单元 1

我的第一个 C#程序

单元导读

本单元将对 C#的基础知识进行简要介绍，其中主要包括.NET Framework、Common Language Runtime 以及 C#的特点等。

本单元的目的，是让读者快速了解 C#的基本概念，对 C#有一个基本认识。

在任何一门新技术的学习过程中，最开始的部分都比较困难，所以读者可以不必过分纠结于本单元中的名词，只需有一个简单的了解即可。

学习目标

- 初步认识.NET，并了解它的组成和特点。
- 熟悉 Visual Studio .NET 开发环境，掌握使用它开发应用程序的步骤。
- 学习 C#程序的基本结构。
- 掌握如何编辑、编译和运行 C#应用程序。

1.1 案 例 描 述

.NET 是目前最主流的一门软件开发技术。自微软 2000 年推出下一代互联网构想以来，伴随着 Microsoft .NET 平台的构建和实施，.NET 以其独有的高效开发特点、简单易行的版本控制等多方面的全新技术优势，迅速风靡北美各大企业，并深受全球开发者的喜爱。.NET 是世界上最大的软件公司——微软花费 300 亿美元精心打造的开发平台，可以开发 Web 程序、Windows 应用程序和 WAP 无线网络应用程序等，其在大型系统开发中的份额越来越重。

本案例中，我们想要初步了解 C#和.NET。将编写一个应用程序，能根据用户输入的名字，显示“Hello, XXX, 欢迎来到 C#的世界！”欢迎词。执行结果如图 1-1 所示。

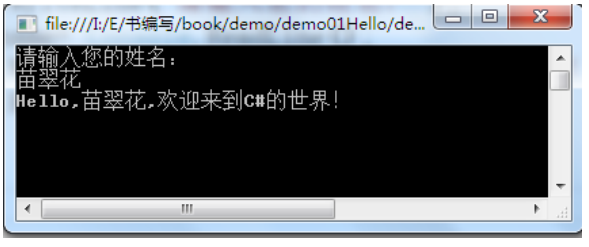

图 1-1 第一个程序

1.2 知 识 链 接

1.2.1 .NET 概述

微软对.NET 的定义是：.NET is a revolutionary new platform, built on open Internet protocols and standards, with tools and services that meld computing and communications in new

ways(.NET 拥有以新方式融合计算和通信的工具和服务，它是建立于开放互联网协议标准的革命性的新平台)。

.NET 框架(.NET Framework)是微软公司为了与 Sun/Oracle 公司的 Java(EE)竞争，于 2000 年 6 月提出来的一种新的跨语言、跨平台、面向组件的操作系统环境，适用于 Web 服务(Web Services)和因特网(Internet)分布式应用程序的生成、部署和运行。.NET 框架也是 Windows Vista、Windows 7 和 Windows 8 等新版 Windows 操作系统的核心部件。

技术人员要想真正了解什么是.NET，必须先了解.NET 技术出现的原因和它想解决的问题。技术人员一般将微软看成一个平台厂商。微软搭建技术平台，而技术人员在这个技术平台之上创建应用系统。从这个角度看，.NET 也可以定义为：.NET 是微软的新一代技术平台，为敏捷商务构建互联互通的应用系统，这些系统是基于标准的、联通的、适应变化的、稳定的和高性能的。从技术的角度看，.NET 应用是一个运行于.NET Framework 之上的应用程序。

1.2.2　.NET Framework

.NET 平台主要包含的内容有.NET Framework(.NET 框架)、基于.NET 的编程语言及开发工具 Visual Studio 等，其体系结构如图 1-2 所示。

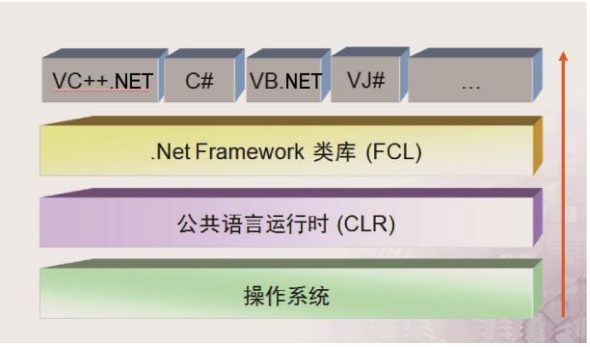

图 1-2　.NET 的体系结构

.NET 平台的基础和核心是.NET Framework，.NET 平台的各种优秀特性都要依赖它来实现。.NET Framework 包括两部分内容：一是框架类库(Framework Class Library，FCL)；二是公共语言运行时(Common Language Runtime，CLR)，也译作“公共语言运行库”。

1. FCL(框架类库)

从图 1-2 中可以看出，在.NET 平台上可以使用 C#、VB.NET 等多种语言来编写程序，不同的语言可以使用相同的 FCL。

FCL(Framework Class Library，框架类库)为开发人员定义并提供了统一的、面向对象的、分层的和可扩展的类库集，其主要部分是 BCL(Base Class Library，基类库)。通过创建跨所有编程语言的公共 API 集，公共语言运行库使得跨语言继承、错误处理和调试成为可能。从 JScript、Visual Basic 到 Visual C++、C#、F#的所有编程语言(通过托管扩展)具有

对框架的相似访问，开发人员可以自由选择它们要使用的语言。

相对于贫乏的 C++类库和丰富的 Java 类库，.NET 框架类库非常庞大，包含数百个命名空间、数千个类、接口和值类型。该库提供对系统功能的访问，是建立.NET 框架应用程序、组件和控件的基础。框架类库采用命名空间来组织和使用，如.NET 4.5 版的类库提供了 588 个命名空间，其中用于应用编程的 BCL(System 型)命名空间有 407 个。

2. CLR(公共语言运行时)

与 Java 虚拟机(Java Virtual Machine，JVM)相似，CLR 也是一个运行时环境。CLR 负责内存分配和垃圾回收，也就是通常所说的资源分配，同时保证应用和底层系统的分离。总而言之，它负责.NET 库所开发的所有应用程序的执行。

CLR 所负责的应用程序在执行时是托管的。托管代码带来的好处是跨语言调用、内存管理、安全性处理等。CLR 隐藏了一些与底层操作系统打交道的环节，使开发人员可以把注意力放在代码所实现的功能上。非 CLR 控制的代码即非托管(unmanaged)代码，如 C++等，这些语言可以访问操作系统的功能，直接进行硬件操作。

垃圾回收(Garbage Collection)是.NET 中一个很重要的功能，尽管这种思想在其他语言中也有实现。这个功能保证应用程序不再使用某些内存时，这些内存就会被.NET 回收并释放。这种功能被实现以前，这些复杂的工作主要由开发人员来实现，而这正是导致程序不稳定的主要因素之一。

一个典型的.NET 程序的运行过程主要包括以下几个步骤。

(1) 选择编译器。为获得公共语言运行库提供的优点，必须使用一个或多个针对运行库的语言编译器。

(2) 将代码编译为 Microsoft 中间语言(MSIL)。编译器将源代码翻译为 MSIL 并生成所需的元数据。

(3) 将 MSIL 编译为本机代码。在执行时，实时(JIT)编译器将 MSIL(微软中间语言)翻译为本机代码。在此编译过程中，代码必须通过验证过程，该过程检查 MSIL 和元数据以查看是否可以将代码确定为类型安全。

(4) 运行代码。公共语言运行库提供使执行能够发生以及可在执行期间使用的各种服务的结构。

1.2.3 开发环境和 C#语言

1. Visual Studio 2017

Visual Studio 是目前最流行的 Windows 平台应用程序的集成开发环境，最新的版本是 Visual Studio 2017，其官方发布时给出的“What’s new in VS 2017”如图 1-3 所示，其特点是适用于开发 Android、iOS、Windows、Web 和云的应用，支持创建.NET Core 程序，可以选择.NET Core 1.0 或.NET Core 1.1。

本书的所有范例是在 Visual Studio 2017 Professional 环境下调试的。

图 1-3　VS 2017 的新特性

VS(C#)不同的版本如表 1-1 所示。

表 1-1　Visual Studio、.NET Framework 和 C#版本

名　称	.NET Framework 版本	发布日期	C#版本
Visual Studio 2002	1.0	2002-02-13	1.0
Visual Studio 2003	1.1	2003-04-24	1.5
Visual Studio 2005	2.0	2005-11-07	2.0
Visual Studio 2008	3.5	2007-11-19	3.0
Visual Studio 2010	4.0	2010-04-12	4.0
Visual Studio 2012	4.5	2012-08-25	5.0
Visual Studio 2013	4.5、4.5.1	2013-10-17	5.0
Visual Studio 2015	4.6	2015-07-21	6.0
Visual Studio 2017	4.7	2017-03-07	7.0

说明： C#版本指 C#语言规范的版本；Visual C#版本是开发工具的版本，它是 Visual Studio 的一个组件。

安装 Visual Studio 2017 的过程如下。

(1) 从微软网站下载安装文件 visual_studio_professional_2017_x86_x64.exe，只有不到 1MB 大小。这只是一个引导程序(Web Installer)，启动之后，将看到安装界面变得更可视化了，总共有三个选项卡。

