第 1 章　C++语言与面向对象程序设计概述

C++语言是以 C 语言为基础发展起来的面向对象程序设计语言，但二者是相互独立的。本章简要说明 C++语言的发展历程，以及 C++程序的一般结构和主要成分。同时，概括说明面向对象程序设计的思想、问题分析和基本设计方法，以期使读者建立起面向对象程序设计的初步印象。

1.1　C++语言概述

C++是一种以 C 语言为基础开发的高级语言，保留了 C 作为它的一个子集。与 C 语言的面向系统和底层程序开发的目的不同，C++的设计目标是面向大型应用程序的开发与设计。虽然 C++基本包含了 C 语言的所有主要特性，但更趋向于对数据结构的表述，这使得 C++的核心由支持过程化程序设计转向以类为基础的面向对象程序设计，这种转变导致了程序分析、设计方法的重大改变。

考虑到目前多数读者都有学习 C 语言的经历，而且 C++本身所包含的内容、技术较多，本书的写作是以假设读者具有一些 C 语言编程知识为前提的，同时也有选择地忽略或淡化了某些不太常用或复杂的内容，以使其易于作为教材使用。

1.1.1　标准 C++语言的产生与发展

C++的发明者是 AT&T 贝尔实验室的 Bjarne Stroustrup 博士，他给出了 C++的第一个定义。首先，C++保留了 C 作为一个子集，使其具有 C 语言的处理复杂底层系统程序设计工作的能力。其次，为了增加面向对象特性，C++从 Simula 语言引入了类的概念，包括派生类和虚函数。此外，C++还借鉴了 Algol 语言的运算符重载等特性，形成了 C++的早期版本，被称为“带类的 C”，这种转变是因为引入事件驱动机制所导致的，对面向对象的支持还不够完善。早期的 C++编译系统只是一个将 C++代码翻译成 C 代码的预编译系统。

随后，C++的语法经过了若干次审查和修订，主要包括对重载的解析、连接以及存储管理，并在 static 成员函数、const 成员函数、protected 成员、多重继承、模板、异常处理、运行时类型识别和名字空间等方面进行了扩充，还增加了一些重要的数据结构和算法，目的是使 C++程序更容易编写。1988 年诞生了第一个真正的 C++编译系统。对于容器和泛型算法的支持，最初以标准模板库 STL 形式提供，目前也已正式纳入 C++标准，通常称为 C++标准库。同时，C++标准化的工作也在推进。

1991 年，C++标准化 ANSI 委员会的 C++标准化工作正式成为 ISO 的 C++标准化工作的一部分。1998 年，ISO/ANSI C++标准正式通过并发布。鉴于 C++产生和应用的实际状况，C++标准保留了对 C 及早期 C++版本的兼容，利用文件名的不同体现它们之间的差异，并容忍一些早期的语言现象如函数声明等。

总体上，C++是一种混合语言，或者说 C++是一种集过程化设计、面向对象、基于对象和泛型算法等多种技术于一体的编程语言。

本书以标准 C++为基准，重点介绍语言对面向对象程序设计的支撑技术，以及面向对象程序的分析和设计方法，体现在对数据结构（类型）以及建立在此基础上的程序处理、组织技术。在学习 C++语言时，最重要的是集中关注概念和思想，不要迷失在语言的技术细节中，因为对于程序设计内涵和设计技术的理解远比对细节的理解更重要。

1.1.2 编写简单的 C++语言程序

最简单的 C++程序由如下代码给出，它不做任何工作，仅是一个程序框架：

```
int  main( )                              //主函数定义
{
  return  0;
}
```

与 C 程序类似，一个完整的 C++程序必须且只能有一个名字为 main 的函数，称作“主函数”，而这个面向过程的主函数是使 C++被称为混合语言的象征。但 C++的程序只依靠 main 函数作为程序入口，供操作系统调用，其他成分以类而不是函数形式构成。main 函数负责组织、协调类的对象共同协作而不是函数调用来完成既定任务。

在细节上，C++的 main 函数的声明类型为 int 而非 C 语言常见的 void，此时，需要在函数末尾加上“return 0;”语句。

这里给出了一个简单的 C++程序示例，功能是计算两个字符串的“和”并将结果输出到屏幕上。

```
/* 示例程序 Example1_1.cpp */
#include <iostream>                       //头文件包含
#include <string>
using namespace std;                      //名字空间声明
int  main( )
{
  string  x("Hello"), y("world.");        //对象生成（变量定义）
  string  z;
  z = x + " " + y;                        //消息驱动（运算）
  cout << z << endl;                      //消息驱动（输出）
  return  0;
}
```

