

资深软件开发专家撰写，系统且深入阐释VSTO开发涉及的工具、方法和实践

由浅入深剖析VSTO开发过程中遇到的各个层面的问题，涉及Visual Studio、C#开发、创建Office外接程序、自定义Office功能区、任务窗格、自定义工具栏等



VSTO Development Introductory Tutorial

VSTO

开发入门教程

C# & VBA双语对照版

刘永富◎著
Liu Yong Fu



清华大学出版社

购买此电子书，不提供光盘\视频内容，敬请谅解



VSTO Development Introductory Tutorial

VSTO

开发入门教程

刘永富◎著
Liu Yong Fu

清华大学出版社
北京

内 容 简 介

本书从初学者角度出发,详细介绍了使用C#语言进行VSTO开发需要掌握的知识。全书分为12章,内容包括VSTO入门概述、C#语法基础、C#进阶技术、C#操作Excel对象、创建Office外接程序、自定义Office功能区、自定义任务窗格、自定义工具栏、VSTO外接程序的部署分发、VSTO开发Office文档、VSTO开发资源大全、C#与VB/VBA语言的差异对比。书中所有章节涉及的程序代码都给出了详细的注释。本书可以让读者轻松熟悉Visual Studio开发环境,跨入C#编程的门槛,掌握VSTO开发的步骤。

本书可作为职场办公人员、高校理工科师生、Office专业开发人员自学用书,也可以作为Office编程培训讲师的教学参考书。

本书封面贴有清华大学出版社防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

图书在版编目(CIP)数据

VSTO开发入门教程 / 刘永富著. — 北京:清华大学出版社, 2017
ISBN 978-7-302-45371-0

I. ①V… II. ①刘… III. ①BASIC语言—程序设计—教材 IV. ①TP312.8

中国版本图书馆 CIP 数据核字(2016)第 260921 号

责任编辑:杨如林 秦 健

封面设计:李召霞

责任校对:胡伟民

责任印制:沈 露

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

印 装 者:北京国马印刷厂

经 销:全国新华书店

开 本:186mm×240mm 印 张:15.25 字 数:372 千字

版 次:2017 年 1 月第 1 版 印 次:2017 年 1 月第 1 次印刷

印 数:1 ~ 3500

定 价:45.00 元

产品编号:071876-01

前言

VSTO是指Visual Studio Tools for Office，其含义是在Visual Studio开发环境中进行Office专业开发。

Visual Studio是目前最流行的Windows平台应用程序的集成开发环境。VSTO是VBA的替代，使得开发Office应用程序更加简单，并且用VSTO来开发Office应用程序可以使用Visual Studio开发环境中的众多功能。

VSTO是一套用于创建自定义Office应用程序的Visual Studio工具包，可以用Visual Basic或者Visual C#扩展Office应用程序（例如Word、Excel、PowerPoint和Outlook）。正是由于VSTO具有诸多优势，吸引了越来越多的人开始转入研究VSTO开发，但是到目前为止，国内还没有一本比较适合初学者的入门教材，网络上查到的相关知识点也比较片面。本书是国内第一部关于VSTO开发Office的经典书籍，作者根据自己的开发经验，将开发过程中的关键技术和要点都融入本书。全书共12章，作者从读者的角度考虑，编排了从Visual Studio的安装、C#语言入门，一直到创建第一个完整的VSTO项目，基本是手把手地告诉读者每一个环节。读者阅读本书后，可以自行开发完整的VSTO项目，并制作成可以分发的安装包。最重要的是，通过本入门教程的学习，能让读者增强信心，产生进一步深入研究和探索VSTO的兴趣。

也许很多读者朋友看到诸如Visual Studio、C#这样的字眼望而生畏，其实VSTO并没有想象的那么难，只要按照本教程按部就班学习，结合视频教程的讲解，很快就能掌握这项开发技术。

VSTO学习路线图

对于VSTO的初学者，可以按照如下的路线图进行系统学习。如果是已经具有Visual Studio和C#基础的读者，则可以直接从第三阶段学起。

VSTO课程学习路线图

第一阶段：熟悉Visual Studio编程环境

- 理解VSTO的概念
- 安装Visual Studio
- 熟悉解决方案和项目文件夹

第二阶段：熟悉C#语言

- 创建C#窗体应用程序、窗体与常用控件的使用
- C#语法基础（变量、控制语句、不同数据类型的转换等）
- 程序代码调试、错误处理
- 使用类

第三阶段：C#操作和控制Excel对象

- 熟悉Excel对象模型，Excel对象的常用属性、方法和事件
- 加强从VBA代码向C#代码迁移的能力

第四阶段：界面设计部分

- 定制功能区：使用功能区设计器或使用XML代码，理解Custom UI机制
- 任务窗格：任务窗格中控件和用户控件的添加
- 创建文档自定义项，使用文档操作窗格
- 自定义工具栏

第五阶段：VSTO项目分发与安装程序的制作

- 使用Advanced Installer

其他知识点

- 使用C#制作Excel自定义函数（UDF）
- 创建Word、PPT等组件的VSTO项目

本书内容

本书内容以VSTO学习路线图为依据编排而成，全书共分12章。

第1章：VSTO入门概述

本书介绍的是一项程序开发技术，因此首先要让读者安装必要的程序语言和开发环境。然后讲述VSTO开发的意义和任务，以及创建和调试C#解决方案的方法与步骤。

第2章：C#语法基础

本书以C#为开发语言，因此读者需要掌握一定程度的C#语法基础。该章介绍了变量的声明和赋值、流程控制与类的使用。

第3章：C#进阶技术

该章讲述了C#窗体与控件的相关技术，以及像正则表达式、字典等高级对象的用法，目的是让读者在开发过程中，能够熟练应用这些高级对象去处理实际问题。

第4章：C#操作Excel对象

前面3章讲述的是纯粹的C#语言，而从这一章开始，讲述如何使用C#操作和控制Office对象，该章以Excel为例，介绍了Excel的应用程序、工作簿、工作表、单元格区域等对象的成员。

第5章：创建Office外接程序

VSTO开发的旨在创建Office外接程序（即COM加载项）。该章介绍了COM加载项的工作原理和开发基本步骤。

第6章：自定义Office功能区，第7章：自定义任务窗格

第6章和第7章分别介绍了VSTO开发的重点，一般来说，创建一个Office外接程序，界面定制是非常必要的，为此本书在这两章里详尽地介绍了功能区的自定义技术和自定义窗格的设计方法。

第8章：自定义工具栏

Office工具栏是Office组件中很重要的一个界面对象，为此本书通过典型的实例，讲述了工具栏和控件的自定义方法。

第9章：VSTO外接程序的部署分发

VSTO开发的成品，一般需要能够在其他计算机上正常使用，为此，该章介绍了使用Advanced Installer软件来创建VSTO项目的安装程序。

第10章：VSTO开发Office文档

文档自定义开发是VSTO另一类型的项目。该项目允许自定义文档，可以向文档中加入C#控件，以及创建和控制文档窗格。

第11章：VSTO开发资源大全

“工欲善其事，必先利其器”，为了能够驾轻就熟地进行VSTO开发，还需要使用其他一些工具的辅助，为此，该章介绍了典型工具的安装和使用技巧。

第12章：C#与VB/VBA语言的差异对比

考虑到很多读者是从VBA转过来的，对VBA的语法和对象模型更为熟悉，为了能够帮助读者更快地从VBA转入VSTO，该章列出了两种语言典型的语法差异。

本书特点

本书是目前市面上稀缺而Office开发人员急需的、Office和C#技巧完美融合的经典书籍，为了让读者快速了解和熟悉VSTO，本书第2~4章的C#代码都配备了对应的VBA代码，可以让之前从事VBA开发的读者迅速学会VSTO开发。同时本书配套资源中包括本书涉及的所有项目的源文件，以便读者加以验证和核对。另外，本书配套资源中还有与VSTO开发相关的全部有声视频教程。

本书配套资源内容说明

本书配套资源包括VSTO开发入门视频教程、本书所有示例程序、VSTO开发资源大全三大部分内容。关于本书配套资源，读者可访问<http://vba.mahoupao.net/forum.php?mod=viewthread&tid=2407&fromuid=1>进行下载。