① “工作负载”选项卡。工作负载总共分 3 类，拥有 13 个具体选项。

Windows 类(共 3 项)，如图 1-4 所示。

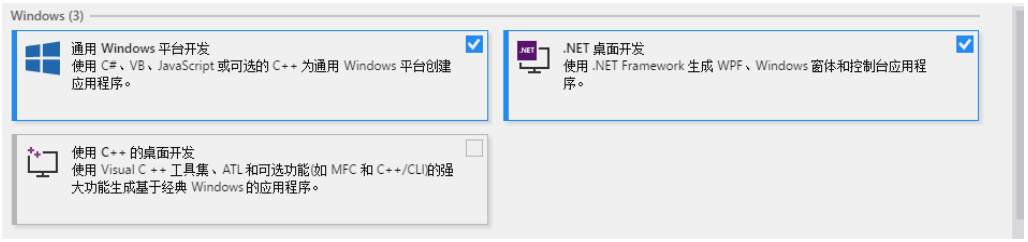

图 1-4　Windows 类

Web 和云类(共 5 项)，如图 1-5 所示。

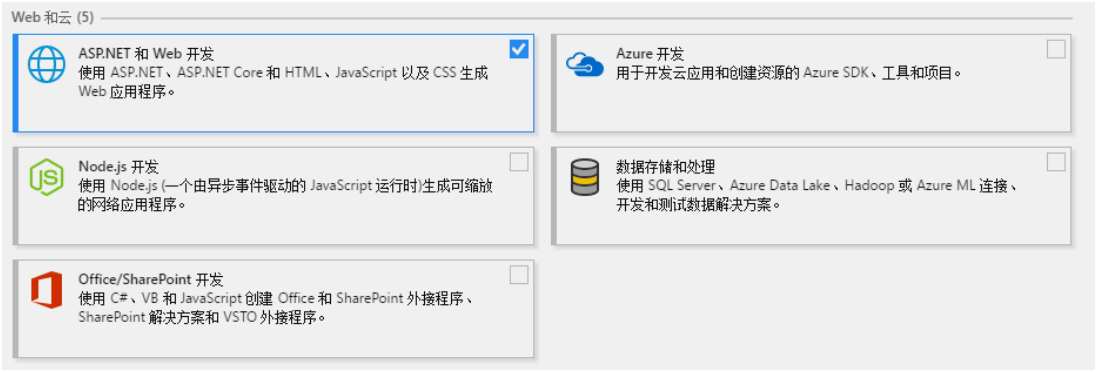

图 1-5　Web 和云类

移动与游戏类(共 5 项)，如图 1-6 所示。

图 1-6　移动与游戏类

② “单个组件”选项卡。包含其他工具集(共 3 项)，如图 1-7 所示。

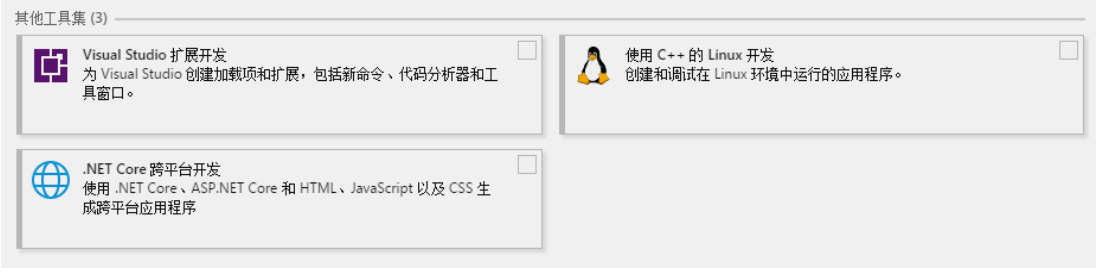

图 1-7　其他工具集

③ “语言包”选项卡。从中可以选择各种语言，在这里我们选择简体中文。

(2) 勾选需要的组件即可进行安装。为学习本课程以及后继的 ASP.NET 和网络编程，可以按如图 1-8 所示勾选(自定义选择安装路径时，注意所属路径的预留空间要充足，否则安装会失败)。然后进行安装。

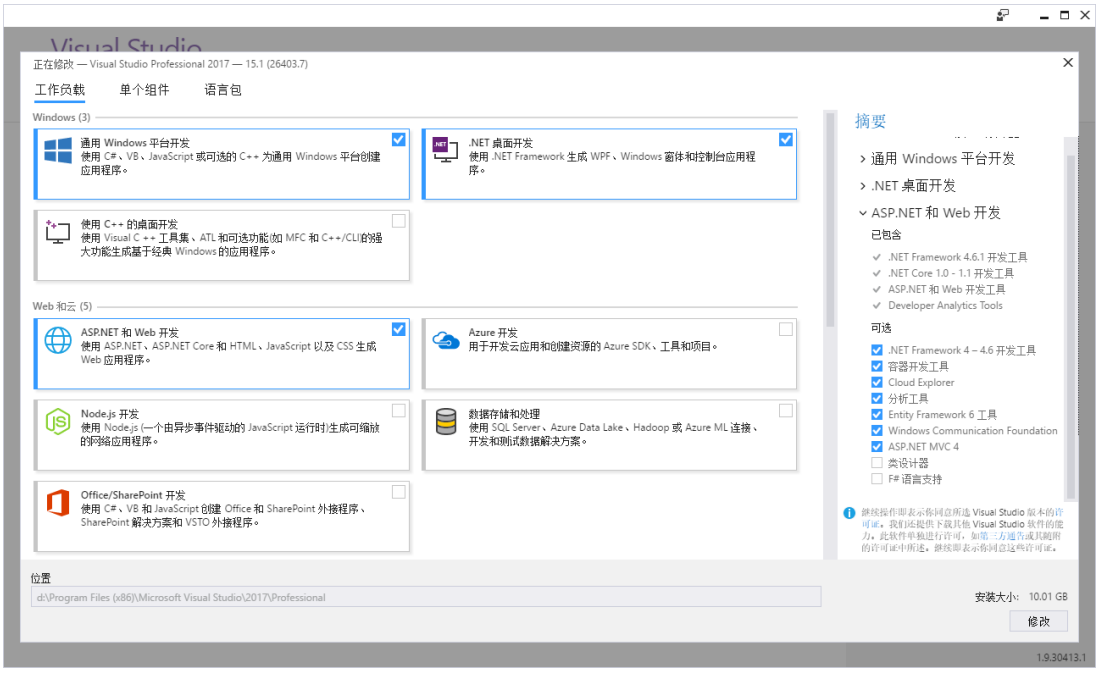

图 1-8　勾选需要的功能并安装

(3)　进入下载和安装界面。安装过程中，Visual Studio 2017 会占用很多的系统资源，所以最好不要开启太多其他软件，等待安装完成，如图 1-9 所示。

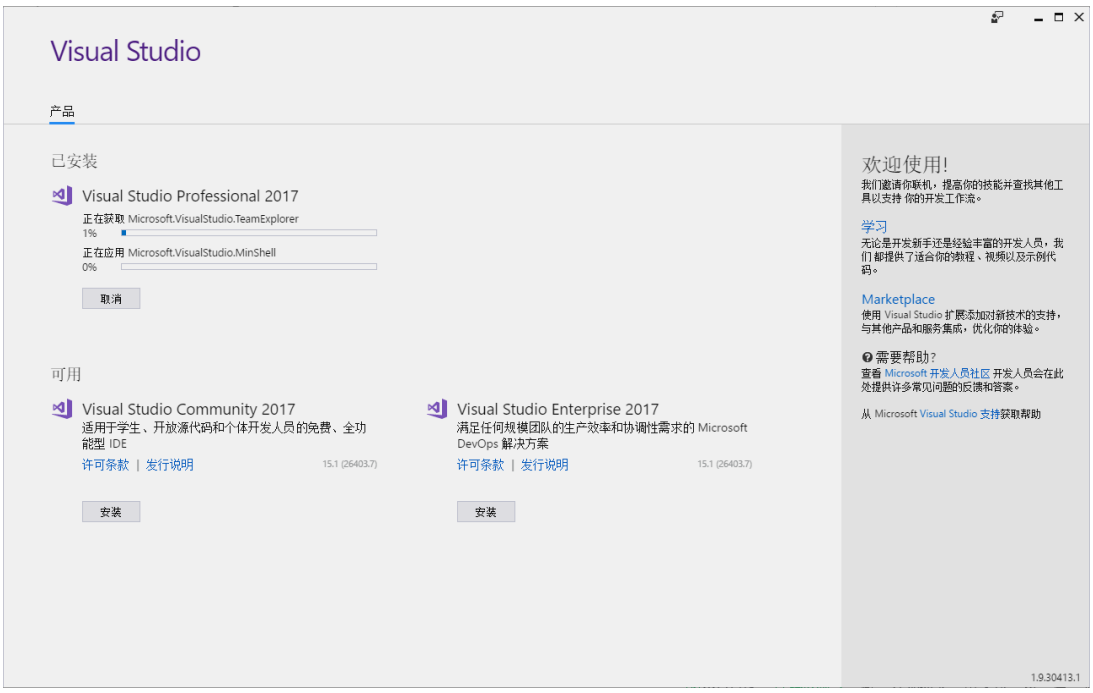

图 1-9　正在安装

(4) 等待大概 30 分钟，就可以完成安装，安装成功后就可以启动 Visual Studio 2017 并开始使用了，如图 1-10 所示。

(5) 第一次打开 Visual Studio 2017 时，需要进行一些基本配置，如开发设置、颜色主题，如图 1-11 所示。可以根据自己的需求设置，然后等待几分钟就可以使用。

