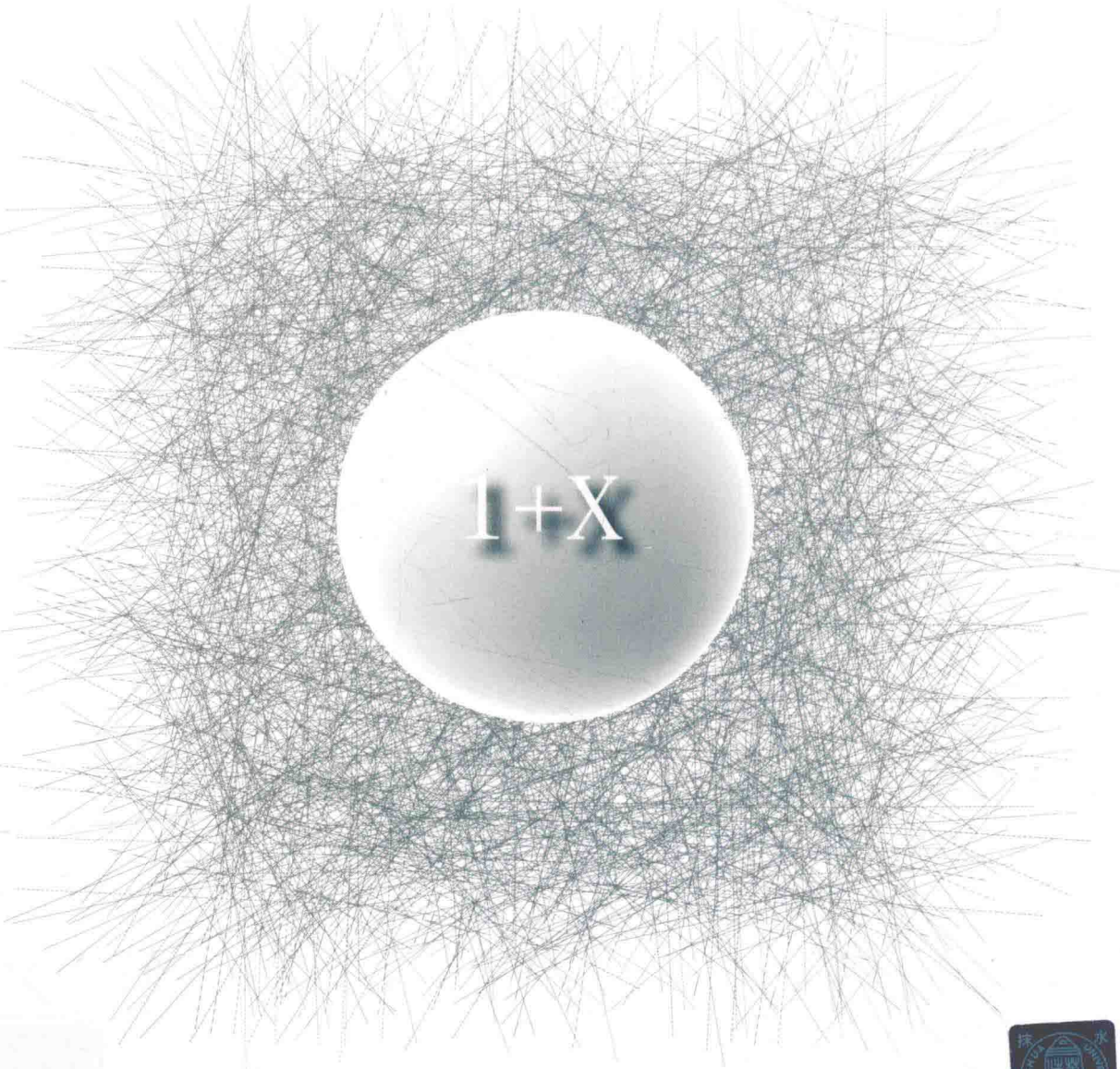
大学计算机基础教育规划教材

Qt图形界面编程入门

仇国巍 编著

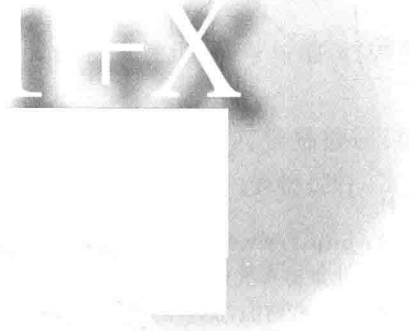


清华大学出版社

大学计算机基础教育规划教材

Qt图形界面编程入门

仇国巍 编著



清华大学出版社
北京

内 容 简 介

本书着重讲解利用 Qt 开发图形界面程序的基础知识。全书共 10 章，主要内容包括 C++ 语言中面向对象的知识、集成开发环境 Qt Creator、基本窗体控件、菜单和工具栏、对话框、界面布局、事件系统、二维绘图、样式表等方面的内容。基本覆盖了利用 C++ 语言在 Qt 开发平台下开发窗口界面的知识。第 10 章给出 3 个比较大的范例，建议先自己思考并编写程序，而后和本书例程对照，从而更有效地提高编程水平。本书讲述力求简单实用、步骤详尽，非常适合课堂讲解少而练习时间多的授课方式，也适合于在翻转式教学模式下引导学生自我学习。本书要求读者具有 C 语言编程基础，在此基础上即可顺利地学习本书内容。建议共安排 48 学时，其中，24 学时授课，24 学时上机练习。

本书适合作为高校相关专业教材，也可供软件开发人员自学参考。

图书在版编目(CIP)数据

Qt 图形界面编程入门/仇国巍编著. —北京：清华大学出版社，2017
(大学计算机基础教育规划教材)
ISBN 978-7-302-46063-3

Ⅰ. ①Q… Ⅱ. ①仇… Ⅲ. ①软件工具－程序设计－高等学校－教材 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字(2017)第 004889 号

责任编辑：张　民　战晓雷
封面设计：何凤霞
责任校对：白　蕾
责任印制：何　芊

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62795954

印 刷 者：三河市君旺印务有限公司