程序运行时的输出结果为：

```
Hello world.
```

1．程序结构

在整体框架上，C++程序的外观结构与 C 较为类似，主要包括预处理指令部分、声明部分和定义部分，包括 main 函数定义。

预处理指令用于指定编译器的某些操作，包括文件包含、宏定义和条件编译等。声明部分包括函数声明、类型声明和变量声明等，它们存在的主要目的是供编译器进行语法检查，以发现设计中存在的错误。函数定义是指规定函数的格式和要执行的代码。这些代码放在一对花括号“{ }”内，称为“函数体”。函数体一般包括变量和对象定义、输入、运算和输出等内容。

2．头文件包含与名字空间

程序中的“#include <iostream>”和“#include <string>”是用于包含头文件的预处理指令。程序包含<iostream>和<string>分别是因为使用的对象 cout、常量 endl 和 string 类型定义在这两个头文件中。这里的 string 是一个 C++的字符串类（类型），用于替代以'\0'结尾的 C 字符串。在没有头文件被包含时，编译器不能识别 cout、endl 和 string 的含义。应注意在指定头文件时没有使用文件扩展名.h 或.hpp。

语句“using namespace std;”的作用是说明使用名字空间 std。这是一种 C++为了减少程序中的名

字冲突而引入的技术。名字空间用于规划程序中的空间范围，不同的定义归属不同的名字空间，程序通过指定名字空间来表明所使用的名字的来源。标准 C++的所有定义都属于名字空间 std。该语句的作用是向编译器表明，以下程序中出现的定义如果不是局部的，应属于名字空间 std。只要使用 C++标准库，就应该说明使用 std 名字空间。

说明一个名字的所属名字空间有几种不同的方式，这里采用了一种统一说明的方法，即将使用名字空间语句“using namespace std;”加在程序开头。当然，也可以不统一说明，在使用每个名字时将 std 直接放在它的前面，如：

```
std::cout << z << std::endl                    //输出
```

这种方式更明确地表明了一个名字的所属关系。

3．注释

程序注释是对程序中的代码、变量等所做的说明，在程序编译时对注释不做任何处理。程序中添加注释有助于提高程序的可读性，是非常必要的部分。在一些特殊程序中，注释可达整个篇幅的 1/3 以上。

C++支持两种注释，其一是继承自 C 的“/* 注释 */”，一般称为“注释对”形式的注释；另一种是由//引导的“//注释”形式。前者是段落形式的注释，可以包含连续的任意多行，可以在任何允许插入空格的地方插入。后者称为“行式注释”，标志着从“//”开始到本行结束的内容均为注释。

示例程序 Example1_1.cpp 中包含了上述两种注释。

通常，可以这样对程序进行注释：

（1）在程序头，用于说明程序名、功能、作者、目的、用途、修改史等；

（2）在函数或方法声明前，用于说明它们的功能、参数、返回值和副作用等；

（3）在变量定义前（或后），说明变量的含义和作用；

（4）在函数体开头，用于解释算法思路；

（5）在类定义前或类定义中添加注释，用于说明类和成员的功能。

限于篇幅，本书通常只在行末添加少量、必要的注释，这些注释以中文形式给出，多用于解释所在行代码的功能、含义和应注意的事项。

注释是程序的重要组成部分，必须养成及时添加注释的习惯，且尽量使用 C++的行式注释，因为//可以嵌套在//中，但/**/不可以嵌套。不过，应注意位于宏末尾的行式注释有可能被看作宏的一部分。**

4．输入/输出对象

程序中通常要从键盘接收用户的输入，并将运算结果输出到显示器上。与 C 语言的函数式输入/输出技术不同，C++借助标准库中定义的对象来实现。输入数据需要使用的对象是 cin，语法形式示例如下：

```
int  x;
double  y;
cin >> x >> y;                                  //也可写成：cin>>x; cin>>y;
```

上述语句可以将用户输入的数据保存到变量 x 和 y 中。

输出数据使用对象 cout 实现，语法形式示例如下：

```
int x = 10;
cout << "x is " << x << '.' << endl; //输出 x is 0.
```

上述语句连续输出 4 部分的值。语句末端的 endl 是一个 std 空间中定义的常量，含义是刷新输出缓冲区，其作用与字符‘\n’基本相同，可以将光标转移到下一行的开头。

5. 编码习惯

C 和 C++的程序都以书写格式自由著称，这是指程序的书写几乎不受什么限制，可以在一行上书写几个语句，或者把一个语句写在多个行上，但正确的做法一般是一个语句占用一行。应该说，尽量保持好的书写风格是必须养成的习惯。因此，应尽量注意程序的书写“格式”，如缩进格式和成对符号的对齐排列等。本书所提供的示例程序将尽量给读者提供一种可借鉴的规范，偶尔（如习题中）可能将相近的代码写在同一行内，这纯粹是出于篇幅的考虑。