图 1-10 安装完成后启动

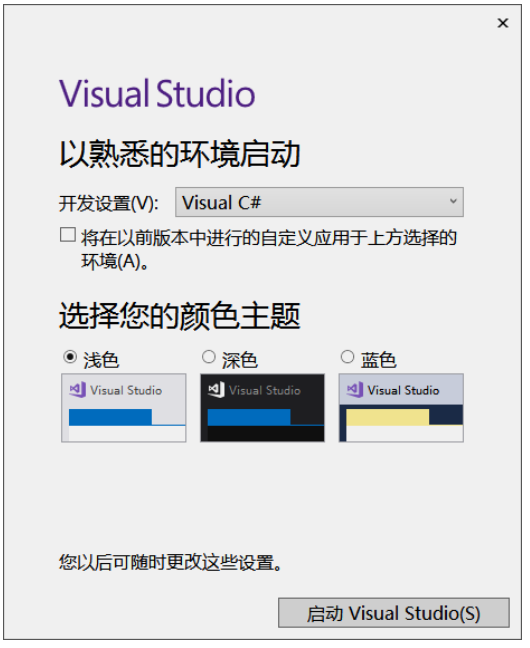

图 1-11 基本配置

由于 Visual Studio 引入了一种联网 IDE 体验，所以可以使用微软的账户登录，将会自动采用联网 IDE 体验的设备同步设置，包括快捷键、Visual Studio 外观(主题、字体)等各种类别的同步设置。

2. C#语言

C#是微软公司发布的一种面向对象的、运行于.NET Framework 之上的高级程序设计语言，也是.NET 平台上最重要的语言之一。C#和.NET Framework 同时出现和发展。由于 C#出现较晚，吸取了许多其他语言的优点，解决了许多问题。

C#看起来与 Java 有着惊人的相似；它包括了诸如单一继承、接口、与 Java 几乎同样的语法和编译成中间代码再运行的过程。但是 C#与 Java 有着明显的不同，它借鉴了 Delphi 的一个特点，与 COM(组件对象模型)是直接集成的，而且它是微软公司 .NET Windows 网络框架的主角。

C#是一种安全的、稳定的、简单的、优雅的由 C 和 C++衍生出来的面向对象的编程语言。它在继承 C 和 C++强大功能的同时，去掉了一些它们的复杂特性(例如没有宏以及不允许多重继承)。C#综合了 VB 简单的可视化操作和 C++的高运行效率，以其强大的操作

能力、优雅的语法风格、创新的语言特性和便捷的面向组件编程的支持等特色成为.NET 开发的首选语言。

C#是面向对象的编程语言。它使得程序员可以快速地编写各种基于 Microsoft .NET 平台的应用程序，Microsoft .NET 提供了一系列的工具和服务来最大程度地开发利用计算与通信领域。

C#使得 C++程序员可以高效地开发程序，且因可调用由 C/C++ 编写的本机原生函数，因此绝不损失 C/C++原有的强大的功能。因为这种继承关系，C#与 C/C++具有极大的相似性，熟悉类似语言的开发者可以很快地转向 C#。

使用 C#，程序员可以创建传统的 Windows 客户端应用程序、XML Web 服务、分布式组件、客户端/服务器应用程序、数据库应用程序以及很多其他类型的程序。

1.2.4　使用 Visual Studio 创建项目

启动 Visual Studio 2017 后，默认出现起始页，如图 1-12 所示。

图 1-12　Visual Studio 2017 的起始页

在起始页中，我们可以直接查看最新功能、最新开发人员新闻以及各个 Dev Center 开发中心，也可以查看新的项目、近期的工作。如果起始页被关闭，需要再次显示，可以在“视图”菜单中找到“起始页”命令来显示。

程序员可以使用起始页的“开始”→“新建项目”来创建新的项目，或者通过“文件”菜单中“新建”→“项目”命令(见图 1-13)，调出 Visual Studio 的“新建项目”对话框，如图 1-14 所示。

图 1-13　Visual Studio 2017 创建新项目

图 1-14　“新建项目”对话框

Visual Studio 的界面会随着所打开文件的类型动态地改变。图 1-15 和图 1-16 分别是创建一个控制台应用程序时的界面和一个窗体程序时的界面。

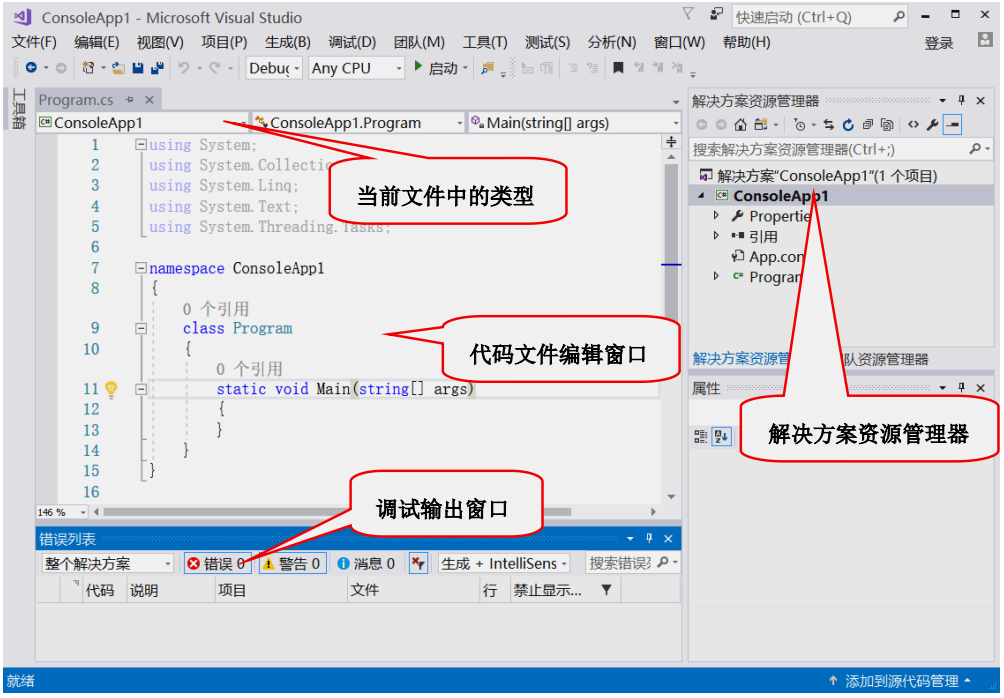

图 1-15　Visual Studio 工作界面(控制台应用程序)

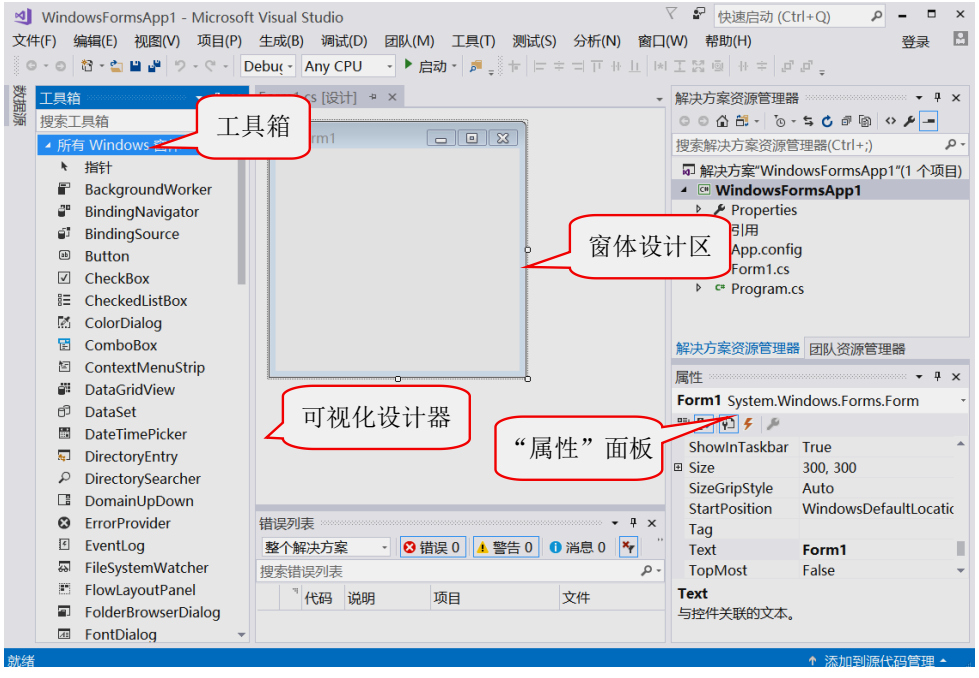

图 1-16　查看窗体设计视图时的界面

在项目保存位置，以控制台应用程序为例说明我们可以看到相应的文件和文件夹，如图 1-17 所示。

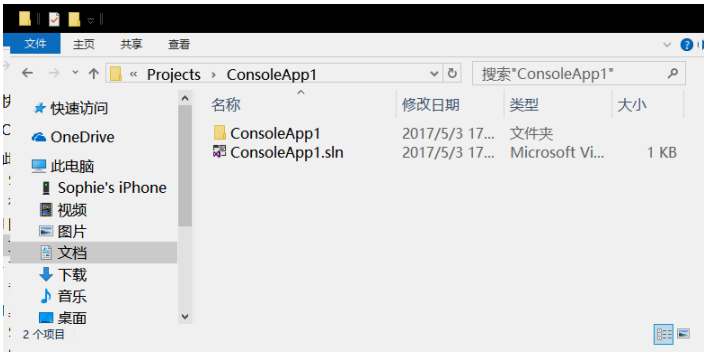

图 1-17　解决方案文件夹

扩展名为.sln 的文件是解决方案文件。直接双击此文件，会启动 Visual Studio 并打开该解决方案。

子文件夹 ConsoleApp1 是项目文件夹，其中存放了该项目的相关项。在这个文件夹下，扩展名为.csproj 的文件是项目文件，其中记载着关于项目的管理信息。扩展名为.cs 的文件是 C#的源代码文件。子文件夹 bin 中存放项目编译后的输出。子文件夹 obj 存放编译时产生的中间文件。而 Properties 文件夹中存放有关程序集的一些内容，主要是一个 AssemblyInfo.cs 文件，里面包含程序版本、信息、版权的属性文件。