装 订 者：三河市新茂装订有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　张**：18　　**字　数**：415 千字
版　　次：2017 年 5 月第 1 版　　**印　次**：2017 年 5 月第 1 次印刷
印　　数：1～2000
定　　价：39.50 元

产品编号：070540-01

前 言

Qt图形界面编程入门

Qt是基于C++语言的著名的跨平台开发框架，自20世纪90年代出现以后，不断发展壮大，现在已经发展成为强大的、几乎全功能的开发框架。它不仅可以用于开发用户界面，还可以进行数据库、网络、多媒体、嵌入式等方面的编程开发，但是Qt最侧重的，历史最悠久的仍然是GUI图形界面开发。Qt开发的程序可以运行于Windows、Linux、UNIX等主流操作系统，只要没有调用专属于某个操作系统的功能，Qt开发的源程序一般不用修改，只需将它的源码在不同的操作系统下编译后即可执行，真正达到了“一次编写，处处编译”的境界。

全书共分10章。

第1章和第2章讲述C++语言面向对象的基础知识。因为本书假定读者了解C语言的编程基础，所以这里用两章的篇幅介绍面向对象的知识，包括类和对象、类的继承和多态等方面的知识。

第3章介绍Qt的安装、Qt Creator的基本使用、信号与槽通信机制，以及编程中常用的几个基本字符串类。

第4章讲解基础窗口类以及各种常用的界面控件，包括按钮、标签、单选按钮、检查框、组合框、列表框、编辑框、进度条、选项卡、树状控件、表格控件、富文本控件等。这些控件可以方便地构造图形界面。

第5章讲解菜单、工具栏和状态栏的基础知识，以及对话框的基础知识。了解手工编程和利用设计器构建菜单的差异，了解模态、非模态对话框的不同之处。

第6章介绍控件布局管理、窗口切分与停靠、单文档与多文档界面的实现方式。有了布局管理的知识，就可以灵活高效地安排控件的位置并使之随界面大小而动态变化。大大简化了界面编程的强度。

第7章介绍事件系统的基本知识。窗体程序的一举一动全由事件驱动，鼠标操作、键盘操作、定时发生的动作、界面重绘等全是事件，有了事件概念并且适当地利用事件处理机制编写程序是界面编程的要点之一。