大分类	文件名	对应章节或描述
VSTO开发 入门视频教程	VSTO概述.wmv	第1章
	C#语法基础.wmv	第2章
	类的创建和使用.wmv	2.12节
	窗体和控件的设计技术.wmv	3.7节
	C#操作Excel对象.wmv	第4章
	创建Office外接程序.wmv	第5章
	使用Ribbon设计器自定义Office功能区.wmv	6.4节
	使用XML自定义Office功能区.wmv	6.5节
	自定义任务窗格的设计.wmv	第7章
	VSTO外接程序的打包.wmv	第9章
	VSTO开发Office文档-文档操作窗格.wmv	第10章
示例程序	ConsoleApplication20160629	1.2.1节
	ExcelAddIn20160514	6.4节
	ExcelAddIn20160515	6.5节
	ExcelAddIn20160516	7.2节
	ExcelAddIn20160517	8.2.1节
	ExcelWorkbook20160519	10.3节
	Solution20160705	1.4节
	UDF20160521	4.9.1节
	VSTOBOOK-C#	第2章
	VSTOBOOK-VB	第2章
	WindowsFormsApplication20160522	4.2节
	WindowsFormsApplication20160523	3.4节

大分类	文件名	对应章节或描述
示例程序	WindowsFormsApplication20160524	3.5.4节
	WindowsFormsApplication20160525	3.6节
	WindowsFormsApplication20160526	3.7节
	WindowsFormsApplication20160527	3.7.9节
	WindowsFormsApplication20160528	3.8节
	WindowsFormsApplication20160606	2.2节
	WindowsFormsApplication20160625	2.12节
	WindowsFormsApplication20160629	1.2.2节
VSTO开发 资源大全	OfficeCommandbarDesigner20160709.rar	11.1节
	OfficeCommandbarViewer20160709.rar	
	FaceIDs_V2_20160709.xls	
	FaceIDs_V2_20160709.doc	
	Office2010ControlIDs.rar	11.2节
	imageMso7345.xlsm	
	OfficeCustomUIEditorSetup.msi	
	RibbonXMLEditor20160709.rar	
	ribbon回调函数大全.xlsm	3.8节
	UseAPI.rar	
	VBE2014_Setup_20160709	11.3节
	VisualStudioAddin2016Setup.exe	

读者对象

- 职场办公人员
- 理工科类大学生、研究生
- 编程爱好者
- 培训机构的老师和学员

本书约定

书中述及的多级菜单和工具栏的图示中，鼠标单击的各级菜单或命令均放在中文方括号之中，各级之间以斜杠隔开。例如【文件/打开】表示连续单击了“文件”菜单的“打

开”子菜单。

书中所有的VBA和C#代码段，代码左侧均有行号，这些行号只是为了便于讲解，并不属于代码部分。

读者服务

为了方便本书内容答疑，读者朋友可加入VBA/VSTO开发QQ群：61840693，也可以在VBA/VSTO论坛（<http://vba.mahoupao.net/forum.php>）发帖，还可以直接给作者发Email：lyflyf715@sina.com。无论哪一种方式，作者将竭诚为您服务。

如果要进一步学习Office、VBA、VSTO等学科的视频课程，读者可在51CTO学院搜索作者主讲的相关课程：http://edu.51cto.com/user/user_id-6673733.html。

致谢

感谢刘爱珍、儒道佛潘淳、西西老师、张杰、闻启学等朋友以及兄长刘永和在本书编写过程中给予的无私帮助和鼓励。

本书在出版过程中，得到了清华大学出版社策划编辑秦健先生的大力支持和配合，在此表示衷心感谢。另外，本书所有的编审、发行人员为本书的出版和发行付出了辛勤劳动，在此一并致谢。

特别说明

本书编写时所用的VSTO开发环境如下：

- 操作系统：Windows 7（32bit）
- Office：Office 2010完整版
- Visual Studio：Visual Studio 2012
- 开发语言：C#

读者可以根据自身条件适当调整。

另外，本书涉及的所有VSTO示例，均以Excel 2010为开发对象，对于其他Office组件的开发，过程非常类似，读者可以在Excel开发的基础上自行探索。

致读者

微软Office套件称得上是全世界最成功的办公软件，拥有非常多的用户。它之所以受到人们的青睐，有多方面的原因，但是以下几点是有目共睹的：一是功能完善而且强大；

二是容易操作，用户容易学会；三是具有强大的编程开发功能。

随着计算机的发展，以往的手工操作办公软件已经不能满足现代办公的需要，因此，VBA以及VSTO开发和应用技术应运而生。本书在编写过程中，受到了众多Office开发人员的关注，他们殷切希望本书尽早出版。除了刘永富之外，参与本书编写的人员还有章晓琳、马成林、钟卓成、李四桂、何明、段留柱、高大伟、肖云、谭信章、戴海东、朱辉、徐鹏、祝磊、管洪洋、刘爱珍、王继成、汪龙、林兴龙、梁加成等。在编写过程中难免会有漏洞，欢迎读者通过清华大学出版社网站（www.tup.com.cn）与我们联系，帮助我们改正提高。

刘永富

2016年7月于北京

目 录

第1章 VSTO入门概述	1	1.4.4 引用管理	18
1.1 VSTO简述	1	1.5 使用帮助系统	19
1.1.1 VSTO的功能与特点	1	1.5.1 设置帮助查看方式	19
1.1.2 VSTO开发语言	2	1.5.2 下载和安装Help Viewer	19
1.1.3 VSTO开发环境配置	3	1.5.3 管理帮助内容	19
1.1.4 Visual Studio开发环境	4	本章要点回顾	21
1.1.5 Visual Studio版本沿革	4	第2章 C#语法基础	22
1.1.6 Visual Studio的安装	4	2.1 变量的声明和赋值	22
1.2 创建第一个C#应用程序	5	2.1.1 常用的数据类型	22
1.2.1 控制台应用程序	5	2.1.2 赋值运算符	23
1.2.2 Windows窗体应用程序	9	2.1.3 变量的作用范围	23
1.2.3 生成可执行文件	11	2.2 字符与字符串处理	24
1.3 认识Visual Studio开发环境	11	2.2.1 字符变量	24
1.3.1 【文件】菜单	12	2.2.2 字符串变量	25
1.3.2 【视图】菜单	12	2.2.3 转义字符	26
1.3.3 【项目】菜单	12	2.2.4 字符串连接	27
1.3.4 菜单栏和工具栏的自定义	13	2.2.5 子字符串	27
1.3.5 Visual Studio选项	14	2.2.6 格式化字符串	28
1.4 Visual Studio项目组织结构	14	2.2.7 字符串的替换	28
1.4.1 解决方案	14	2.2.8 字符串与数组	28
1.4.2 项目	17	2.3 逻辑运算	29
1.4.3 类模块	18	2.3.1 布尔型变量	29

2.3.2	比较运算符	30
2.3.3	多条件的与或非运算	30
2.4	不同类型的强制转换	31
2.4.1	ToString	31
2.4.2	Parse	31
2.4.3	Convert	32
2.5	使用数组	33
2.5.1	数组的声明和初始化	33
2.5.2	一维数组	33
2.5.3	数组元素的遍历	34
2.5.4	二维数组	35
2.6	条件选择语句	37
2.6.1	三元运算符	37
2.6.2	if语句	38
2.6.3	switch语句	39
2.7	循环语句	40
2.7.1	while循环	40
2.7.2	do循环	41
2.7.3	for循环	42
2.7.4	foreach循环	43
2.8	流程控制语句	43
2.8.1	break语句	43
2.8.2	continue语句	44
2.8.3	goto语句	44
2.8.4	return语句	45
2.9	输出对话框 (MessageBox)	46
2.9.1	MessageBox语法	46
2.9.2	自定义对话框的按钮	47
2.9.3	自定义对话框的图标	48
2.9.4	自定义对话框默认按钮	48
2.9.5	处理对话框的用户响应	48
2.10	输入对话框 (InputBox)	49
2.11	过程与函数	50