1.2.5　C#程序结构

创建好一个控制台程序后，在 Visual Studio 的资源管理器中双击 Program.cs 文件，我们可以看到如图 1-18 所示的已经生成的程序框架。我们在 Main()方法下面的大括号里可编写自己的代码。


```
//这是用 C# 编写的一个简单的 HelloWorld 程序
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Program
{
    class HelloWorld
    {
        public static void Main()
        {
            Console.WriteLine("Hello World");
        }
    }
}
```

图 1-18　已经生成的程序框架

1.2.6　命名空间

命名空间是一种组织 C# 程序中出现的不同类型的方式。命名空间在概念上与计算机文件系统中的文件夹有些类似。与文件夹一样，命名空间可使类具有唯一的完全限定名称。一个 C# 程序包含一个或多个命名空间，每个命名空间或者由程序员定义，或者作为先前编写的类库的一部分定义。

例如，命名空间 System 包含 Console 类，该类包含读取和写入控制台窗口的方法。System 命名空间也包含许多其他命名空间，如 System.IO 和 System.Collections。

.NET Framework 本身有八十多个命名空间，每个命名空间有上千个类：命名空间被用来最大程度地减少名称相似的类型和方法引起的混淆。

如果在命名空间声明之外编写一个类，则计算机将为该类提供一个默认命名空间。

若要使用 System 命名空间包含的 Console 类中定义的 WriteLine 方法，可使用如下所示的代码行：

```
System.Console.WriteLine("Hello, World!");
```

记住要在 Console 类中包含的所有方法之前加 System，这一做法很快便会令人觉得很繁琐，因此将 using 指令插入到 C#源文件的开头是个非常有用的快捷方式，如下所示：

```
using System;
```

使用 using System;后，就会假定处于 System 命名空间中，以后就可以这样编写代码：

```
Console.WriteLine("Hello, World!");
```

编写大型程序时，经常会用到命名空间。使用自己的命名空间对名称相似的方法和类型提供了一定程度的控制。例如：

```
namespace AdminDept
class Manager
{
  long int salary;
  ...
}
...
```

```
namespace ITDept
class Manager
{
  long int salary;
  ...
}
...
```

以上都有 Manager 这个类的定义，但它们处于不同的命名空间中，当我们要引用这两个类时，应当分别写成 AdminDept.Manager 和 ITDept.Manager。

1.2.7　程序的运行与调试

在程序编写过程中，如果有语法错误，会在出错代码下方出现红色波浪线，并在下方的“错误列表”框中会出现提示，如图 1-19 所示，C#语句是以分号“;”结束的，所以在错误列表框中提示“应输入;”。

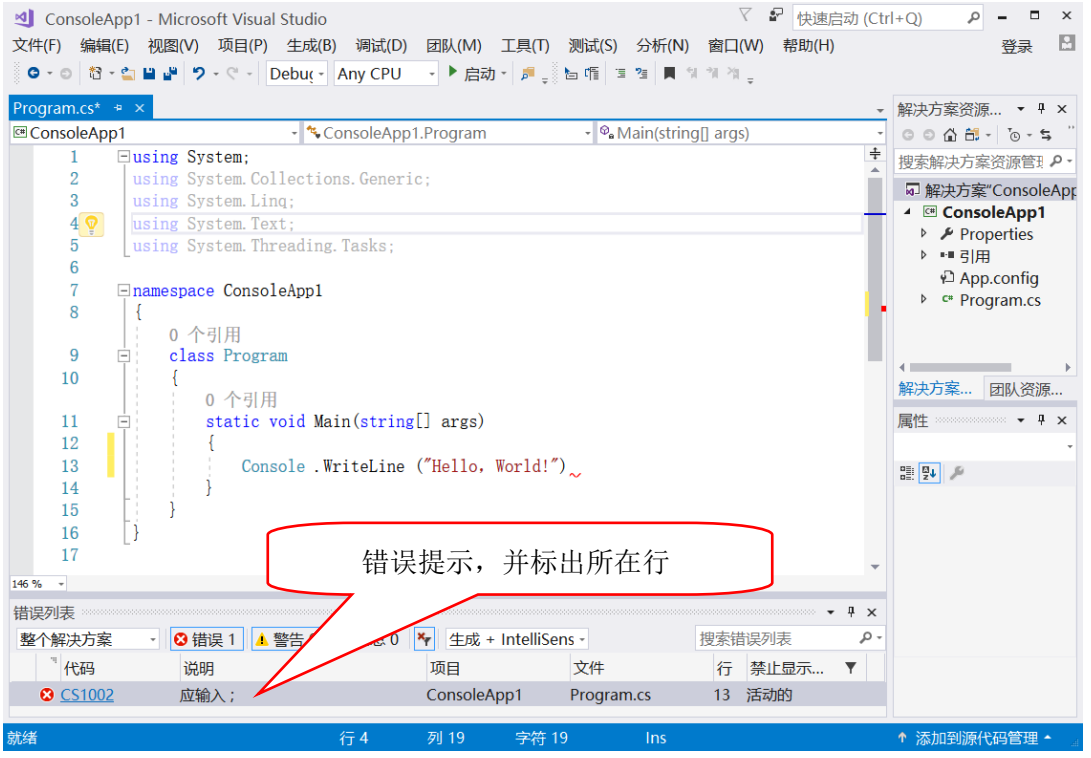

图 1-19　错误列表

更正后，我们可以运行程序查看结果，运行的方式有两种。

(1)　单击 Visual Studio 工具栏中的“启动”按钮，如图 1-20 所示。

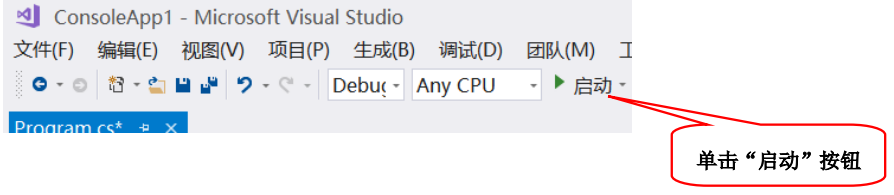

图 1-20　启动并运行程序

(2)　按快捷键 F5 也可以直接运行程序。按快捷键 F11 可以逐语句来执行，方便追踪变量的变化。

更多的调试方法与技巧在后面的单元中会详细讲解。

1.2.8　了解 MSDN

MSDN(Microsoft Developer Network) Library 涵盖了微软全套产品线的技术开发文档和科技文献(部分包括源代码)，也包括一些 MSDN 杂志节选和部分经典书籍的节选章节。付费后，MSDN Library 可以通过在线订阅或者以脱机方式浏览。

MSDN 实际上是一个以 Visual Studio 和 Windows 平台为核心整合的开发虚拟社区，网址为 https://msdn.microsoft.com/zh-cn/，包括技术文档、在线电子教程、网络虚拟实验室、微软产品下载(几乎全部的操作系统、服务器程序、应用程序和开发程序的正式版和测试版，还包括各种驱动程序开发包和软件开发包)、Blog、BBS、MSDN WebCast、与 CMP 合作的 MSDN.HK 杂志等一系列服务。

MSDN 可以随同 Visual Studio 一同安装，也可以单独安装。

1.3　案例分析与实现

1.3.1　案例分析

针对本单元所给出的案例，分析如下。

(1)　首先是显示“请输入您的姓名：”，用 Console 类的 WriteLine()方法即可。

(2)　控制台要接收用户输入的姓名，使用 Console 类的 ReadLine()方法。

(3)　在控制台打印输出“Hello, XXX, 欢迎来到 C#的世界！”，其中 XXX 是输入的姓名。C#中，Console.WriteLine()函数可以进行格式化输出，形式如下：

```
Console.WriteLine("格式字符串", 变量列表);
```