第8章讲解二维绘图系统。画笔、画刷的利用和图形绘制是界面编程中不可或缺的内容，坐标变换和特殊填充方式体现了Qt二维绘图功能的强大。

第9章介绍利用样式表美化界面的方法。在Qt中利用类似于网页CSS脚本的QSS脚本可以直接设定各种控件的大小、颜色、背景等属性，极大地方便了界面的美化。

第10章给出3个编程实例——接金币、俄罗斯方块、游戏大厅界面。通过这些范例

让读者进一步了解界面编程所需要的综合能力。

由于本书内容广泛，加上编写时间仓促，以及作者水平有限，书中可能有错误及不合理之处，恳请读者指正。

仇国巍
qwqiu@mail.xjtu.edu.cn
2017年1月

目 录

第1章

类和对象

1.1 面向对象程序设计

在20世纪90年代以前，程序设计的主流方法是面向过程，如C语言等当时流行的都是面向过程程序设计的编程语言。从20世纪90年代开始，面向对象的程序设计方法逐渐流行，并成为程序设计的主流方法，而C++语言既可用于面向过程的程序设计，也可用于面向对象的程序设计，这使得C++语言成为传统开发方法和现代开发方法之间的桥梁。可以说在习惯于传统开发方法的老一代程序员熟悉并掌握面向对象开发方法的过程中，C++语言发挥了关键性作用。面向对象方法成为主流并不意味着面向过程方法的彻底消失，相反，它们相辅相成，各自都发挥着重要作用。

面向过程就是分析出解决问题所需要的步骤，然后用函数把这些步骤一一实现，使用的时候一个一个依次调用就可以了。面向对象是把构成问题的事务分解成许多对象，建立对象的目的不是为了完成一个步骤，而是为了描叙某个事物自身及其在整个解决问题的过程中的行为。

面向过程的程序设计是一种自上而下的设计方法，设计者用一个main函数概括出整个应用程序需要做的事。在main函数中对一系列子函数进行调用，其中的每一个子函数还可以再被精炼成更小的函数，这就是模块分解的过程，如图1-1所示。重复这个过程，就可以完成一个过程式的设计。其特征是以函数为中心，用函数作为划分程序的基本单位，数据在过程式设计中往往处于从属的位置。

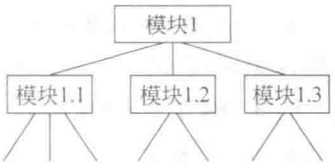


图1-1 模块分解

面向过程设计的优点是易于理解和掌握，这种逐步细化问题的设计方法和大多数人的思维方式比较接近。然而，过程式设计对于比较复杂的问题，或是在开发中需求变化比较多的时候，往往显得力不从心。这是因为过程式的设计是自上而下的，这要求设计者在一开始就要对需要解决的问题有一定的了解。在问题比较复杂的时候，要做到这一点会比较困难，而当开发中需求发生变化的时候，以前对问题的理解也许会变得不再适用。事实上，开发一个系统的过程往往也是对系统不断了解和学习的过程，而过程式的设计方法忽略了这一点。

在面向过程式设计的语言中，一般都既有定义数据的元素，如C语言中的结构，也有

定义操作的元素，如C语言中的函数。这样做的结果是数据和操作被分离开，容易导致对一种数据的操作分布在整个程序的各个角落，而一个操作也可能会用到很多种数据，在这种情况下，对数据和操作的任何一部分进行修改都会变得很困难。

面向对象是一种自下而上的程序设计方法。不像过程式设计那样一开始就要用main函数概括出整个程序，面向对象设计往往从问题的一部分着手，分析问题包含哪些对象，将这些对象抽象成为类的形式，一点一点地构建出整个程序。类成为模块化的元素，是划分程序的基本单位。在面向对象设计中，类封装了数据，而类的成员函数作为其对外的接口，抽象地描述了类。用类将数据和操作这些数据的函数放在一起，可以说这就是面向对象设计方法最基本的特征。