2.11.1	过程与函数的定义	50
2.11.2	过程与函数的调用	51
2.12	类的创建和使用	52
2.12.1	非静态类	52
2.12.2	静态类	54
2.13	using指令	55
2.14	错误处理	55
	本章要点回顾	56

第3章 C#进阶技术 57

3.1	文件与文件夹操作	57
3.1.1	System.IO命名空间	57
3.1.2	文件与文件夹处理实例	58
3.2	文本文件的读写	59
3.3	数据库操作	60
3.4	使用资源文件	61
3.4.1	添加资源文件	62
3.4.2	资源文件中的字符串	62
3.4.3	资源文件中的图像	63
3.5	使用正则表达式	65
3.5.1	创建Regex对象	65
3.5.2	元字符	65
3.5.3	正则表达式选项	66
3.5.4	正则表达式方法	67
3.5.5	正则表达式测试器	71
3.6	使用字典	72
3.6.1	字典对象的创建	72
3.6.2	根据键检索值	73
3.6.3	遍历所有键名	74
3.6.4	遍历所有值	74
3.6.5	去除重复	74
3.7	窗体设计技术	76
3.7.1	窗体的显示	76

3.7.2 窗体的卸载.....	77	4.4 操作Workbook对象.....	112
3.7.3 窗体与控件的事件.....	78	4.4.1 Workbook对象常用属性.....	112
3.7.4 使用窗体菜单.....	82	4.4.2 Workbook对象常用方法.....	113
3.7.5 使用工具栏.....	85	4.4.3 Workbook对象常用事件.....	114
3.7.6 使用右键菜单.....	87	4.4.4 Workbook重要集合对象.....	114
3.7.7 使用状态栏.....	88	4.5 操作Worksheet对象.....	115
3.7.8 使用文件选择对话框.....	90	4.5.1 Worksheet对象常用属性.....	115
3.7.9 运行期间动态增删控件.....	91	4.5.2 Worksheet对象常用方法.....	116
3.8 使用Windows API函数.....	94	4.5.3 Worksheet对象常用事件.....	117
3.8.1 窗口类名和句柄.....	95	4.6 操作Range对象.....	117
3.8.2 使用Spy++.....	98	4.6.1 Range对象常用属性.....	117
3.8.3 使用UseAPI.....	100	4.6.2 Range对象常用方法.....	118
3.8.4 获取光标位置.....	101	4.6.3 Range对象的遍历.....	119
本章要点回顾.....	101	4.6.4 二维数组与Range数据交换.....	120
		4.6.5 一维数组与Range数据交换.....	121
第4章 C#操作Excel对象.....	102	4.7 操作Commandbar对象.....	121
4.1 Excel对象模型概述.....	102	4.8 操作VBE工程.....	123
4.1.1 Application对象.....	103	4.8.1 引用VBIDE类型库.....	123
4.1.2 Workbook对象.....	104	4.8.2 允许对VBA工程访问.....	123
4.1.3 Worksheet对象.....	104	4.8.3 操作VBE各级对象.....	125
4.1.4 Range对象.....	105	4.9 创建Excel自定义函数.....	125
4.1.5 Window对象.....	105	4.9.1 使用C#创建类库.....	126
4.2 创建可以访问Excel对象的C#窗体 应用程序.....	105	4.9.2 工作表中使用C#开发的自定义 公式.....	129
4.2.1 添加Excel 2010对象引用.....	105	4.9.3 VBA中调用C#开发的自定义公式... 131	
4.2.2 添加Office 2010对象引用.....	105	4.9.4 C#中调用C#开发的自定义公式..... 131	
4.3 操作Application对象.....	107	4.9.5 客户机使用C#制作的自定义函数..... 132	
4.3.1 获取正在运行的Excel对象.....	107	本章要点回顾.....	133
4.3.2 创建新的Excel对象.....	109		
4.3.3 Application对象常用属性.....	109	第5章 创建Office外接程序.....	134
4.3.4 Application对象常用方法.....	110	5.1 Office COM加载项简介.....	134
4.3.5 Application对象常用事件.....	110	5.2 认识Office COM加载项管理 对话框.....	134
4.3.6 Application重要集合对象.....	111		

5.3 创建第一个Office外接程序项目	135
5.4 ThisAddin的启动事件和卸载事件 ..	136
本章要点回顾	137

第6章 自定义Office功能区

6.1 CustomUI概述	138
6.1.1 CustomUI的意义	140
6.1.2 CustomUI的作用范围	140
6.1.3 手工定制Office界面	140
6.2 CustomUI与XML	141
6.2.1 XML语法规则	141
6.2.2 描述Office界面的XML	142
6.2.3 使用Ribbon XML Editor	148
6.3 CustomUI元素详解	149
6.3.1 选项卡 (tab) 元素	150
6.3.2 组 (group) 元素	151
6.3.3 控件 (control) 元素	151
6.4 VSTO中使用功能区可视化 设计器	154
6.4.1 为按钮指定回调过程	156
6.4.2 Group中加入DialogBoxLauncher	157
6.5 使用XML进行CustomUI定制	159
本章要点回顾	162

第7章 自定义任务窗格

7.1 任务窗格行为控制	163
7.2 VSTO外接程序项目中添加任务 窗格	164
7.2.1 创建Excel 2010外接程序	165
7.2.2 添加用户控件	165
7.2.3 静态类中声明任务窗格对象	166
7.2.4 创建并显示任务窗格	167
7.3 功能区与任务窗格的交互控制	169

7.3.1 利用功能区切换按钮控制任务窗格的 显示隐藏	169
7.3.2 处理自定义任务窗格事件	172
7.3.3 完全卸载任务窗格	173
本章要点回顾	173

第8章 自定义工具栏

8.1 Office工具栏对象简述	174
8.1.1 Commandbar对象	174
8.1.2 CommandbarControl对象	175
8.1.3 自定义工具栏的作用和意义	176
8.2 VSTO实现自定义工具栏	176
8.2.1 创建自定义工具栏	176
8.2.2 处理工具栏按钮的回调	178
8.2.3 修改右键菜单	179
8.2.4 卸载外接程序时清除自定义	180
本章要点回顾	181

第9章 VSTO外接程序的部署分发

9.1 客户机搭建VSTO运行环境	182
9.2 VSTO外接程序的简单安装	183
9.3 使用Advanced Installer	184
9.3.1 创建aip安装包工程	184
9.3.2 客户机运行安装包	193
本章要点回顾	194

第10章 VSTO开发Office文档

10.1 文档自定义项编程概述	195
10.2 文档自定义项允许添加的界面 元素	195
10.3 创建Office文档项目	196
10.3.1 文档上添加C#控件	198
10.3.2 文档项目的启动事件过程	198

10.4 文档操作窗格概述	199	11.2.2 imageMso7345	216
10.5 文档操作窗格综合实例	200	11.2.3 OfficeCustomUIEditor	217
10.5.1 添加用户控件到文档窗格	202	11.2.4 Ribbon XML Editor	217
10.5.2 添加多个相同控件到文档窗格	204	11.2.5 Ribbon回调函数大全	217
10.5.3 使用代码创建窗体控件并添加到 文档操作窗格	205	11.3 编程环境辅助工具	218
10.5.4 定制功能区按钮控制文档操作 窗格	206	11.3.1 VBE2014	219
10.6 文档自定义项的部署分发	210	11.3.2 VisualStudioAddin2016	220
本章要点回顾	211		
第11章 VSTO开发资源大全	212	第12章 C#与VB/VBA语言的差异	
11.1 Office 2003以下版本工具栏和控件的 自定义	212	对比	222
11.1.1 OfficeCommandbarDesigner	212	12.1 变量必须声明	222
11.1.2 OfficeCommandbarViewer	213	12.2 严格的类型匹配	222
11.1.3 FaceIDViewer	213	12.3 项目的自动保存	222
11.2 Office 2007以上版本功能区的 自定义	215	12.4 严格区分大小写	223
11.2.1 Office2010ControlIDs	215	12.5 语句结束必须加分号	223
		12.6 语句块	223
		12.7 调用其他函数圆括号不能少	224
		12.8 数组的下标为0	224
		12.9 数组或集合对象的索引使用 方括号	225