例如：

```
Console.WriteLine("我的课程名称是：{0}", course);
```

其中{0}代表占位符，可依次使用{0}、{1}、{2}等，与参数列表中的多个参数对应。例如：

```
Console.WriteLine("我叫{0},今年{1}岁了,我的工资是{2}元.",name,age,salary);
```

1.3.2　案例实现

案例的实现代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace demo01Hello
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("请输入您的姓名：");
            string name = Console.ReadLine();
```

```
            Console.WriteLine("Hello,{0},欢迎来到C#的世界！", name);
            Console.ReadKey();
        }
    }
}
```

说明： 结尾的 Console.ReadKey()的作用是等待键盘输入，退出程序。使调试时能看到输出结果。如果没有此句，控制台命令窗口会一闪而过。

运行后，就获得了本单元 1.1 节中的案例演示效果。

习　题　一

(1) 简述 C#与.NET 框架的关系。

(2) .NET Framework 的主要组件有哪些？它们的用途分别是什么？

(3) 可以通过 C#开发的应用程序有几种，分别是什么？

(4) Visual Studio 2017 开发环境中主要包含哪些窗口？

(5) 简述 Visual Studio 2017 集成开发环境中创建 Windows 应用程序的主要步骤。

(6) 在 Visual Studio 中，从_____窗口可以查看当前项目的类和类型的层次信息。

A. 解决方案资源管理器

B. 类视图

C. 资源视图

D. 属性

单元 2

变量与数据类型

单元导读

在第 1 单元中，读者接触到了第一个 C#应用程序，虽然只是简单地在控制台输出了一句问候语，但它却充分体现了 Visual Studio 2017 的易用性及 C#语言的特点。理解了 C#的用途之后，就可以学习如何使用它。数据类型在数据结构中的定义是一个值的集合以及定义在这个值集上的一组操作。变量是用来存储值的所在处；它们有名字和数据类型。变量的数据类型决定了如何将代表这些值的位存储到计算机的内存中。在声明变量时，也可指定它的数据类型。所有变量都具有数据类型，以决定能够存储哪种数据。

学习目标

- 理解 C#中的数据类型。
- 理解常量和变量的含义及用法。
- 理解数据类型转换的含义。

2.1 案 例 描 述

在银行系统中，保存了客户的相关信息，并可以对这些信息进行处理，其中包括客户账号、姓名、电话、家庭地址等。银行的实际客户因为主要的特点不同而被分为不同的类型，包括以支票使用为主的支票客户、以活期储蓄为主的储蓄账户客户等。我们现在用 C#来模拟银行系统中对客户各项详细信息的登记，并打印输入客户的基本信息。案例的执行效果如图 2-1 所示。

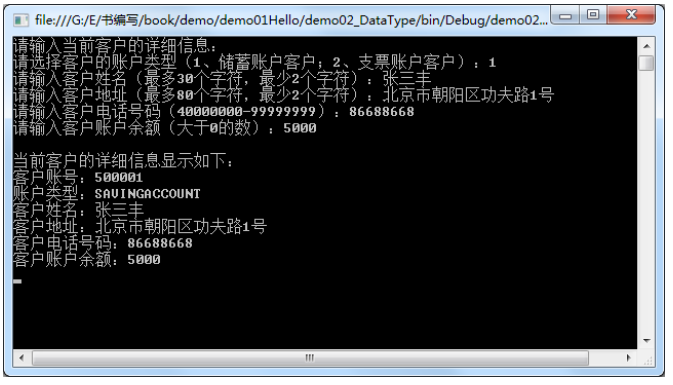

图 2-1 银行系统中客户信息的输入与描述

2.2 知 识 链 接

2.2.1 数据类型

数据就是数值，也就是我们通过观察、实验或计算得出的结果。数据有很多种，最简单的就是数值。数据也可以是文字、图像、声音等。数据可以用于科学研究、设计、查证

等。在程序设计中，数据是程序的必要组成部分，是程序处理的对象。不同的数据有不同的数据类型，不同的数据类型有不同的数据结构和存储方式，并且参与的运算也不同。C#的数据类型采用了类似于 C 和 C++语言的数据类型表示形式，但又有所改进。

C#将所有的数据类型分为两大类：值类型和引用类型。

值类型直接包含数据。每个值类型变量都包含有它自己的数据备份，因此对一个值类型变量的操作不会影响其他变量。引用类型包含指向对象实例的引用或指针。两个引用类型的变量可以指向同一个对象实例，因此对一个引用变量的操作会影响其他引用变量。可以通过下面的例子来加深对值类型和引用类型的理解。

假如把计算机的内存看作是一个储物间，里面有很多储物柜。对于值类型的数据，在内存里分配存储空间时，类似把物品直接存放在储物柜里。而对于引用类型的数据，则是在储物柜里存放物品存储的地址信息，不直接存储具体的物品，如果要用到该物品，就要先到相应的储物柜里查找物品存放的地址，再到相应地址去取该物品。

数据类型的分类如表 2-1 所示。

表 2-1　数据类型的分类

值 类 型	引用类型
数值类型	字符串
字符类型	数组
布尔型	类
结构	接口
枚举	对象

1. 值类型

值类型通常用来表示基本类型。C#的值类型主要包括整数类型、布尔类型、实数类型、字符类型、结构和枚举等。

表 2-2 列出了预定义的简单值类型。

表 2-2　预定义的简单值类型

名　称	CTS 类型	说　明	范　围
sbyte	System.Sbyte	8 位有符号整数	-2^7 到 2^7-1
short	System.Int16	16 位有符号整数	-2^{15} 到 $2^{15}-1$
int	System.Int32	32 位有符号整数	-2^{31} 到 $2^{31}-1$
long	System.Int64	64 位有符号整数	-2^{63} 到 $2^{63}-1$
byte	System.Byte	8 位无符号整数	0 到 2^8-1
ushort	System.Uint16	16 位无符号整数	0 到 $2^{16}-1$
uint	System.Uint32	32 位无符号整数	0 到 $2^{32}-1$

续表

名　称	CTS 类型	说　明	范　围
ulong	System.Uint64	64 位无符号整数	0 到 $2^{64}-1$
float	System.Single	32 位单精度浮点数	$\pm1.5\times10^{-45}$ 到 $\pm3.4\times10^{38}$(大致)
double	System.Double	64 位双精度浮点数	$\pm5.0\times10^{-324}$ 到 $\pm1.7\times10^{308}$(大致)
decimal	System.Decimal	128 位高精度十进制数	$\pm1.0\times10^{-28}$ 到 $\pm7.9\times10^{28}$(大致)
bool	System.Boolean	true 或 false	—
char	System.Char	单个 Unicode 字符	—

为了避免因为数据类型选择不当造成的程序错误，一定要综合考虑数据的范围及符号，选择合适的数据类型，在保证不出错的前提下做到节约存储空间。

例如，要存储人的年龄，就可以将存储年龄的变量数据类型指定为 byte 类型。因为 byte 型的变量所能表示值的范围在 0~255 之间。这个值范围能很好地表示人的年龄，并且所占的内存空间只有 1 个字节，存储空间最省。而 sbyte 类型的数据能存储−128~127 的数据，负数对于人的年龄来说，没有实际意义。

如果要让计算机处理带小数的数据，就要用到 float 数据类型或 double 数据类型，也叫作浮点类型。从表 2-2 中可以看出，浮点类型的数据有更高的精度，但是占用更多的内存空间。

注意： 应该注意区分浮点类型和数学中的实数。数学中的实数是连续的数据，有着严格的计算公式，而浮点数并非如此。

小数类型(decimal)数据是高精度的类型数据，占用 16 个字节，主要是为了满足需要高精度的财务和金融方面的计算。小数类型数据的后面必须跟 m 或者 M 后缀，来表示它是 decimal 类型的，如 3.15m，0.35M 等，否则就会被视为标准的浮点类型数据，导致数据类型不匹配。

在 C#中，可以通过给数值常数加后缀的方法来指定数值常数的类型，示例如下：

```
137f            //代表 float 类型的数值 137.0
137u            //代表 unit 类型的数值 137
137.2m          //代表 decimal 类型的数值 137.2
137.22           //代表 double 类型的数值 137.22
137             //代表 int 类型的数值 137
```

C#中的字符类型数据采用 Unicode 字符集，类型标识符是 char，因此也可称为 char 类型。凡是在单引号中的一个字符，就构成一个字符常数，如下所示：

```
'a'、'o'、'*'、 '9'
```

在表示一个字符常数时，单引号内的有效字符必须且只能有一个，并且不能是单引号或者反斜杠(\)等。

为了表示单引号和反斜杠等特殊的字符常数，提供了转义符，在需要表示这些特殊常数的地方，可以使用转义符来替代这些特殊的字符。常用的转义字符如表 2-3 所示。

表 2-3　常用的转义字符

转义字符	含　义
\'	单引号
\"	双引号
\\	反斜杠
\0	空
\a	警告
\b	退格
\f	换页
\n	换行
\r	回车
\t	水平制表符
\v	垂直制表符

例如要向控制台输出“I’m a Chinese”，可以编写如下代码：