下面的例子概要地说明了面向过程和面向对象程序设计的区别。

问题：现在要将一批鸭子用若干个盒子包装起来，请将此过程用程序表现出来。

方法一：面向过程的分析设计。

把鸭子装进包装盒的过程主要包括：打开盒子，装进鸭子，盖好盒盖。每个步骤都可以编写一个函数，于是得到3个函数：

- Open()，把盒子打开，目标是得到打开的盒子。
- Put()，把鸭子装进去，目标是得到里面装着鸭子的盒子。
- Close()，把盒子盖好，目标是获得盖好盒盖的有鸭子的盒子。

以上3个过程每个过程都有一个阶段性的目标，依次完成这些过程，就能把鸭子装进盒子。在上面的函数中，参数都没有确定，因为这个问题用什么样的数据形式表达尚未确定。可以尝试用下面的数据表示问题：

用数组state[]表示盒子的状态，state[k]表示第k个盒子的情况。可将盒子的状态分为4种情况：

0——未开盖的空盒。

1——开盖的空盒。

2——装好鸭子，未盖好盒盖的盒子。

3——装好鸭子，盖好盒盖的盒子。

于是，state[]定义为int型数组最合适。

同样，鸭子的情况也应该用数组表示。这里用duckState[]表示鸭子的状态，duckState[n]表示第n只鸭子的情况。鸭子的状态是“未被装入”和“已经装入”两种，因此可以用bool型数组表示。于是前面几个函数可定义如下：

```
//打开第 k 个盒子，成功则返回盒子编号，失败返回-1
int Open(int state[], int k)
//将第 n 只鸭子放入第 k 个盒子，成功则返回盒子编号，失败返回-1
int Put(int state[],int k,bool duckState[], n)
//将第 k 个盒子盖好
void Close(int state[], int k)
```

至此，3个步骤都用函数实现了。看起来还不错。下面看看面向对象的方法。

方法二：面向对象的分析设计。

面向对象的方法不是从“怎么做”出发，而是从“是什么”出发考虑问题。首先寻找问题中的对象。为了发现对象，要在系统说明文档中查找名词和名词短语，包括物品、可感知的事物(压力、温度等)、角色(母亲、教师、政治家)、事件(着陆、中断、请求)、相互作用(借贷、开会)、人员、场所、组织、设备、地点等。通过浏览问题的描述发现重要的对象和其责任是面向对象分析和设计初期的重要任务。

在上述问题中，可以找到的重要名词有盒子、鸭子。然后需要将这些对象的共有特性以及其责任(或者说具有的方法)抽象为特殊的数据类型——类。在C++中，类用class定义。

可以看出，盒子类具有状态属性，同时可以把“将鸭子放入盒子中”归纳为盒子的任务。于是为实现这一任务，盒子类应该具有的方法有：打开自己的盖子，将鸭子放入自己内部，合上自己的盖子。为什么要加上“自己”两个字呢？因为一个盒子没有必要管别人的事情，它可以请求别人做一件事，但不应该越俎代庖，自己替别人做。于是盒子类可定义如下：

```
class BOX
{
    int ID;                              //盒子编号，为什么不是数组？
    int state;                           //盒子状态，为什么不是数组？
    int Open();                          //打开盒子，为什么没有参数？
    int Put(bool duckState[], n);        //将第 n 只鸭子放入盒子
    void Close();                        //将盒子盖好，为什么没有参数？
}
```

首先，盒子有许多个，那么盒子类当中为什么ID和state不是数组呢？因为，这里抽象出来的类是盒子类，而不是“盒子群组”类。这个类表达的是每一个个体的特征。那么，如何表达若干个盒子呢？其实C++中的类是一种数据类型，只不过比较复杂而已。既然是数据类型，就可以定义数组，定义一个BOX类型的数组就可表达一组盒子了。

另外，为什么在Open和Close等方法中没有代表盒子标号的参数呢？因为这里的方法是每个对象(类的一个实际个体)自己具有的方法，合理的设定是：盒子可以打开自己的盖子，即修改自己的state。C++的中类的定义规定，每个对象的函数可以直接操作自己的属性(即数据)，也就是说Open等函数可以在函数内部直接修改自身的state内容，而不必将state作为参数传入Open函数。除非每个盒子都需要打开别人的盖子(多么奇怪啊)，那么则可以将另一个盒子对象作为参数传入Open函数，可以定义为

```
int Open(BOX abox);                      //打开盒子 abox 的盖子(不一定是自己)
```