程序的良好格式（如一致的缩进编排风格）是使程序具有可读性、使设计者能与他人合作的基础，有时比解决问题本身还重要。

1.2 由过程化到面向对象程序设计

作为一种设计技术，面向对象对软件研发、应用甚至日常生活的影响都是巨大的，且这种作用仍在扩大并发展。互联网上大量的应用和功能也都以对象的方式呈现，我们每天都面对着各种各样的对象，了解面向对象程序设计技术已成为学习软件开发者的必然，这意味着设计思想由面向过程向面向对象的转变。本节仅通过一个简单的应用来比较一下两种技术和思想的差别，具体技术细节将在后文中展开。

1.2.1 过程化程序设计

用计算机解决问题时总要设计程序，即以某种语言为工具编写控制计算机执行的动作序列，每个动作规定了一定的基本操作。自计算机问世以来，人们不断地实践并总结着有效的程序设计方法。当然，这种变革是以简化编程和提高软件生产率为目的的。

由于编程问题起源于最初的科学计算，因此，在相当长的时间里，程序都是围绕着数据的组织与算法（处理、操作）的切分展开的。面对一个具体问题，首先要“建立需求分析和系统规格说明”，即“建立一系列规则，根据它判断任务什么时候完成，以及客户怎样才能满意”。如同现实社会中的一个大型项目或产品开发一样，先将一个很大的“产品”分成适当的部件，再由这些部件组合成整个产品，而程序员的职责就是考虑如何更好更快地实现这些部件，一般称为“过程”或“模块”。这样的处理方式被认为是基于或面向过程的。在这里，数据和算法是问题的中心，程序设计的中心工作是研究数据的描述、存储和数据间的关系，以及作用在数据结构上的算法，每个算法通过一个或几个过程来体现，完整的程序由算法（模块）之间互相调用组成，如图 1-1 所示。此时，人们认为“算法+数据结构＝程序”（N. Wirth 的观点）。

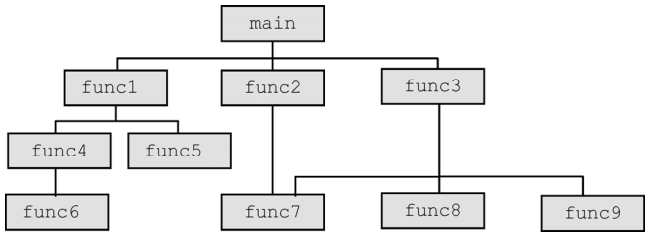

图 1-1 由函数组成的程序

过程化的问题处理思路形成了一套有效的程序设计方法，称为结构化方法，它体现在以下三个方面。

（1）程序设计采用自顶向下、逐步细分的方法展开。

（2）模块化。这是指程序中的过程体和组成部分应以模块表示，模块还可以称为过程、子程序或函数。每个模块应具有较高的独立性，即强内聚性和弱外联性。这里的内聚性是指模块内部功能的单一性，而外联性是指同其他模块的联系。这样做的目的是使对一个模块的修改不致于对其他模块造成太大的影响。

（3）使用三种基本控制结构。这是指描述任何实体的操作序列只需要“顺序、选择、循环”这三种基本流程控制结构，参见图 1-2。从整体上看，模块中的指令（语句）总是由上到下按顺序逐个执行的，但局部可能有选择或循环。三种基本结构的共同特点是每种结构只有一个入口和一个出口，这使程序更容易理解和维护。

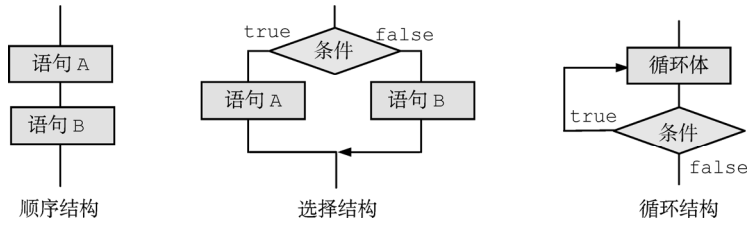

图 1-2　三种基本结构

这里考虑一个五子棋游戏的设计示例。采用过程化方法可以将整个问题按处理过程分解如下：

开始函数 start( )；

绘制棋局画面函数 drawChess（棋局数据）；

走棋函数 play（选手）；

判断输赢函数（棋局数据）；

输出结果函数 judgeWinner（棋局数据）；

结束函数 stop( )。

实现游戏的程序可以按下述简化流程实现：

```
int  main( )
{
   drawChess(初始棋局);              //绘制棋局
   start( );                         //初始化工作，如启动计时器，假定黑棋先走等
   do                                //重复执行下述操作
   {
     play(执黑选手);                 //执黑选手走棋
     drawChess(棋局);                //刷新棋局
     if(judgeWinner(棋局))           //判断，已定出输赢
       break;                        //停止重复
     play(执白选手);                 //执白选手走棋
     drawChess(棋局);                //刷新棋局
     if(judgeWinner(棋局))           //判断，已定出输赢
       break;                        //停止重复
   }while(未超时);                   //达到预定时间时停止重复
   showResult(棋局);                 //输出胜负结果等
   stop( );                          //终止
}
```