```
Console.WriteLine("I\'m a Chinese");        //在输入单引号时用到了\'转义字符
```

为了能表示更多的字符编码，C#用 16 位的 Unicode 字符集，可以编码 65535 个字符。一些大的符号系统(如中文)中的每个字符都能被编码，常见的 ASCII 编码则是用一个字节的低 7 位(高位为 0)进行编码，可以表示 128 个字符。ASCII 编码集是 Unicode 的一个子集。

布尔类型(bool)是用来表示真和假这两个概念的。这虽然看起来很简单，但实际应用非常广泛。布尔类型表示的逻辑变量只有两种取值：“真”和“假”。在 C#中，分别采用 true 和 false 两个值来表示。例如：

```
bool b1 = true;
bool b2 = false;
```

C#中的 bool 类型对应于 System.Boolean 结构。虽然只有两个取值，但它占 4 个字节。

说明： 布尔类型和其他数值之间不存在任何对应关系。不能认为整数 0 是 false，其他值是 true，这是和 C 以及 C++的区别。故 bool x=1 是错误的。

当在程序设计中需要定义一些具有整型赋值范围的变量(如星期、月份等)时，可以用枚举类型来定义。枚举将变量所能赋的值一一列举出来，给出一个具体的范围。

枚举类型用关键字 enum 来说明，定义如下：

```
enum 枚举名
{
枚举常量 1[=整型常数],
枚举常量 2[=整型常数],
...
```

```
枚举常量 n[=整型常数],
};
```

下面是一个定义枚举类型的例子：

```
enum WeekDay
{Sun,Mon,Tue,Wed,Thu,Fri,Sat};
```

上面的语句中定义了一个名称为 WeekDay 的枚举类型，它包含 Sun、Mon、Tue、Wed、Thu、Fri、Sat 这 7 个枚举成员。有了上述定义，WeekDay 本身就成了一个类型说明符，此后就可以像常量那样使用这些符号。

注意： 两个枚举成员不能完全相同。

在定义的枚举类型中，每一个枚举成员都有一个相对应的常量值，如前面定义的名为 WeekDay 的枚举类型中，其枚举成员 Sun、Mon、Tue、Wed、Thu、Fri 和 Sat 在执行程序时，分别被赋予整数值 0、1、2、3、4、5 和 6。

对于枚举成员对应的常量值，默认情况下，C#规定第 1 个枚举成员的值取 0，它后面的每一个枚举成员的值自动加上 1 递增。

在编写程序时，也可根据实际需要为枚举成员赋值。

每个枚举成员都有一个与之对应的常量值，在定义枚举类型时，可以让多个枚举成员具有同样的常量值，如下面的代码所示：

```
enum color
{yellow,brown=2,blue,red=blue,black};
```

yellow 对应整数 0，brown 对应整数 2，blue 对应整数 3，red 对应整数 3，black 对应整数 4。

定义了枚举型变量后，可以给枚举型变量赋值，需要注意的是，只能给枚举变量赋枚举常量，或把相应的整数强制为枚举类型再赋值。如例 2-1 所示。

例 2-1 定义一个表示星期的枚举类型，输出枚举变量的值：

```
namespace demo02_enum
{
    class Program
    {
        enum week
        {
            Sunday = 7, Monday = 1, Tuesday, Wednesday,
            Thursday, Friday, Saturday
        }
        static void Main(string[] args)
        {
            week w;
            w = week.Monday;
            w = (week)2;
            w = (week)(w + 2);
            Console.WriteLine("Today is " + w + ".");
```

```
            Console.ReadKey();
        }
    }
}
```

运行结果如图 2-2 所示。

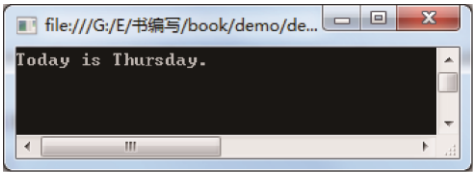

图 2-2　表示星期的枚举类型

利用上面介绍过的简单类型，进行一些常用的数据运算、文字处理似乎已经足够了。但是会经常遇到一些更为复杂的数据类型。比如，通信录的记录中可以包含他人的姓名、电话和地址。如果按照简单类型来管理，每一条记录都要存放到 3 个不同的变量中，这样工作量很大，也不够直观。有没有更好的办法呢？

在实际生活中，经常会把一组相关的信息放在一起。编程时，把一系列相关的变量组织成为一个单一实体的过程，称为生成结构的过程。这个单一实体的类型就叫作结构类型。结构类型的变量采用 struct 来进行声明。例如，可以定义通信录记录结构：

```
struct PhoneBook
{
    public string name;
    public string phone;
    public uint age;
    public string address;
}
PhoneBook p1;
```

上面声明的 p1 就是一个 PhoneBook 结构类型的变量。public 表示对结构类型的成员的访问权限，有关访问的细节问题，我们将在后续单元中详细讨论。对结构成员的访问通过结构变量名加上访问符“.”，再跟成员的名称来实现。如以下例子中定义了一个表示学生基本信息的结构类型。

例 2-2　定义一个表示学生基本信息的结构类型：

```
namespace demo02_struct
{
    class Program
    {
        struct Student
        {
            public int stuNo;
            public int age;
            public double score;
        }
```

```
        static void Main(string[] args)
        {
            Student mike;
            mike.stuNo = 168;
            mike.age = 20;
            mike.score = 93;

            Console.WriteLine("Mike's info:");
            Console.WriteLine("No: " + mike.stuNo);
            Console.WriteLine("Age: " + mike.age);
            Console.WriteLine("Score: " + mike.score);
            Console.ReadKey();
        }
    }
}
```

程序运行结果如图 2-3 所示。

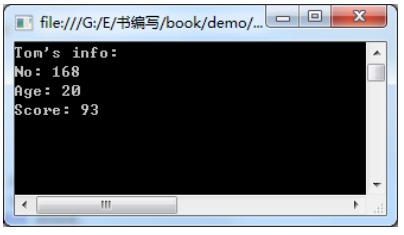

图 2-3　用结构类型表示学生信息

2. 引用类型

引用类型的变量又称为对象，可以存储对实际数据的引用。C#支持两个引用类型，如表 2-4 所示。

表 2-4　预定义的简单引用类型

名　称	CTS 类型	说　明
object	System.Object	根类型，CTS 中的其他类型都是从它派生来的
string	System.String	Unicode 字符串

在 C#的统一类型系统中，所有类型(预定义类型、用户定义类型、引用类型和值类型)都是直接或间接从 System.Object 继承的。这是 C#的一个重要特性。

string 类型表示零或更多 Unicode 字符组成的序列，它等同于.NET 中的 System.String 类，该类提供了很多内置方法，可以轻松地实现对字符串的一些基本操作。

说明： string 类型是一个引用类型，该类型数据保存在堆上。因此，当把一个字符串变量赋值给另一个字符串时，会得到对内存中同一个字符串的两个引用。

例 2-3　判断字符串是否相等，代码如下：

```
namespace demoIsStringEqual
{
    class Program
    {
        static void Main(string[] args)
        {
            string a = "hello";       //声明 string 类型变量 a
            string b = a;         //将 a 赋予 b，实际上是将 a 的引用地址赋给 b
            string c = "h";           //声明 string 类型变量 c
            c += "ello";
            //比较两个引用是否相等
            Console.WriteLine((object)a == (object)b);
            Console.WriteLine((object)a == (object)c);
            //比较 a、c 两个对象的值是否相等
            Console.WriteLine(a == c);

            Console.ReadKey();
        }
    }
}
```

本例判断字符串是否相等，声明了三个 string 类型的变量。先将变量 a 初始化为字符串“hello”，再将 a 赋值给 b，这时，实际上是把变量 a 的引用赋给了 b，也就是说，赋值完成后，a 和 b 是指向同一个地址的。然后声明 string 类型变量 c，虽然它的字面量也是“hello”，不过，它和 a、b 不是指向同一地址。程序运行结果如图 2-4 所示。

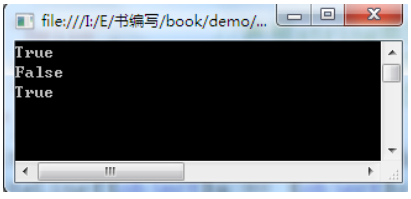

图 2-4　程序运行结果

从输出结果可以看出，变量 a 和 b 是指向同一个地址的， c 则不同，它是重新声明的一个字符串变量，有自己的地址，这个地址不同于 a 里存储的地址信息。

说明： 尽管 string 是引用类型，但定义相等运算符(==)是为了比较 string 对象(而不是引用)的值。这使得对字符串相等性的测试更为直观。

2.2.2　变量与常量

1. 常量

常量是指那些基于可读格式的固定数值，在程序的运行过程中其值是不可改变的。通过关键字 const 来声明常量，其格式如下：

```
const 类型标识符 常量名 = 表达式;
```

例如：

```
const double PI = 3.14159265;
```

上面的语句就定义了一个 double 类型的常量 PI，它的值是 3.14159265。

定义常量时，表达式中的运算符对象只允许出现常量，不能有变量存在。例如：