实际上，类似“每个盒子都需要打开别人的盖子”这种情况是很少见的，因为不太合理。

为了把鸭子装进盒子，需要做3个动作：打开盖子，装入鸭子，合上盖子。每个动作有一个执行者，它就是对象。我们要做的是对某个盒子对象发出指令：

(1) k号盒子，请把盖子打开。

(2) k 号盒子,请把第 n 个鸭子装进去。

(3) k 号盒子,请把盖子盖上。

这时盒子 b[k]应当执行的代码如下:

```
b[k].Open()
b[k].Put(duckState, n);
b[k].Close()
```

显然面向对象的编码含义更加清晰。

比较上面两种编程方法,可以看出:

(1) 在面向过程的方法中,数据和函数是分离的,数据可能散落在程序的各个角落,不易维护;而在面向对象的方法中,数据和函数结合在一起,每个对象都有自己独立的数据,含义清晰,易于维护。

(2) 在面向过程的方法中,需要在开始阶段就分析清楚每个过程的作用,并编写合适的函数,一旦在后续工作中发现初始的分析不恰当,则可能改动很大;而面向对象的设计方法允许开发者从问题的局部开始,先找出类,再找出类之间的关系,在开发过程中逐步加深对系统的理解,最终利用类构造出整个系统。对于比较大的软件系统,面向对象的方法在分析阶段、维护阶段都更有优势。

1.2 类的声明和对象创建

类(class)是现实世界或思维世界中的实体在计算机中的反映,它将数据以及这些数据上的操作封装在一起。对象是具有类类型的变量。类是对象的抽象,而对象是类的具体实例(instance),即类的实例化。类是抽象的,不占用内存,而对象是具体的,占用内存。

1.2.1 如何声明一个类

声明一个类的一般形式为

```
class 类名              //class 是声明类的关键字
{
    private:
        ⋮
        //私有成员数据(属性)和私有成员函数(方法),不能在类外直接访问
    protected:
        ⋮
        //保护成员数据(属性)和保护成员函数(方法),也不能在类外直接访问
    public:
        ⋮
        //公有成员数据(属性)和公有成员函数(方法),能在类外直接访问
};      //用一对花括号括起来的部分是类体。类声明以分号结尾
```

私有成员、保护成员、公有成员限定了对有关成员的访问方式。简单讲就是对公有成

员的访问限制最宽松，对私有成员的访问限制最严格。它们的差别将在后面章节详细介绍。

这里的成员既包括类的属性，即成员变量，也包括类的方法，即成员函数。成员函数的实现可以在类的声明内部完成，也可以在外部。请看下面的例子，这里先全部使用公有成员变量和成员函数。

```
//声明一个银行账户类
class Account
{
public:                                    //公有成员
    int ID;                                //账户 ID
    char Name[20];                         //姓名
    float balance;                         //余额
    int withdraw(float m)                  //取出数量为 m 的钱，返回-1 表示失败
    {
        if(balance>m) { balance=balance -m; return 1; }  //正常，返回 1
        else return -1;                                  //失败，返回-1
    }
    void deposits(float m);                              //存入数量为 m 的钱
};                                                       //这里一定要有分号
void Account::deposits(float m)
{
    balance=balance+m;
}
```

在上面的类中，取款函数 withdraw 定义在类的内部，这种情况称为内联函数。而存款函数 deposits 定义在类的外部。

在类的外部定义成员函数的格式为

```
返回类型 类名::函数名(形式参数)
{
    函数体
}
```

这里的符号::表示该函数属于哪个类。如果没有“类名::”部分，则该函数不属于任何类，也就成了一个 C 语言意义下的函数，而且是全局函数（即在该文件任何位置均可使用该函数）。

1.2.2 定义和使用对象

如果已经声明了一个类，就可以用它定义变量了。使用类定义的变量一般称为对象。定义对象的方法和使用基本变量类型定义变量的方法一样。例如，下面的语句定义了一个账户对象：

```
Account myAccount;
```