由于过程化设计中的数据与过程是分离或者说相互独立的，且数据（如棋局）有时是全局的。一

个过程完全可以作用到并不相关的数据上，也不能保证对数据操作的合理性，数据对于算法（操作）完全是被动的。这种操作是一种“谓语＋宾语”结构。

1.2.2 面向对象的程序设计

面向对象设计方法将客观世界看成是由对象组成的。对象是一种实体，具有自己的属性和行为。一个程序由对象及其相互间的协作来实现。

例如，对于五子棋游戏，完成一次比赛应由两个选手、一个裁判和一个组织者组成。选手只负责走棋，裁判负责判定输赢和确定比赛是否结束，组织者提供场所和设施。这里的每个人是一个对象，对象之间通过消息实现协作。不同对象具有不同的行为。例如，比赛选手负责走棋，裁判负责确定比赛开始、判断输赢、确定比赛结束，而组织者负责绘制画面和输出结果等。为此，首先要定义如下数据结构（类）：

```
class  Player                        //选手描述
{
    string  piece;                   //执黑或执白
  public:
    Player(string  p);               //构造器
    play( );                         //走棋
}
class  Referee                       //裁判描述
{
  public:
    start();
    judgeWinner(棋局);
    judgeTime();                     //判断超时
    stop();
}
class  Playmaker                     //组织者描述
{
    ChessInfo  chessInfo;            //棋局信息
  public:
    drawChess();
    showResult();
    announceInfo();                  //公布棋局
}
```

于是，可以按如下方式组成程序：

```
int  main( )
{
   Player  player1("执黑"), player2("执白");    //初始化，生成对象
   Referee  referee;
   Playmaker  playmaker;
   playmaker.drawChess();
   referee.start( );                 //初始化工作，如启动计时器，假定黑棋先走等
   do
   {
```

```
            player1.play();
            if(referee.judgeWinner(playmaker.announceInfo()))
              break;
            playmaker.drawChess( );
            player2.play();
            if(referee.judgeWinner(playmaker.announceInfo()))
              break;
            playmaker.drawChess();
          }while(referee.judgeTime());
          playmaker.showResult();                    //输出胜负结果等
          referee.stop();

          return 0;
        }
```

在程序中，Player、Referee、Playmaker 和 ChessInfo 分别是对参赛选手、裁判、组织者和棋局这些概念的简化描述（ChessInfo 的定义未给出），可以理解为是一些数据类型。每种概念所生成的实例（变量）称为“对象”。棋局信息由组织者维护，并可以向外界公布。其中，棋手对象 player1 和 player2 负责走棋，并告知组织者对象 playmaker 有关棋子布局的变化。组织者对象接收到这些信息后负责在屏幕上显示这种变化，裁判对象 referee 负责对棋局以及比赛时间进行判定。程序实现时，组织者先显示棋局，裁判宣布比赛开始，执黑和执白选手交替落子。每次落子后，裁判确定是否该选手已获胜，一旦获胜或比赛时间到则终止比赛。否则，组织者刷新棋局。

尽管上述程序还不能全面解释面向对象程序设计的所有特性，但可以明显看出，面向对象是以功能而不是步骤来划分问题的，这种方式与人的日常思维方式吻合。不同对象维护着自身的相关信息（数据、属性），并具有一定的功能（函数、方法）。所有对象各司其职。对象自身属性和行为方式的改变不会影响到其他对象，因为对象间仅通过互通消息实现合作，如图 1-3 所示。

图 1.3 中的接口就是指一个对象能够对外提供的服务（方法）。从实现上看，对象的每次操作都是在该对象接收到一定消息（用“对象.函数名”形式表示）后的自主行为，具有“主语＋谓语”的形式。

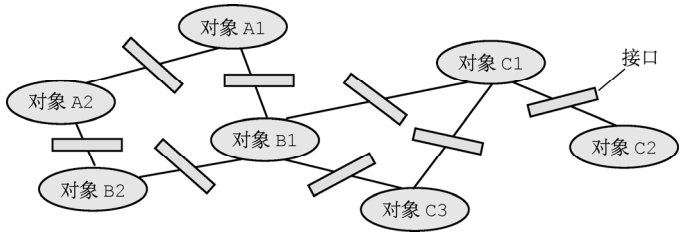

图 1-3　面向对象运行模式

在采用面向对象技术解决问题时，首先要考虑整个项目的求解是由哪些对象组成的，再分析对象应具有什么样的行为，最后考虑对象之间应怎样协作。

总之，过程化与面向对象是两种分析和解决问题的不同方法，对于一些简单的问题，基于过程的解决方法是十分有效的，而对于大型、复杂的系统，采用面向对象方法更能显示出优势，有利于采用对象构成软件“积木插件”，进而在一定程度上解决软件重用的难题。