第1章

VSTO入门概述

也许您之前从未接触过Office开发方面的知识，只是停留在Office办公软件的手工操作，也许您已经能够熟练使用VBA来解决实际工作中遇到的问题。本章将向读者介绍微软最新推出的Office开发工具包：VSTO。

 本章视频：VSTO概述.wmv^①

1.1 VSTO简述

VSTO（Visual Studio Tools for Office）是.NET平台下的Office开发技术。相对于传统的VBA（Visual Basic Application）开发，VSTO为中高级开发人员提供了更加强大的开发平台和语言，并部分解决了传统Office开发中的诸多问题（难以更新、可扩展性差、难以维护、安全性低等），开发人员可以使用熟悉的技术来构建更加灵活的、强大的、跨平台的企业级解决方案。

简言之，VSTO就是在Visual Studio这个开发环境中，使用C#语言开发用于微软Office的插件或文档。

■ 1.1.1 VSTO的功能与特点

对于Office解决方案开发来说，VSTO是简单但强大的框架。这个框架为每个Office开发者带来了许多令人惊叹的好处：窗体控件、类、安全性、服务器可测量性、面向对象特征、完整性、易发布，等等。

^① 关于本书配套资源，读者可访问<http://vba.mahoupao.net/forum.php?mod=viewthread&tid=2407&fromuid=1>进行下载。

1. 更安全的托管代码扩展

VSTO允许托管和非托管代码一起无缝地放在相同的.NET程序集里，这允许开发者保留非托管代码而无须完全重写。带有链接或引用托管代码程序集的文档或工作簿被作为托管代码扩展。通过使用VSTO在Word或Excel中创建托管代码扩展，与宏相似但更安全。使用VSTO能够创建仅需要装载数据的模板。

2. 自定义功能

使用可重复使用的类，VSTO 3.0提供极好的控制来自定义Office应用程序。不像VBA开发者，VSTO开发者不局限于VBA函数库。VSTO提供了相当广泛的类、对象和事件来创建Office商业解决方案。使用VSTO，开发者能够为Office应用程序自定义功能。这能够简单到在应用程序命令栏中添加按钮或自定义任务窗格，或者复杂到用于访问不同数据源的数据报表模板。

3. 自定义用户界面

VSTO提供Windows窗体控件，帮助你为Office解决方案开发富用户界面（UI）。通过使用大量各种各样的控件集，VSTO开发者能够为用户创建丰富的数据视图。每种和每类Windows窗体控件都有自己的属性、方法和事件设置，适合不同的需要。

通过在文档和任务窗格里使用控件，VSTO使创建丰富的用户界面更容易。例如，可以创建一个活泼的按钮命令产生套用信函。又如，假设公司在其服务器上存储了数据内容，用户在处理文档时想从服务器中引用一些内容并且不想离开当前编辑的文档，使用VSTO可以使服务器内容在文档的任务窗格中可用而无须干扰用户当前的工作。

4. WPF支持

WPF能用于创建丰富的、具有吸引力的外观。在VSTO环境中可使用WPF。VSTO的可视设计器支持Windows窗体和WPF控件的使用。WPF为创建基于客户和基于网络的应用程序提供了可靠的编程模型，并且在商务逻辑和UI之间呈现清楚的分离。

5. 可视化的设计器

VSTO为Office应用程序提供了可视化的设计器，例如Word 2007、Excel 2007，显示在Visual Studio IDE里。在Visual Studio IDE里创建窗体只需拖动并放置窗体到Office文档中。开发者能够访问Visual Studio IDE中的许多工具和功能，例如智能感知、拖放控件和数据源。VSTO也提供了Ribbon可视化设计器，用于通过使用简单的.NET应用程序编程模型自定义Office功能区和编程。

■ 1.1.2 VSTO开发语言

可以选择使用Visual Basic.NET或者Visual C#语言进行VSTO开发。本书只讲述以C#为

编程语言的VSTO开发技术。

■ 1.1.3 VSTO开发环境配置

进行VSTO开发需要具备如下环境：

- Windows。系统用户可以在Windows XP或Windows 7中进行VSTO开发。
- Microsoft Office。VSTO支持Office的最低版本是Office 2003。
- Visual Studio。VSTO支持的最低Visual Studio版本是Visual Studio 2005。

针对计算机系统、Office版本以及Visual Studio版本的选择，这几个方面需要根据开发内容和使用对象而定。比如要开发带有自定义功能区的VSTO项目，则要求必须安装Office 2007以上的版本。

Visual Studio和Office之间的版本兼容关系如图1.1所示。

Visual Studio Version	Office 2003	Office 2007	Office 2010	Office 2013
VS 2005	Supported	Not Supported	Not Supported	Not Supported
VS 2008	Supported	Supported	Not Supported	Not Supported
VS 2010	Supported	Supported	Supported	Not Supported
VS 2012	Supported	Supported	Supported	Supported
VS 2013	Supported	Supported	Supported	Supported

图1.1 Visual Studio与Office版本对应关系

图中以两种分类方法列出了Visual Studio和Office版本的对应关系。

如果你的计算机安装了VS 2010，则可以为Office 2007和Office 2010进行VSTO开发。

如果你要为Office 2010进行VSTO开发，则可以选择VS 2010/2012/2013中的任何一个版本。

另外，Visual Studio和Office的版本选用也要参照计算机安装的操作系统。如果是XP系统，可以安装的Office最高版本为2010，可以安装的Visual Studio版本是VS 2010。如果要使用更高版本进行开发，需要使用Windows 7系统。

■ 1.1.4 Visual Studio开发环境

Visual Studio是目前最流行的Windows平台应用程序的集成开发环境。截止到写作本书时，最新版本为 Visual Studio 2015 版本，基于.NET Framework 4.5.2。在Visual Studio中可以使用Visual Basic、Visual C#、Visual C++、Visual F# 4种程序语言。

■ 1.1.5 Visual Studio版本沿革

自微软公司1997年发布Visual Studio 97以来，现在最新的版本为Visual Studio 2015，表1.1列出了Visual Studio比较新的版本及其发布日期。

表1.1 Visual Studio各版本及其发布日期

名称	内部版本	发布日期
Visual Studio 2005	8.0	2005-11-07
Visual Studio 2008	9.0	2007-11-19
Visual Studio 2010	10.0	2010-04-12
Visual Studio 2012	11.0	2012-08-25
Visual Studio 2013	12.0	2013-10-17
Visual Studio 2015	14.0	2014-11-10

■ 1.1.6 Visual Studio的安装

确认计算机中已经安装Office后，再安装Visual Studio。Visual Studio 2012的安装文件名为“CN_Visual_Studio_Ultimate_2012_x86.iso”，大约1.5 GB。安装时可以用Daemon Tools虚拟光驱装载这个iso压缩文件，会自动运行安装程序。

Visual Studio的安装过程比较简单，根据安装向导的提示，进行少量的设定即可完成安装。要注意的是安装向导中间的一个对话框，让用户选择“要安装的可选功能”，进行VSTO开发，此处一定要勾选“Microsoft Office开发人员工具”复选框，如图1.2所示。

此外，还要注意开发语言的选择。在安装Visual

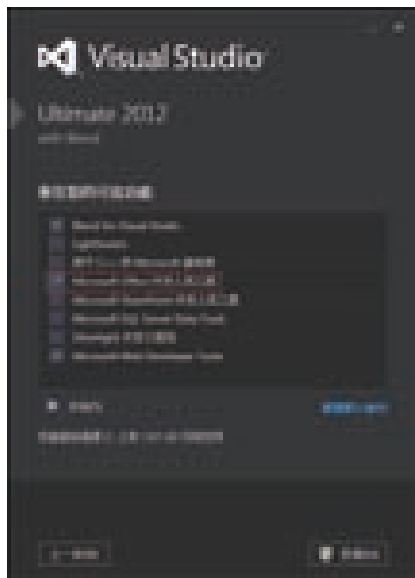


图1.2 Visual Studio 2012安装过程

Studio 2008过程中，一定要勾选“Language Tools/Visual C#/Visual Studio Tools for Office”复选框，如图1.3所示。