```
int a = 20;
const int b = 30;
const int c = b + 25;        //正确，因为 b 是常量
const int k = a + 45;        //错误，表达式中不允许出现变量
c = 150;                  //错误，不能修改常量的值
```

2. 变量

C#是一种“强类型”编程语言，在声明变量时必须指明它的数据类型。声明变量的作用之一是告诉编译器要为变量分配多少内存空间。就像要将一个物品存进储物间，应该事先让保管员知道，这个物品有多大，以便分配合适大小的储物柜。大了会浪费空间，小了东西放不下，会造成不必要的错误。变量是存储数据的一个基本单元，主要有变量名、变量类型和变量值三方面的含义，可以用如图 2-5 所示的对应例子来帮助理解。变量就像房间一样，通过内存中小房间的别名找到数据存储的位置。

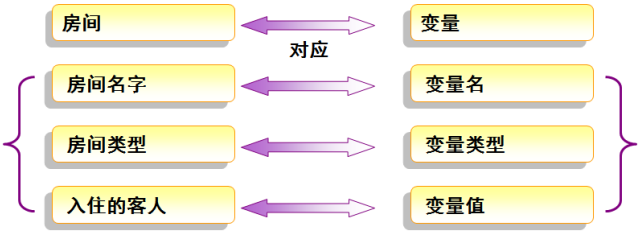

图 2-5　变量的含义

声明变量的格式如下：

```
数据类型 变量名;
```

比如，下面的代码声明一个 int 型的变量 i：

```
int i;
```

变量声明后，可在程序运行中给变量赋值，或者可以在声明的时候给变量赋初值。一个变量声明以后可以多次赋值。

在初始化时对变量赋值采用下面的格式：

```
数据类型 变量名 = 初始值;
```

例如：

```
double total = 34.3D;
```

注意： 给 double 数据类型赋值的时候，可以在数值后面加 d 或 D 表示，而 float 类型用 f 或 F 表示，decimal 类型用 m 或 M 表示。

如果在一个语句中声明和初始化了多个变量，那么所有的变量都具有相同的数据类型，例如：

```
int x=10, y=20;      //x 和 y 都有相同的数据类型 int，但是它们的值不同
```

给 bool 数据类型赋值要用 true 或 false，例如：

```
bool validateInput = true;
```

变量的初始化是 C#强调安全性的另一个方面。C#编译器需要用某个初始值对变量进行初始化，之后才能在操作中引用该变量。如下面的一段语句，声明了两个变量 i 和 j，并将 i 初始化成 12，再将 i 和 j 相加，把结果存入 i 中：

```
int i, j;   //定义变量
i = 12;   //给变量赋值
i = i + j;
```

在编译时，会有错误提示：使用了未赋值的局部变量“j”，要消除这种错误，只需要给 j 一个明确的赋值就可以了，比如将 j 赋值成 10。

3. 变量的命名规范及编码规则

在 C#中，对变量的命名有一些限制，包括以下规则：

- 变量名必须以字母或下划线开头。
- 变量只能有字母、数字、下划线，不能包含空格、标点等，且不能由数字开头。
- 变量名不得与 C#中的关键字同名。
- 变量名不得与 C#中的库函数同名。

下面给出了一些合法和不合法的变量名：

```
string 3str;             //不合法，以数字开头
float total count;       //不合法，变量名包含空格
int prod2;               //合法
double Main;             //不合法，与 Main 函数同名
double float;            //不合法，float 是关键字，不能用作变量名
```

关键字也被称为保留字，是 C#中有特殊用途的一些英文单词，不能再用做标识符。根据关键字的不同，其用途也不同，这其中既有用于声明变量的类型别名(如 int、float)，也有用于表示特定语句的关键字(如 if、while 等)。本书的后续部分将逐渐学习到大部分关键字的用法。

C#中的标识符不能与关键字相同，但是可以使用“@”前缀来避免这种冲突。例如：

```
@while
while
```

上面两个标识符中，第一个标识符是合法的，而第二个标识符不是合法的，因为 while 是关键词。

变量名除了合法性之外，还要考虑变量名的清晰性。对变量的命名最好能做到见名知意。一个好的变量名，能让人很快地知道这个变量的作用。比如，用来表示售货员的标识

符使用 salesman 比用 people 更容易理解。

多数变量命名采用 Camel 命名方法，即首字母小写后，其后续每个单词的首字母均大写。例如，下面的变量名就是用 Camel 命名方法来命名的：

```
string stuName;
float totalCount;
int productId;
```

说明： 对于变量、类名、方法名等都有不同的命名约定，包括骆驼(Camel)命名法、帕斯卡(Pascal)命名法等。

2.2.3 数据类型间的转换

不同的数据有不同的数据类型，那么，不同类型之间可以转换吗？答案是肯定的。

所有值类型和引用类型都由一个名为 object 的基本类发展而来。在 C#中还可以通过隐式转换(不会造成数据丢失)或显式转换(可能造成数据丢失或降低精确度)来改变数据类型。

1. 装箱和拆箱

任何值类型、引用类型可以和 object 类型之间进行转换。装箱是值类型到 object 类型或到此值类型所实现的任何接口类型的隐式转换，拆箱是从 object 类型到值类型或从接口类型到实现该接口的值类型的显式转换。简言之，装箱就是将值类型转换为引用类型；反之就是拆箱。

例 2-4 简单的装箱操作如下：

```
namespace demo02_BoxingExample
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 10;
            object obj = i;                 //隐式装箱
            object obj2 = (object)i;        //显式装箱
            if(obj is int)                  //is 运算符检查对象是否与特定的类型兼容
            {
                Console.WriteLine("OK");
            }
            Console.WriteLine(obj.GetType());  //返回当前对象的类型
            Console.ReadKey();
        }
    }
}
```

运行效果如图 2-6 所示。

说明： 要判断对象是否与某个给定的类型兼容，用 is 运算符；如果要返回一个类型的字符串，可以用 object 类的 GetType()方法。

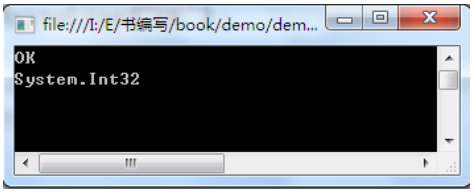

图 2-6　装箱操作

拆箱是装箱的逆过程。一般拆箱过程分成两步：首先，检查这个对象实例，看它是否为给定的值类型的装箱值；然后，把这个实例的值复制给值类型的变量。比如：

```
int i = 10;
object obj = i;     //隐式装箱
int j = (int)obj;       //拆箱
```

注意： 在拆箱时，必须非常小心，要确保得到的值变量有足够的空间存储拆箱值中的所有字节。

在实现拆箱的时候，如果待拆的对象无法转换为给定的值类型，则会引发异常。比如下面的代码，long 类型无法转换为 int 类型，就会发生异常：

```
long i = 1234567;
object j = (object)i;       //装箱
int k = (int)j;         //在拆箱时会发生异常
```

2. 隐式转换

隐式转换是系统默认的，不需要加以声明就可以进行转换。

在隐式转换过程中，编译器不需要对转换进行详细的检查就能安全地执行转换，例如数据从 int 类型到 long 类型的转换。比如从 int 类型转换到 long 类型就是一种隐式转换。隐式转换一般不会失败，转换过程中也不会导致信息丢失。方法如下所示：

```
int a = 10;         //a 为整型数据
long b = a;         //b 为长整型数据
double c = a;       //c 为双精度浮点型数据
```

3. 显式转换

显式转换又称为强制类型转换，与隐式转换相反，显式转换需要用户明确地指定转换类型。显式转换可以将一数值类型强制转换成另一种数据类型，其格式如下：

```
(类型标识符)表达式
```

上式的含义为：将表达式值的类型转换为类型标识符的类型。比如：

```
(int)5.17           //把 double 类型的 5.17 转换成 int 类型
```

如果需要在数字和字符串之间转换，不能使用上面强制转换表达式，而需要使用.NET 架构提供的 ToString 和 Parse 方法。

整型、浮点型、字符型和布尔类型都对应地有一个结构类型，该结构类型中提供 Parse 方法，可以把 string 类型转换成相应的类型。

例如，要把 string 类型转换成 int 类型，则有相应的 int.Parse(string)方法，如：

```
string str = "123";
int i = int.Parse(str);
```

则 i 的值为 123。

注意： 如果不能将指定的字符串转换成数值类型，如要把字符串“你好”转换成整数，Parse 方法会自动抛出异常。

计算后的数据如果要以文本的方式输出，如在文本框中显示计算后的数据，则需要将数值数据转换成 string 类型，转换方法是执行 ToString 方法。例如：

```
int j = 5 * 8;
string str = "5 * 8 的积是: " + j.ToString();
```

除了使用相应类的 Parse()方法之外，还可以使用 System.Convert 类的对应方法将数字转换为相应的值。例如上述例子也可以写成：

```
string str = "123";
int i = Convert.ToInt32(str);

int j = 5 * 8;
string str = "5 * 8 的积是: " + Convert.ToString(j);
```

Convert 类的静态方法用于支持 .NET Framework 中与基数据类型之间的转换。更多方法可以通过 IDE 查看 Convert 类的成员。

2.2.4 DateTime

DateTime 是用来表示时间和处理时间的类型，属于结构类型。下面通过示例来看它的基本用法。

例 2-5 用户输入一个日期，要求输出这个日期是星期几和在这一年中的第几天：