1.3　面向对象程序的主要特征

客观世界是由对象组成的，这是面向对象程序设计（Object Oriented Programming，OOP）的基础和出发点。例如，考虑“我看电视”这样一个场景。“人”、“遥控器”、“电视”是问题中所涉及的概念，

每种概念在 C++中用一个数据类型来描述，称为“类”（class），而客观存在的实体对象都是某种概念的一个具体实例，有形体特征和具体行为。这里有一个“我”，是“人”类的一个对象。一个具体的电视机，是“电视”类的一个对象，还包括一个“遥控器”类的对象。在实际工作时，对象“我”操纵遥控器对象，遥控器对象向电视机对象发送开机、关机和调换频道等消息（指令），电视机对象响应这些消息，执行相应的操作。

1.3.1 抽象与封装

尽管一些简单的数据如整数、字符等可以由系统提供的基本数据类型来刻画，但更一般的概念必须自己来描述，其结果就是定义一个类。下述代码给出了对电视的一种概略描述：

```
class  TV
{
  public:                                          //实际设计时应采用 private 关键字
     int color;                                    //颜色属性
     int size;                                     //尺寸属性
     //...                                         //其他属性
   public:
     TV(参数列表);
     void open();                                  //打开
     void close();                                 //关闭
     void changeChannel(int channel);              //调换频道
     virtual void display(int channel);            //显示频道
     //...                                         //其他方法
};
```

通常，任何概念都会包含两方面的内涵，其一体现事物的状态，其二是其行为。例如，“人”是一种概念，通过身高、体重、性别、年龄和肤色等体现了人的自然状态，而生长、学习、吃饭和劳动等说明了人可具有的行为。这种状态特征用数据来刻画，称为类的“属性”，而行为特征用函数（算法）来体现，称为类的“方法”，属性和方法都是类的成员。

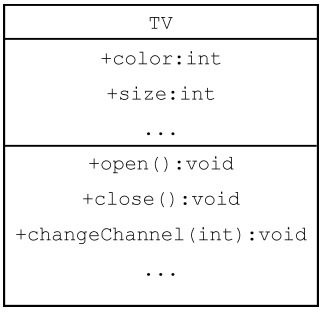

图 1-4　类 TV 的图形描述

在定义中，类 TV 有两类成员，color、size 等为属性，open、close 和 changeChannel 等为类的方法。通常，类可由图 1-4 所示的图描述，它来自于统一建模语言 UML 的表示方法。

类的定义是一种数据类型定义，因此，可以像 C++的内置类型那样生成变量，这种变量就称为一个类的实例或对象。例如，下述代码生成了类 TV 的一个对象：

```
TV  tv(参数列表);                                  //对象定义
```

生成对象时，参数列表被传递给类的一个特殊方法 TV，它与类的名字相同，称为“构造器”。因为构造器的存在，使我们能够生成具有各种各样状态而非千篇一律的对象。可见，类是一种用来生产对象的“模板”。

一个类定义所达到的最重要的效果是实现了对数据和函数的封装（encapsulation），而封装的主要目的是数据隐藏。这不仅是对客观实体的一种合理完整的刻画，也使对象本身所包含的内部数据不致于被无意中破坏。对电视机的使用者来说，如果需要调换频道，可以通过遥控器向电视机 tv 发送调换频道信号（消息）。电视机可以接收和响应此消息，但如何调换频道是电视机本身的能力，由电视机而

不是人或遥控器来实现。使用者不必关心和改变电视机的内部结构，也不必知道其内部工作原理。

类的所有实例具有相同的行为，但不同的属性使每个对象的行为也会体现出与自身属性相关的特点。例如，每个人都有劳动行为，但因为自然条件如体力、智力的不同，其劳动所体现的结果也存在差异。

访问对象的属性或方法一般采用“对象名.成员名”表示。例如，下述代码设置 tv 对象的尺寸为 100，并执行它的 open 方法，使电视机打开：

```
tv.size = 100;
tv.open();
```

不过，由于类定义时允许指定成员对外界的公开程度，如 public 或 private 等，因此外界有可能无权访问类的某些特殊成员。

应该说，将概念用类来刻画是一种设计方法。为了能对类进行合理规划，正确反映出问题领域中的概念，必须对各种对象进行细致观察、分析和分类，以总结出它们的共性和差异，最终用正确的属性和方法实现对概念的描述。这种过程称为“抽象”。