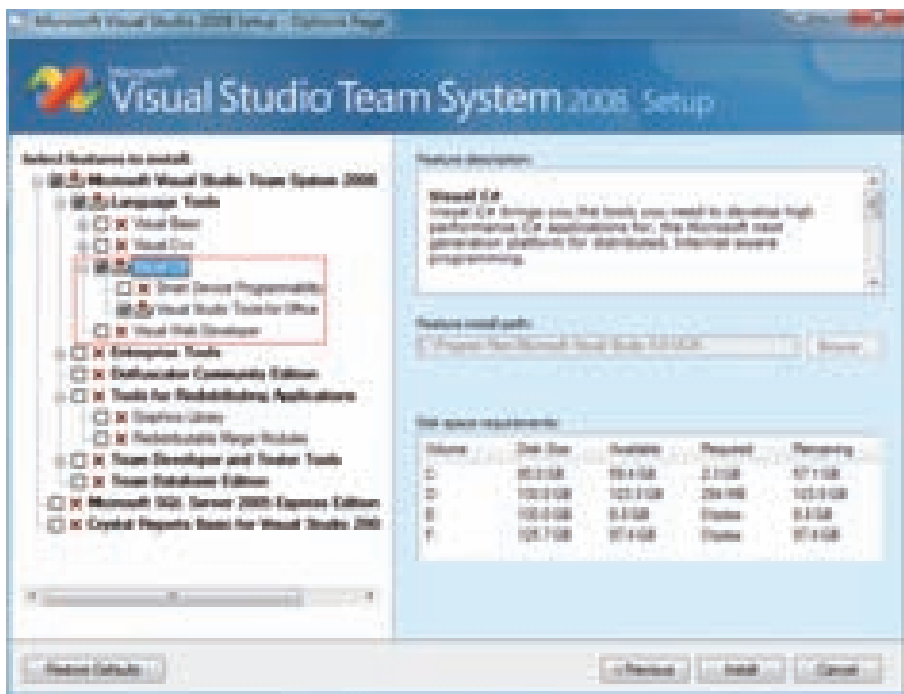


图1.3 Visual Studio 2008安装过程

1.2 创建第一个C#应用程序

计算机中顺利安装Visual Studio以后，就可以开始编写你的第一个C#程序。因为VSTO是C#的一种项目类型，因此在学习VSTO之前，首先应该学会最基本的C#程序的编写和调试技巧。

最基础的C#程序要数控制台应用程序和Windows窗体应用程序这两种项目类型。分别介绍如下。

1.2.1 控制台应用程序

控制台应用程序是最基本的C#项目，程序在运行期间，输入和输出都在一个黑屏窗口中进行操作。

启动Visual Studio 2012，单击菜单【文件/新建/项目】，在新建项目对话框中，依次选择【模板/Visual C#/Windows/控制台应用程序】，项目名称重命名为“ConsoleApplication20160629”，单击“确定”按钮，如图1.4所示。

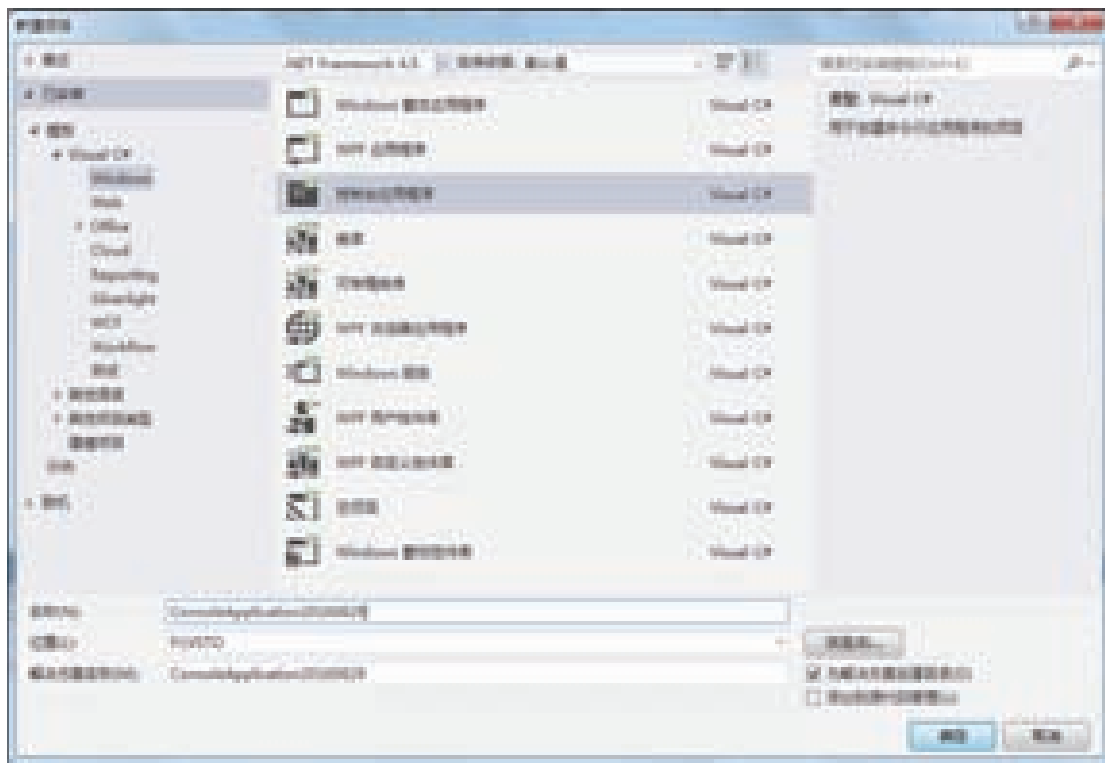


图1.4 创建控制台应用程序

双击解决方案资源管理器中的Program.cs，打开这个类模块，编辑代码如下：

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace ConsoleApplication20160629
8 {
9     class Program
10     {
11         static void Main(string[] args)
12         {
13             Console.Write("Hello ,C#");
14             Console.ReadKey();
15         }
16     }
17 }

```

控制台应用程序的程序入口是Program类的Main函数。因此，编辑代码后，按下快捷键【F5】，输出结果，如图1.5所示。

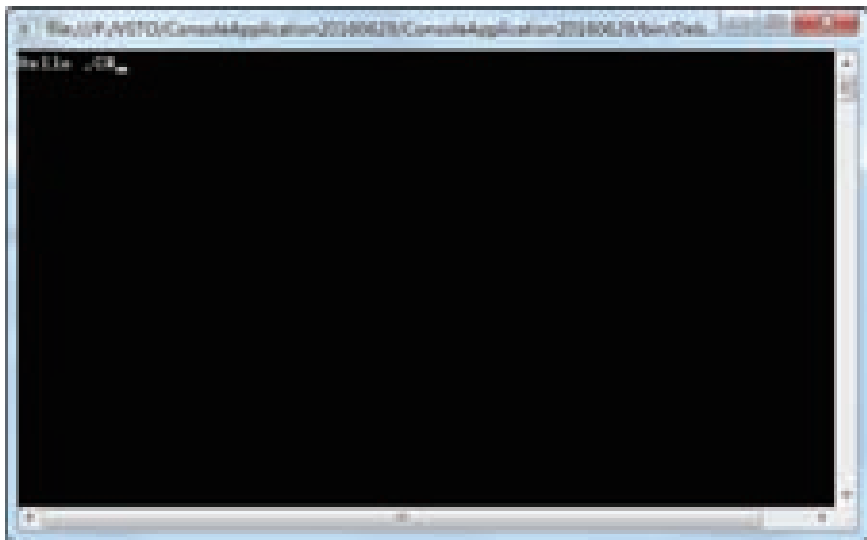


图1.5 控制台应用程序输出窗口

此时，在键盘上按下任何键，黑屏窗口消失，调试结束。

注意 在C#中，单步调试代码的快捷键是【F11】。

1. Console类

System.Console类表示控制台应用程序的标准输入流、输出流和错误流。Console类最常用的方法如表1.2所示。

表1.2 Console类常用方法

方法	描述
Write	将指定的字符串值写入标准输出流
WriteLine	将指定的字符串值（后跟当前行终止符）写入标准输出流
Read	从标准输入流读取下一个字符
ReadLine	从标准输入流读取下一行字符
ReadKey	获取用户按下的下一个字符或功能键。按下的键显示在控制台窗口中

2. 接收用户输入

使用Console.ReadLine方法，可以读取用户输入的内容，在下面的范例中当用户输入一个英文句子，按下回车后，在控制台中输出相应的大写字符串。

修改Main函数中的代码为：

