
int move(string snake);

int move(string turtle);
```

教学内容

1. 访问控制
2. 函数重载
3. 存储管理

存储管理

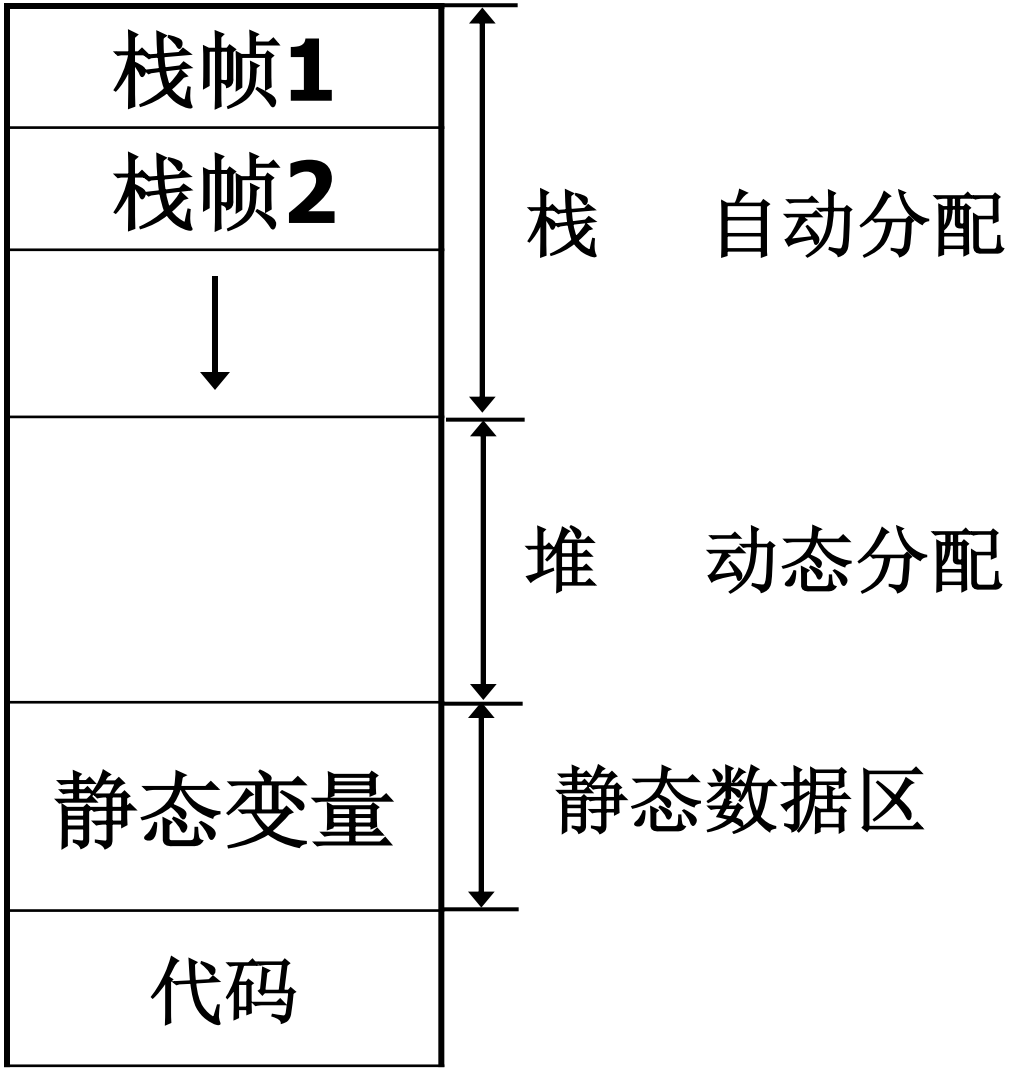
思考：

当一个C++程序在运行时，其数据是存放在哪儿？作用范围？生存期？

寄存器、 栈、 堆、静态数据区

stack heap

内存分布



数据存储分析(1)

```
int global = 0;
int main( )
{
    int *p;
    if(true) {
        int x = 5;
        p = &x;
    }
    cout << *p << endl;   // ???
}
```

	存储区域	作用范围	生存期
全局变量	静态数据区	源文件	程序运行
局部变量	栈	函数内部	函数调用时
块变量	栈	块内声明后	块执行时

数据存储分析(2)

```
int *getPtrToFive() {
    int *x = new int;
    *x = 5;
    return x;
}
void main( )
{
    int *p;
    for(int i = 0; i < 3; ++i) {
        p = getPtrToFive();
        cout << *p << endl;
        delete p;
    }
}
```

动态变量位于堆，整个程序可用、须主动释放

数组空间分配

数组的存储区域、作用范围和生存期与普通变量类似

```
int n;
cin >> n;
int a[n]; //错
```

静态数组、位于栈或静态区，有作用范围和生存期，数组长度必须是常量

```
int n, *a;
cin >> n;
a = new int[n];
...
delete[] a;
```

动态数组、位于堆，作用范围更广，释放前一直可用，长度可是变量

类对象的空间分配

局部、全局和动态类对象（成员变量）的存储区域、作用范围和生存期与普通的变量是类似的。

数据存储分析(3)

```
class Point {
public:
    int x, y;
};
Point r;
void main( )
{
    Point q;
    Point *p = new Point();
    delete p;
}
```

析构函数

```
// 当对象被释放时，析构函数被自动调用
class Point{
public:
    int x, y;
    Point() {
        cout << "构造函数被调用" << endl;
    }
    ~Point() {
        cout << "析构函数被调用" << endl;
    }
};
```

析构函数举例

```
// 对于堆对象，即delete被调用；对于栈对象，
// 即超出范围
void main()
{
    Point *p = new Point;
    delete p;
    if(true){
        Point x;
    }
    cout << "x 超出范围" << endl;
}
```

数据存储分析(4)

```
class Counter {
public:
    int myCount;
    static int ourCount;
    Counter() : myCount(0) {}
    void increment(){
        myCount ++;
        ourCount ++;
    }
};
int Counter::ourCount = 0;
```

数据存储分析(cont.)

```
int main( )
{
    Counter *counter1 = new Counter();
    Counter *counter2 = new Counter();
    Counter counter3;
    counter1->increment();
    counter2->increment();
    counter3.increment();
    cout << Counter::ourCount << ' '
         << counter1->myCount << ' '
         << counter2->myCount << ' '
         << counter3.myCount << endl;
    delete counter1;
    delete counter2;
}
```

存储管理小结

- 形参和局部变量（包括基本数据类型和对象）是存放在栈中，系统自动分配和释放
- 所有的动态数据（即new创建的普通变量和对象）都存放在堆中，用户申请，运行时创建，用delete来释放
- 全局变量和static类变量存放在静态数据区，程序运行时一直存在

‘this’关键词

```
class Banana {
public:
    void f(int i) {
        /* ... */
    }
}
Banana *a = new Banana();
Banana *b = new Banana();
a->f(1);
b->f(2);
```

f函数知道谁在调用自己吗？

‘this’关键词

编译器会偷偷塞进一个参数！

```
Banana.f(a, 1);
Banana.f(b, 2);
```

在此情形下，在成员方法内部，若想得到当前对象的引用，可以使用this（当前对象）。

```
class Banana {
public:
    double price;
    void setPrice(int price) {
        this.price = price;
    }
}
```

本讲小结

- 访问控制
- 函数重载
- 存储管理