1.3.2　由继承实现重用

如果考虑制造新的电视机，可以有如下两种方法：

（1）从底层全新设计，即从勾画电视机的原始草图开始；

（2）在原设计基础上进行改造、提高和增添新的功能。

很明显，除非极特殊的情况，现实生活中很少采用第一种方法，因为后一种方法会得到更多好处，不仅可以降低工作的难度，也能大幅度减少工作量。事实上，不仅人类生活，自然界的物种进化也遵循着类似的规则。为了适应新的情况变化，物种会增加和改变某些能力或特性，但主要的属性和行为仍从祖先继承（inheritance）而来。

面向对象设计采取了类似的方法，即从已有的类派生新类，并将其作为重要的支撑技术之一。这种技术使得新类继承了已有类的所有属性和方法，但又可以增加必要的成员，以体现新的功能和适应新的要求，这种技术可以由图 1-5 来描述。

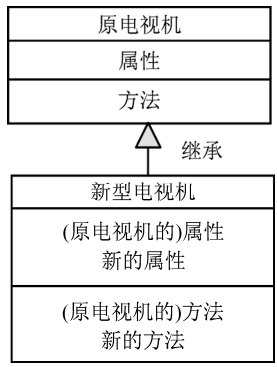

图 1-5　类的继承

例如，我们以前述的电视机 TV 为基础，生产一种能够支持网络访问的新型电视机 NetTV，可以在原设计基础上增加网络访问器件和网络访问能力，通过如下方式实现：

```
class  NetTV : public  TV
{
    NetDevice  netDevice;          //NetDevice 为网络访问器件结构
  public:
    NetTV(...);
    setNetDevice(...);
    connect();
    disconnect();
}
```

设计中增加了一个描述网络访问设备的 netDevice 属性，并添加了调整该属性的一个方法 setNetDevice，重要的是，新型电视机增加了网络连接功能 connect 和断开连接功能 disconnect。

代码的继承性体现在“: public TV”，它说明类 NetTV 继承自类 TV，也可以说由 TV 类派生了

NetTV 类，TV 类与 NetTV 类之间形成了“父类”与“子类”的关系。NetTV 类从 TV 类得到了全部属性和方法，并进行了适度扩充。对于新类来说，继承得到的成员与自己新增的成员在地位和使用方法上是一致的。

支持继承可以使已经设计好并经过考验的类得到最大限度的重用，不仅减少了新设计的工作量，也使新类更健壮和容易调试。可以说，继承的主要目的就是代码重用。

1.3.3 由多态反映变革

多态（polymorphism）是指多种形态，或者说不同对象在相同概念下能够表现出各自的特殊行为。例如，一个技术部经理和一个销售部经理是不同的对象，都有“工作”的行为，但目标和方式都不相同。作为对象，一个矩形和一个圆都可能有“绘制自身”的行为，但所得结果完全不同。这是一种“低级的”多态行为，C++用“函数名相同，但函数体不同”的函数重载来体现。

“高级的”多态是指在具有继承关系的类中，父（祖先）类和子（孙）类之间行为上的差异，采用虚函数方法实现。例如，为了实现一种支持 3D 显示的电视机，需要改造原电视机的显示方式，可以按如下方式由 TV 派生新类：

```
class  TV3D : public  TV
{
    //此处增加必要的新属性
   public:
     TV3D(...);
     void  display(int  channel);
     ...;                        //其他
}
```

生成这两类电视机的对象，并按如下方式执行显示功能：

```
TV  tv(参数);
TV3D  tv3d(参数);
tv.display(channel);
tv3d.display(channel);
```

对象 tv 和 tv3d 都有相同的行为 display，但两个类中各自有具体的实现，执行方式各异。更为重要的是，如果采用一个父类指针来指向不同对象并调用对象的方法，系统仍能够支持这种不同的行为：

```
TV *ptv = &tv;
ptv->display(channel);       //调用 tv 的 display
ptv = &tv3d;
ptv->display(channel);       //调用 tv3d 的 display
```

当指针分别指向父类对象和子类对象时，程序中采用完全相同的代码但调用了不同的方法，创造这种“奇迹”的就是“虚函数”机制。注意到 TV 类的 display 方法声明时增加了一个关键字“virtual”，它的作用就是说明 TV 及其派生类的 display 方法处于一种“虚”的关系，要在程序运行时根据指针或引用所指向的对象决定应该执行的方法。由于第一次调用 display 时 ptv 指向 TV 的对象，应该调用 TV 的方法，而第二次调用 display 时 ptv 指向 TV3D 类的对象，故调用 TV3D 类的方法。从程序编译的角度说，这种编译方法称为“动态联编”。