```
1      static void Main(string[] args)
2      {
3          string nm = Console.ReadLine();
4          Console.WriteLine(nm.ToUpper());
```



```
5         Console.ReadKey();  
6     }
```

上述代码中，nm是一个字符串变量，接收用户的输入。ToUpper是字符串的一个转换函数，将字符串转换为对应的大写内容。

程序的执行效果，如图1.6所示。

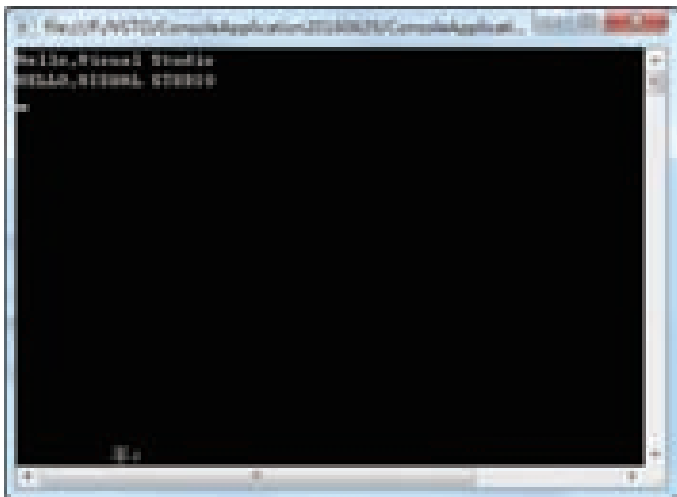


图1.6 接收用户输入

3. 输出结果到控制台

用于结果输出的方法有Write和WriteLine，这两个方法的不同之处在于，后者输出结果后自动换行，而Write则会在上一个结果之后继续输出。

```
1     static void Main(string[] args)  
2     {  
3         Console.Write("白日依山尽，");  
4         Console.Write("黄河入海流。");  
5         Console.Write("欲穷千里目，");  
6         Console.Write("更上一层楼。");  
7         Console.WriteLine("\n");  
8         Console.WriteLine("锄禾日当午，");  
9         Console.WriteLine("汗滴禾下土。");  
10        Console.WriteLine("谁知盘中餐，");  
11        Console.WriteLine("粒粒皆辛苦。");  
12        Console.ReadKey();  
13    }
```

上述代码在控制台输出了两首古诗，第一首使用Write输出，四句古诗输出在一行中，并没有换行，“Console.WriteLine(“\n”)”;这一句，表示在两首古诗之间输出一个空白行，如图1.7所示。

程序调试完毕后，单击【文件/关闭解决方案】。

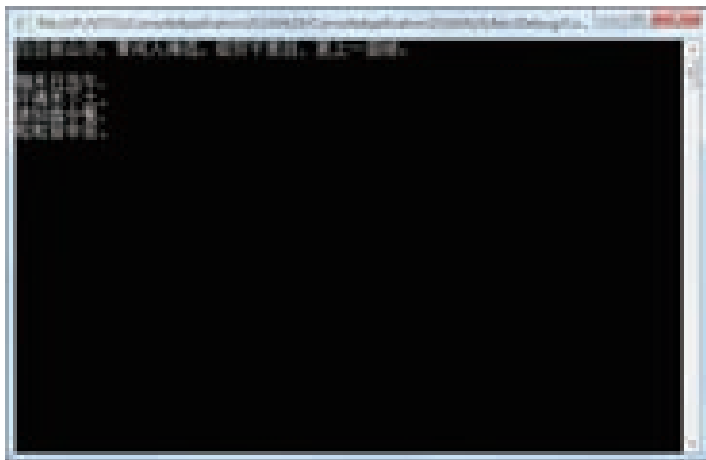


图1.7 输出结果到窗口

■ 1.2.2 Windows窗体应用程序

与控制台应用程序相比，Windows窗体应用程序允许使用窗体和控件，可以做出更美观的界面，完成更复杂的编程任务。

启动Visual Studio，单击菜单【文件/新建/项目】，在新建项目对话框中，依次选择【模板/Visual C#/Windows/Windows窗体应用程序】，项目名称重命名为“WindowsFormsApplication20160629”，单击“确定”按钮，如图1.8所示。



图1.8 创建Windows窗体应用程序

在Visual Studio中，自动打开Form1的设计视图，从控件工具箱中拖动一个button控件到Form1，如图1.9所示。

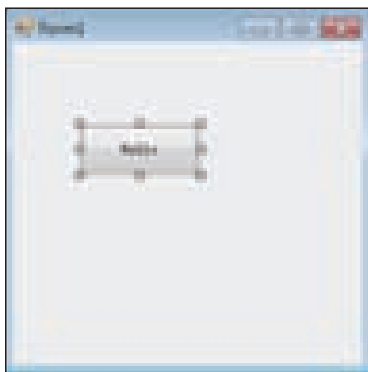


图1.9 窗体设计视图

双击窗体空白区域，进入窗体的Load事件代码区域；然后回到设计视图，双击button1，编写按钮的单击事件，Form1.cs的完整代码如下：