```
namespace demo02 DateTime
{
    class Program
    {
        static void Main(string[] args)
        {
            //声明一个 DateTime 类型的变量，用于存放用户输入的日期
            DateTime dt;
            Console.WriteLine("请输入日期: (例如:2000-01-01 或 2000/01/01)");
            //把输入的日期字符串转换成日期格式类型
            dt = DateTime.Parse(Console.ReadLine());
            //因为 DayOfWeek 返回的是 0、1、2、3、4、5、6，
            //分别对应的是日、一、二、三、四、五、六
```

```
            //Substring 是检索字符串并返回匹配的指定长度的子字符串
            string str = "日一二三四五六".Substring((int)dt.DayOfWeek, 1);
            Console.WriteLine(
              "{0}年{1}月{2}日是星期{3}", dt.Year, dt.Month, dt.Day, str);
            Console.WriteLine("{0}年{1}月{2}日是这一年的第{3}天",
              dt.Year, dt.Month, dt.Day, dt.DayOfYear);
            Console.WriteLine("{0}是星期{1}", dt.ToShortDateString(), str);
            Console.WriteLine(
              "{0}是这一年的第{1}天", dt.ToLongDateString(), dt.DayOfYear);
            Console.ReadKey();
        }
    }
}
```

运行结果如图 2-7 所示。

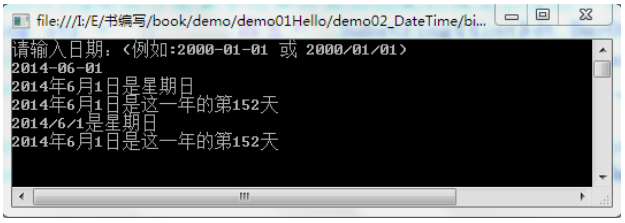

图 2-7　利用 DateTime 来计算天数

例 2-6　显示当前日期和时间的不同格式(运行结果见图 2-8)：

```
namespace demo02_DateTime
{
    class Program
    {
        static void Main(string[] args)
        {
            //以不同的格式显示当前日期和时间
            Console.WriteLine(
              "现在时间是：{0}", DateTime.Now.ToString("yyyy-M-d H:m:s"));
            Console.WriteLine("现在时间是：{0}",
              DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"));
            Console.WriteLine(
              "现在时间是：{0}", DateTime.Now.ToString("yyyy-MM-dd"));

            Console.WriteLine("短日期字符串表示现在时间是：{0}",
              DateTime.Now.Date.ToShortDateString());
            Console.WriteLine("长日期字符串表示现在时间是：{0}",
              DateTime.Now.Date.ToLongDateString());

            Console.ReadKey();
        }
    }
}
```

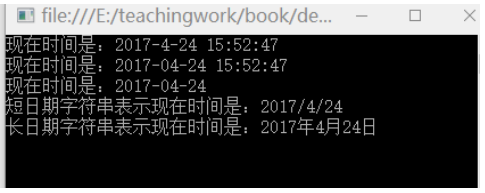

图 2-8　当前日期和时间的不同格式

2.3　案例分析与实现

2.3.1　案例分析

针对本单元所给出的案例，分析如下。

(1) 因为需要处理至少两种客户类型，所以定义一个 CUSTOMERCATEGORY 枚举类型，用于表示客户的账号类型。在其中列举两个枚举常量：SAVINGACCOUNT 和 CHECKINGACCOUNT，分别表示储蓄账户客户和支票账户客户。

(2) 定义结构体类型 Customer，用于表示客户的账户信息，包含结构体成员变量。

(3) 定义表示当前可用客户账号的变量、表示客户账号的变量、表示客户姓名的变量、表示客户地址的变量、表示客户电话号码的变量、表示客户账号余额的变量。为它们确定合理的数据类型。

(4) 当创建一个新的客户账号时，则定义一个结构体 Customer 的变量，由控制台输入当前客户的各项信息，按需求执行数据类型的转换和数据处理，并将输入的客户信息打印输出。

2.3.2　案例实现

案例的实现代码如下：

```
namespace demo02_DataType
{
    class Program
    {
        /// <summary>
        /// 客户账号类型，枚举类型
        /// </summary>
        enum CUSTOMERCATEGORY
        {
            /// <summary>
            /// 储蓄账户客户
            /// </summary>
            SAVINGACCOUNT = 1,
            /// <summary>
            /// 支票账户客户
            /// </summary>
```

```
        CHECKINGACCOUNT
    }

    /// <summary>
    /// 客户账户信息，结构体类型
    /// </summary>
    struct Customer
    {
        /// <summary>
        /// 当前可用客户账号，最小为 500000
        /// </summary>
        static public int currentAccountNumber = 500000;

        /// <summary>
        /// 客户账号
        /// </summary>
        public int accountNumber;

        /// <summary>
        /// 客户账户类型
        /// </summary>
        public CUSTOMERCATEGORY accountCategory;

        /// <summary>
        /// 客户姓名
        /// </summary>
        public string name;

        /// <summary>
        /// 地址
        /// </summary>
        public string address;

        /// <summary>
        /// 电话号码
        /// </summary>
        public int phone;

        /// <summary>
        /// 账号余额
        /// </summary>
        public double balance;
    }
    static void Main(string[] args)
    {
        Customer currentCustomer;
        currentCustomer.accountNumber =
          Customer.currentAccountNumber + 1;
        Console.WriteLine("请输入当前客户的详细信息：");
        Console.Write(
```

```
            "请选择客户的账户类型(1、储蓄账户客户；2、支票账户客户)：");
          currentCustomer.accountCategory =
            (CUSTOMERCATEGORY)Convert.ToInt32(Console.ReadLine());
          Console.Write("请输入客户姓名(最多 30 个字符，最少 2 个字符)：");
          currentCustomer.name = Console.ReadLine();
          Console.Write("请输入客户地址(最多 80 个字符，最少 2 个字符)：");
          currentCustomer.address = Console.ReadLine();
          Console.Write("请输入客户电话号码(40000000-99999999)：");
          currentCustomer.phone = Convert.ToInt32(Console.ReadLine());
          Console.Write("请输入客户账户余额(大于 0 的数)：");
          currentCustomer.balance = Convert.ToDouble(Console.ReadLine());

          Console.WriteLine("\n 当前客户的详细信息显示如下：");
          Console.WriteLine("客户账号：" + currentCustomer.accountNumber);
          Console.WriteLine(
            "账户类型：" + currentCustomer.accountCategory);
          Console.WriteLine("客户姓名：" + currentCustomer.name);
          Console.WriteLine("客户地址：" + currentCustomer.address);
          Console.WriteLine("客户电话号码：" + currentCustomer.phone);
          Console.WriteLine("客户账户余额：" + currentCustomer.balance);
          Console.ReadKey();
      }
   }
}
```

运行后，即可见本单元 2.1 节中的演示效果。

2.4 拓 展 训 练

2.4.1 拓展实训 1：使用变量存储一部手机的信息

请按照下面给出的手机信息，存储并打印输出。

(1) 品牌(brand)：苹果 6S Plus。

(2) 重量(weight)：192g。

(3) 后置摄像头像素(camera)：120,000,000px。

(4) 价格(price)：6288。

分析：编程时，请注意各变量数据类型的选取，数值类型需要考虑数值范围。

由于实训题比较简单，具体实现由读者完成。

2.4.2 拓展实训 2：数字加密器

请实现一个数字加密器，要求如下：加密的结果=(需加密的整数×3+8)/2+3.14，加密的结果要求仍然为整数，取计算后结果的整数部分。

分析：需要进行数据类型的转换，考虑应该用隐式转换还是强制转换。

由于实训题比较简单，具体实现由读者完成。

习　题　二

(1) C#变量合法命名的是(　　)。

A. $abcc　　B. 12_223　　C. myString　　D. .123aab

(2) C#变量采用的命名法是(　　)。

A. Pascal　　B. Camel　　C. 变量命名法　D. 其他命名法

(3) 能代表 C#自定义命名空间的关键字是(　　)。

A. class　　B. public　　C. using　　D. namespace

(4) C#引用系统命名空间如 System.Text，是使用哪一个关键字？(　　)

A. class　　B. public　　C. using　　D. namespace

(5) C#中转义符用于换行的是(　　)。

A. \t　　B. \r　　C. /t　　D. /r

(6) C#中每个 int 类型的变量占用(　　)个字节的内存。

A. 1　　B .2　　C .4　　D .8

(7) 在 C#中，表示一个字符串的变量应使用以下哪条语句定义？(　　)

A. CString str;　　B. string str;

C. Dim str as string　　D. char *str;

(8) 在 C#编制的财务程序中，需要创建一个存储流动资金金额的临时变量，则应使用下列哪条语句？(　　)

A. decimal theMoney;　　B. int theMoney;

C. string theMoney;　　D. Dim theMoney as double

(9) C#中，新建一个字符串变量 str，并将字符串“Tom’s Living Room”保存到串中，则应该使用下列哪条语句？(　　)

A. string str =“Tom\'s Living Room”;　　B. string str =“Tom's Living Room”;

C. string str(“Tom's Living Room”);　　D. string str(“Tom"s Living Room”);

(10) 将变量从字符串类型转换为数值类型可以使用的类型转换方法是(　　)。

A. Str()　　B. Cchar　　C. CStr()　　D. int.Parse();

(11) 数据类型转换的类是(　　)。

A. Mod　　B. Convert　　C. Const　　D. Single