抽象、封装、继承和多态构成了 OOP 的最基本特征，而抽象是其他特征的基础。通常，如果一种程序设计技术仅支持对象封装行为，则被认为是基于对象而非面向对象的。

1.4　面向对象的问题分析

与传统的按功能进行分析和设计方法不同，面向对象的程序分析、设计和实现都是围绕对象模型进行的。其中，分析阶段的目的就是找出和建立对象模型，设计阶段则是修改、细化和补充完善对象模型，最终用支持面向对象的语言如 C++对模型进行编码并投入运行。学习和建立用对象观点看待和分析问题是学习 OOP 的关键。为此，需要在深入理解用户需求的基础上，先确定问题领域中的类，也包括确定类的属性和方法，再确定类之间的联系，即对象模式。通常，确定类与对象模式是结合在一起的。

1.4.1　确定类

面向对象程序是由对象的协作实现的，必须先规划出类，才能生成对象。通常，客观世界中的类包括以下三类：

（1）问题领域中可感知的实体和抽象的事物所体现的概念。例如，学校、企业、汽车、大桥、教科书等是实体，而思想、规定、运动等是抽象的事物。

（2）问题领域中的人或组织的角色，如教师、学生、教练、裁判、运动员、医生和病人等。

（3）问题领域中所涉及的重要事件，如教师教课、顾客购物、一次自动化生产过程、一次交通事故等。这里的事件是指一个状态的改变或一个活动的发生。

从类的功能上说，一种是最常见的用于生成对象的类，可简称为“对象类”。另一类是用于规定多个子类或多层继承关系的公共属性和方法的“抽象类”。例如，为了描述矩形 Rect、圆 Circle 和三角形 Triangle，可以定义一个“形状”抽象类 Shape，规定三个对象类 Rect、Circle 和 Triangle 的基本方法如 draw，并使其从形状类 Shape 派生，参见图 1-6。

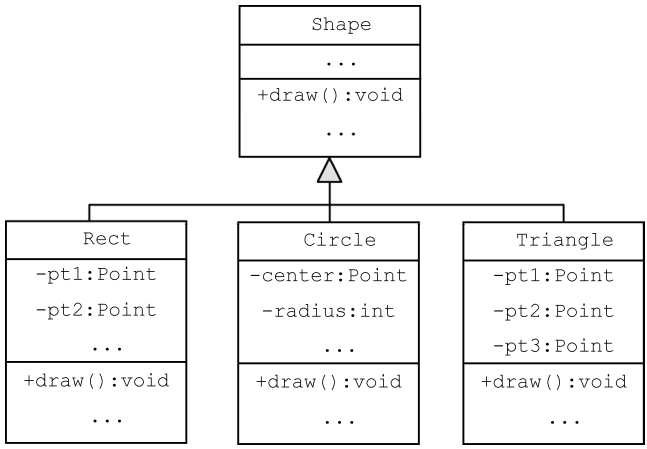

图 1-6　作为公共父类的抽象类

抽象类通常用于规定派生类的公共接口。在图示中，类表示为一个矩形，但也可以将抽象类表示为单边矩形，而对象类表示为一个加双边框的矩形或单边矩形。

1.4.2　确定类的属性

一个类可以包含多个属性，这些属性描述了类所具有的性质、特征和状态。

通常，类的每个对象都有不同或相同的属性值，这样的属性称为实例属性。例如，圆类可以有圆心和半径属性，且每个圆都可以有不同的圆心和半径值。一个类中还可以含有公用属性，此时，所有对象都有相同的属性值，称其为类属性。例如，一个卖场有若干台自动收款机，收款机类可以维持一张价格表属性，所有对象都将使用唯一的价格表。这种属性类似于所有对象都可以访问的公共变量。

在 C++中，对象属性定义为类的普通数据成员，而类属性定义为类的静态（static）成员，且它们只允许由静态的方法访问。

1.4.3 确定类的方法

类的方法用于描述类所具有的功能，也说明了该类能为其他类提供的服务。虽然类可以包含仅供自身调用的方法，但在分析问题时，重要的是确定类的外部方法，它们规定了类与外部进行交互的接口。图 1-7 中的圆类 Circle 描述了这两类接口。

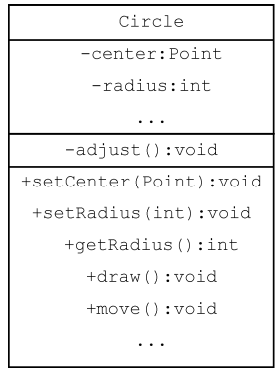

图 1-7 Circle 类

一般情况下，类总有一些方法与属性有关，用于实现对属性的读写访问，如设置圆心 setCenter、取得半径 getRadius 等，另一些则提供了对外服务，如绘制 draw。同样，类 NetTV 中的 setNetDevice 是属性访问方法，connect 和 disconnect 则是对外服务方法。