```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Threading.Tasks;
9 using System.Windows.Forms;
10
11 namespace WindowsFormsApplication20160629
12 {
13     public partial class Form1 : Form
14     {
15         public Form1()
16         {
17             InitializeComponent();
18         }
19
20         private void Form1_Load(object sender, EventArgs e)
21         {
22             MessageBox.Show("Hello,Form");
23         }
24
25         private void button1_Click(object sender, EventArgs e)
26         {
27             this.Text = System.DateTime.Now.ToString();
28         }
29     }
30 }
```

启动调试后，在窗体显示之前，跳出一个对话框，显示“Hello,Form”，单击窗体上的按钮，会看到窗体的标题文字变为当前系统时间。

注意 MessageBox用于显示一个对话框，使用该语句的模块顶部必须写上“using System.Windows.Forms;”这条指令。

■ 1.2.3 生成可执行文件

C#程序不仅可以在开发计算机上调试运行，也可以生成可执行文件，然后把扩展名为.exe的可执行文件发送到其他计算机直接运行使用。

在C#项目中，每当重新调试运行后，计算机总是自动生成可执行文件，对于上述窗体应用程序，可以打开如下路径找到最后生成的可执行文件：

…\WindowsFormsApplication20160629\WindowsFormsApplication20160629\bin\Debug该路径下的“WindowsFormsApplication20160629.exe”就是这个项目的可执行文件。

除了窗体应用程序以外，其他的C#项目类型最后生成的结果文件也均在Debug或者Release文件夹中。

1.3 认识Visual Studio开发环境

对于刚使用Visual Studio的用户，首先需要熟悉一下开发环境的界面构成，如图1.10所示。

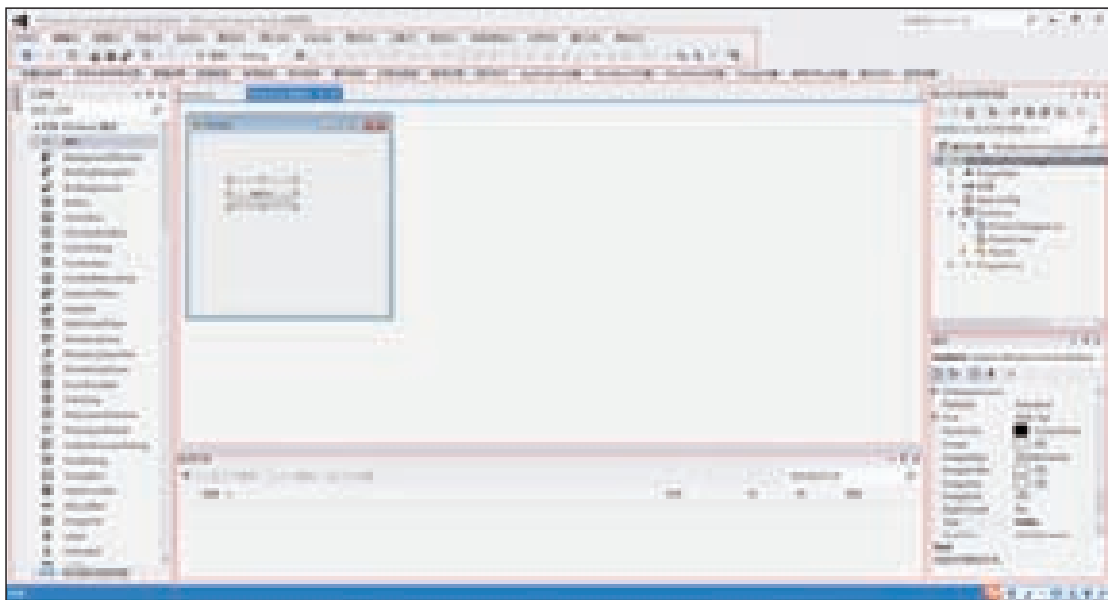


图1.10 Visual Studio开发环境

Visual Studio的界面主要由以下几个部分构成：

菜单栏中包括Visual Studio几乎所有的功能和命令，工具栏中给出了最常用的命令控件。

控件工具箱是用于往窗体或用户控件上增加控件的。

位于Visual Studio正中央区域，通过单击上方的选项卡来切换类模块的代码视图。

在编写代码过程中，或者在调试期间，显示各种错误和异常信息。

以树形结构的方式，列出解决方案、项目、类模块等组成部分，用户通过鼠标双击类模块打开相应的代码视图。

属性窗格用于设置和显示窗体及控件的属性，以及设计窗体和控件的事件过程。

注意 C#窗体和控件的事件过程定义是通过属性窗口来实现的。

【文件】菜单中包含新建、打开和关闭解决方案这些常用命令。