而下面语句则定义了包含 10 个对象的数组：

```
Account account[10];
```

不论是定义了一个对象还是一组对象，C++ 都会在内存中为每一个对象分配必要的空间。如果为上面定义的 myAccount 分配空间，则内存空间需要容纳账户 ID(整型)、姓名(字符数组)、余额(浮点数类型)。每个账户对象都是不同的，因此每个对象都要有存放账户 ID、姓名和余额的空间。

另外，Account 类有两个成员函数 withdraw 和 deposits，而 myAccount 作为 Account 类的具体成员，当然可以使用这两个成员函数。但是成员函数是被各个对象共用的，因此成员函数并不占据一个具体对象的内存空间，这一点和数据成员处理方式不同。反过来想，如果每个对象都存储一个成员函数的副本，既不经济，效率也不高。

类对象使用公有函数的方法是

对象名.函数名(实际参数)

例如，要向 myAccount 存入 600 元，格式如下：

```
myAccount.deposits(600.0);                    //仅对公有函数可以这样使用
```

同时，类对象的公有变量也可以直接修改或显示。例如：

```
myAccount.ID=10001;                           //设定账户 ID
```

同类对象之间可以赋值，例如下面的语句：

```
myAccount=account[0];
```

在赋值过程中，右侧 account[0]的每个属性的内容被复制到 myAccount 对象的相应内存区域，这和 struct 结构是一样的。

类也可以作为一个函数的参数或返回值出现，下面定义一个不属于任何类的函数：

```
//从账户 A 转 m 元到账户 B
bool transfer(Account A, Account B, float m)
{
    if(A.withdraw(m)==1)                      //若 A 取款成功
    {
        B.deposits(m);                        //向 B 存款
        return true;
    }else
    return false;
}
```

这是一个转账函数。对于这个函数，编译器不会报错。但是这个函数很可能无法实现预定的目标。例如，如果按下面的方式调用此函数：

```
transfer(myAccount, account[0], 1000.0);
```

开发人员希望通过此函数将 1000 元从账户 myAccount 转到账户 account[0]。上面的调用将以下面的方式进行：

(1) 调用函数时产生 A 和 B 两个对象。虽然 A 和 B 仅出现在函数列表中，没有在 transfer 的函数体内定义，但 A 和 B 仍然是此函数内的局部变量。

(2) 对象 myAccount 和 account[0]分别赋值给 A 和 B。这里 A 和 myAccount、B 和 account[0]都是彼此独立的对象。

(3) 在函数中从账户 A 转 m 元到账户 B。而 myAccount 和 account[0]无任何变化。

(4) 函数 transfer 执行完毕。transfer 中的局部变量 A 和 B 也随之消失。

上面的执行过程没有完成从账户 myAccount 到账户 account[0]的转账，根本原因是调用函数时参数采用的是值传递。如果要真正实现转账，应当采用指针传递或者引用传递。指针传递的概念和 C 语言中的一致，而引用传递则是 C++ 特有的参数传递方式。

1.2.3　对象的指针和引用

1. 指针和引用的区别

指针是 C 语言的概念，代表内存地址，一个变量的起始地址就是该变量的指针，当然在 C++ 中也可以用指针。引用是 C++ 语言的专有概念，它是另一个变量的别名，它和该变量绑定在一起。

指针和引用有一个共同点：它们都代表某个变量占据的某一块内存区域，通过指针或引用都可以对它们代表的变量进行操作。下面说明它们的编程方式。

以下的语句利用指针对变量赋值：

```
int m;
int *p;                    //指针定义
p=&m;                      //将 m 的内存地址赋给 p
*p=5;                      //通过指针对变量 m 赋值
```

而下面的语句则是利用引用做同样的事情：

```
int m;
int &q=m;                  //定义引用 q 并与 m 绑定
q=5;
```

注意上面两段代码中 & 符号的使用。& 放在等号右侧的某个变量前一般是取该变量的地址；而在定义变量时，在变量前出现的 & 符号表示该变量是引用。

指针和引用主要有下列不同：

(1) 引用只能在定义时被初始化一次，之后不可变；而指针可变。

例如语句 int &q = m 是正确的，但是下面的写法不对：

```
int &q;
q=m
```

而且 q 一旦和 m 绑定，就不可改变。