在图示中，每个属性和方法之前用“+”和“–”表示其对外界公开还是隐藏。

1.4.4 确定对象模式

对象模式是指对象之间的联系方式，主要包括整体-部分模式、一般-特殊模式和消息模式。

（1）整体-部分模式是指一个表示整体概念的类由若干表示部分概念的类（对象）组成。例如，计算机由主机、显示器、键盘和鼠标组成，那么，计算机类 Computer 将包括主机 Mainframe、显示器 Displayer、键盘 Keyboard 和鼠标 Mouse 这些类的对象，参见图 1-8。

（2）一般-特殊模式表示一个类是另一个类的特殊形式，如 NetTV 是一种特殊的 TV，轿车是一种特殊的汽车，圆是一种特殊的形状等。当 A 类和 B 类有一般-特殊关系时，B 类由 A 类派生，B 类对象是 A 类对象的特殊实例。图 1-6 就体现了由 Shape 类派生的类 Rect、Circle 和 Triangle 的情形。

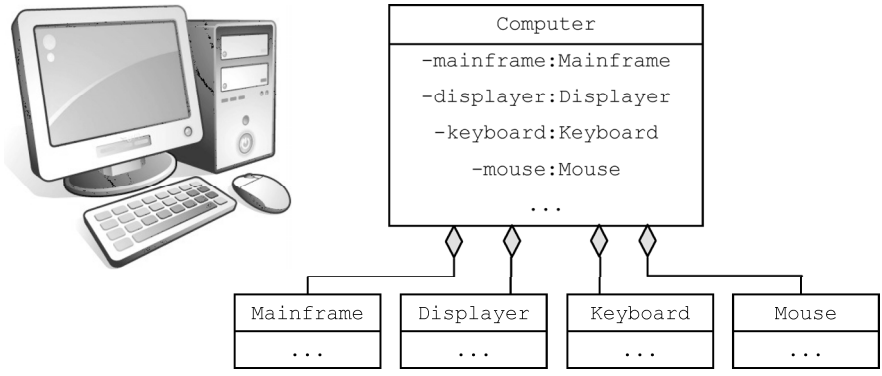

图 1-8 整体与部分关系

（3）消息模式是指类的实例（对象）之间如何通过发送和响应消息进行联系。面向对象的程序是由对象组成的，对象之间需要通过消息传递来协调工作。这是非常普遍的关系，一般称为“关联关系”。

消息是对象之间进行通信的一种规格说明，或者说消息是指希望一个对象执行某种操作的请求，

而对象执行操作称为对消息的响应。如果对象 A 的某个方法中调用了对象 B 的方法，就意味着 A 向对象 B 发送消息并请求响应，这种响应可能引发一定的操作并得到某些返回值。从实现代码看，消息就是通过一个对象对类方法的一次调用。例如，以下代码向 tv 对象发送了一条消息：

```
tv.changeChannel(5);                    //一条消息
```

可见，一条消息包括接收此消息的对象名、消息名和必要的参数，此例中分别是 tv、changeChannel 和 5。上述消息的含义是请求对象 tv 执行调换频道操作，但不涉及返回值。

在“我看电视”问题中，“我”对象向遥控器对象发送消息，遥控器对象响应这些消息，产生向电视机对象发送的消息，电视机对象响应消息，完成打开电视或替换频道等操作。以上三个类之间存在关联关系。

UML 图示中互相关联的类之间用直线相连。

通常，很难一次将问题域中所有对象及其模式分析清楚，因此，设计过程是一个不断精炼、修改和完善的过程，甚至在设计阶段也需要不断反复才能建立正确的模型。简言之，得到正确的类设计需要一个迭代过程。

思考与练习 1

1．OOP 有哪些主要特点？简述每种特点的含义。

2．什么是类？本章介绍的类的 UML 图示中主要包含哪些部分？类的继承关系用什么符号表示？

3．什么是对象？对象包含哪些部分？如何表示？

4．有哪些主要的对象模式？在 UML 图中如何表示？

5．什么是消息？一条消息中包含哪些成分？

6．如何理解结构化程序设计的“谓语＋宾语”结构和 OOP 的“主语＋谓语”结构？

7. 假定有一个网络公司，由总经理室、副总经理室、开发部、测试部、营销部组成，人员有一个总经理和两个副总经理，其他每个部门各有 5 名员工，且其中一个是部门经理。采用面向对象方法时各种部门和人员的类应如何描述？用 UML 图示如何表示公司的组成？

实　验　1

1．使用附录所述环境验证示例程序 Example1_1.cpp，体会程序的编辑、编译、连接和运行过程。

2．按本章中的说明为示例程序 Eample1_1.cpp 增加注释，体会允许插入注释的位置并理解应该如何添加注释。

3．编写一个 C++程序，从键盘读入两个整数 a 和 b，交换它们的值并输出。

4．编写程序并构造适当的测试数据，体会并说明’\n’与 endl 用在 cout 对象中有何异同。