【视图】菜单用于显示和隐藏各个窗格，以及各种工具栏。例如，当我们设计窗体时，看不到快捷工具箱或者属性窗格，可以从视图中调出来，如图1.11所示。

【项目】菜单中的命令是专门设置C#项目各种属性的。例如，向项目中增加和移除窗体、



类模块、增加和删除外部引用等操作，都通过【项目】菜单来操作，如图1.12所示。

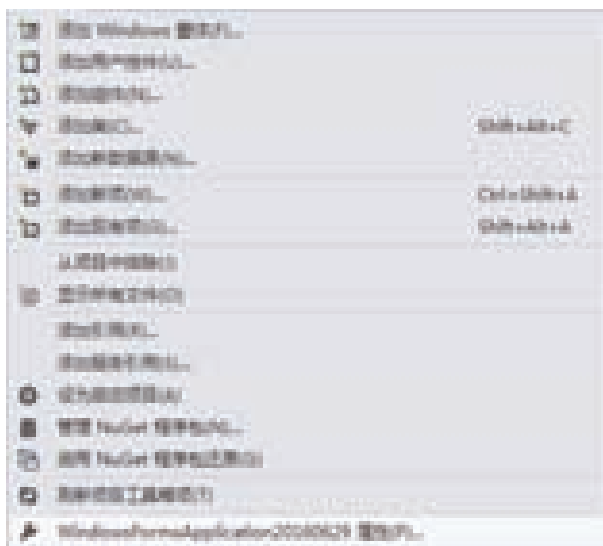


图1.12 【项目】菜单

■ 1.3.4 菜单栏和工具栏的自定义

Visual Studio开发环境的菜单栏以及工具栏的内容，可以通过单击【工具/自定义】，在图1.13所示的对话框中操作。

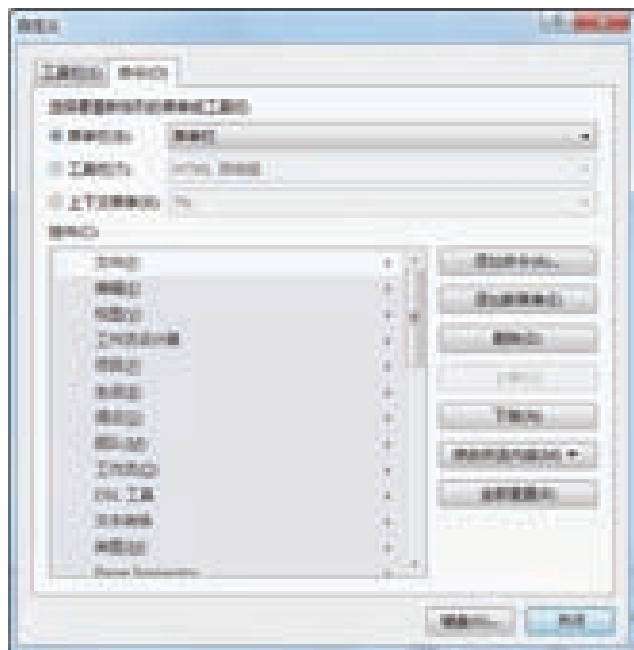


图1.13 工具栏和菜单栏的自定义

这一点有点类似于Office 2003的菜单和工具栏的自定义。

■ 1.3.5 Visual Studio选项

Visual Studio开发环境的界面和外观显示以及编程过程的各种设定，可以通过单击菜单【工具/自定义】，如图1.14所示。

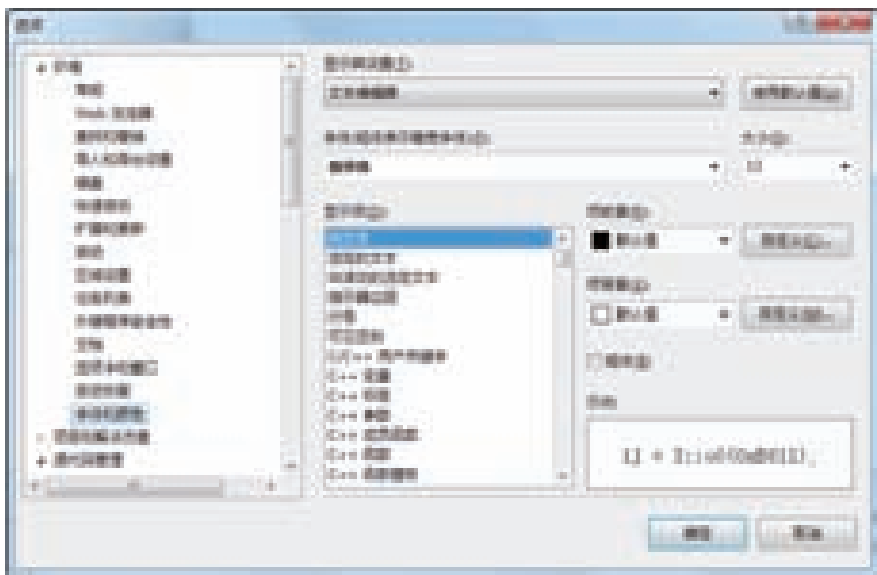


图1.14 Visual Studio选项

例如，我们可以通过这个对话框来修改代码区域的背景色或者代码的字体大小等。

1.4 Visual Studio项目组织结构

Visual Studio工程的组织结构是以解决方案展开的，一个解决方案中可以包含多个项目，每个项目之间允许使用不同的开发语言，每个项目又包含若干类模块，有各自的外部引用管理。

一般情况下，手动创建的Visual Studio工程有一个解决方案，这个解决方案下面只有一个项目。

■ 1.4.1 解决方案

解决方案（Solution）的作用类似于Visual Basic 6中的工程组，是用来统一管理各个项目（Project）的。

为了说明Visual Studio项目组织结构，按照以下步骤进行操作：

第1步：启动Visual Studio 2012，创建一个名为“WindowsFormsApplication20160705”的

Windows窗体应用程序，并且把解决方案名称重命名为“Solution20160705”，如图1.15所示。

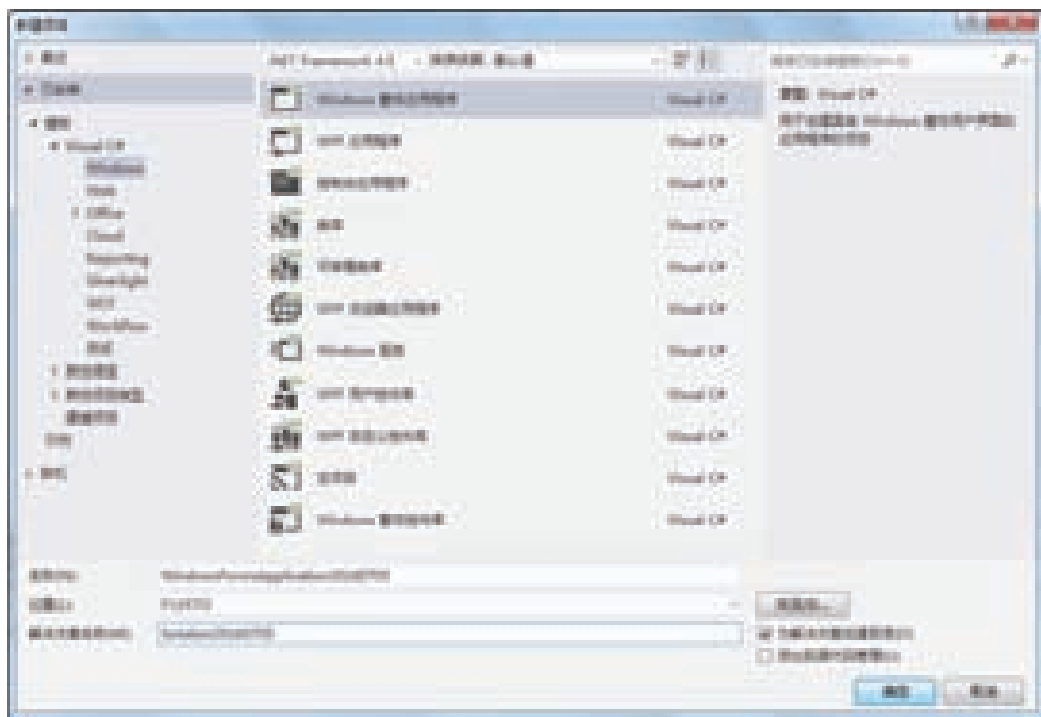


图1.15 新建解决方案

第2步：在Visual Studio中，单击菜单【文件/添加/新建项目】，在“新建项目”对话框中，选择控制台应用程序，项目名称重命名为“ConsoleApplication20160705”，如图1.16所示。

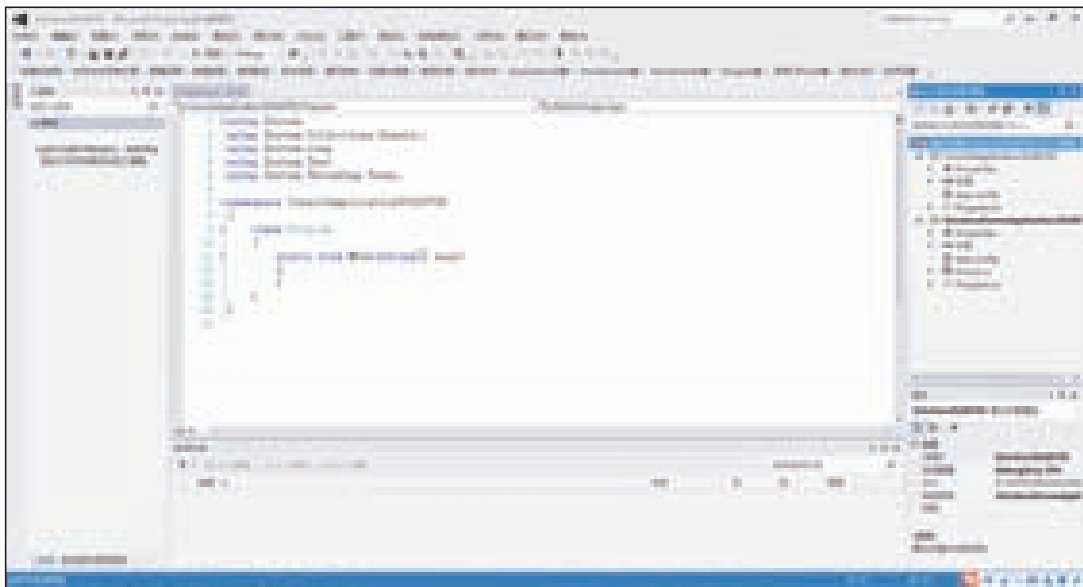


图1.16 添加新项目

可以从Visual Studio右侧的解决方案资源管理器窗格中看到，解决方案现包含两个项目，当然根据需要可以添加更多的项目到这个解决方案中。

此时，按下【F5】键运行调试，会看到屏幕上出现一个Windows窗体，这是因为解决方案中已把WindowsFormsApplication20160715这个项目设置为启动项目。

第3步：更改解决方案的启动项目。在Visual Studio中，右击解决方案资源管理器中ConsoleApplication20160705项目节点，并且选择“设为启动项目”，如图1.17所示。

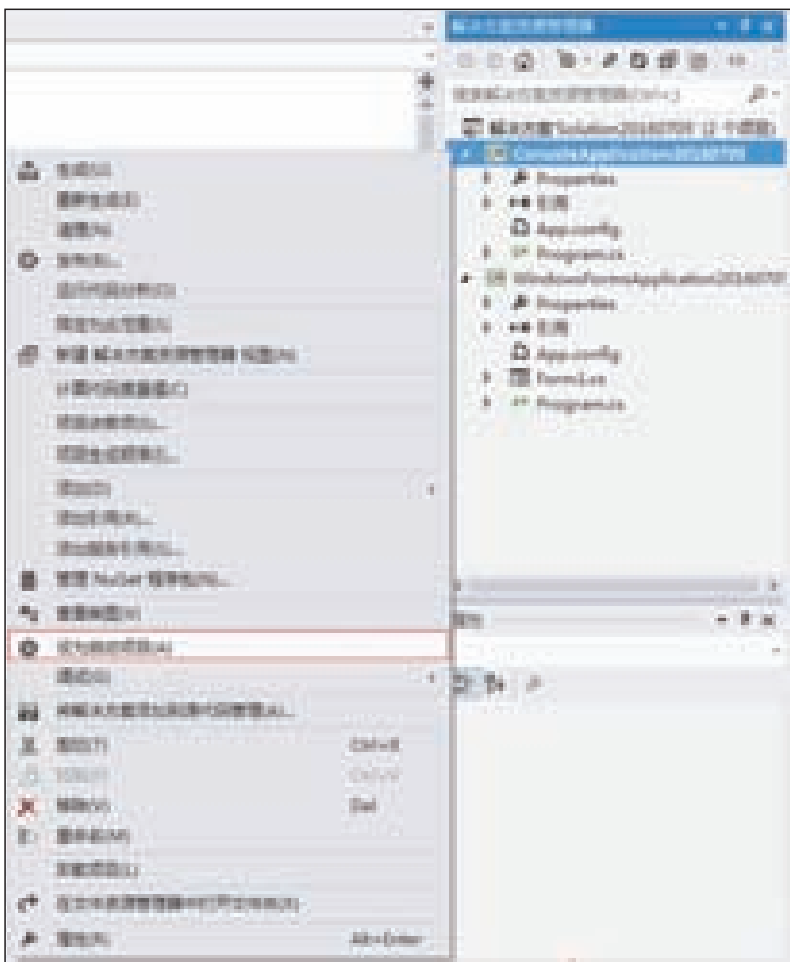


图1.17 更改启动项目

这时再次调试运行，屏幕出现一个控制台窗口。这说明成功地更改了默认的启动项目。

第4步：打开计算机的资源管理器，找到上述解决方案路径，打开文件夹查看，如图1.18所示。

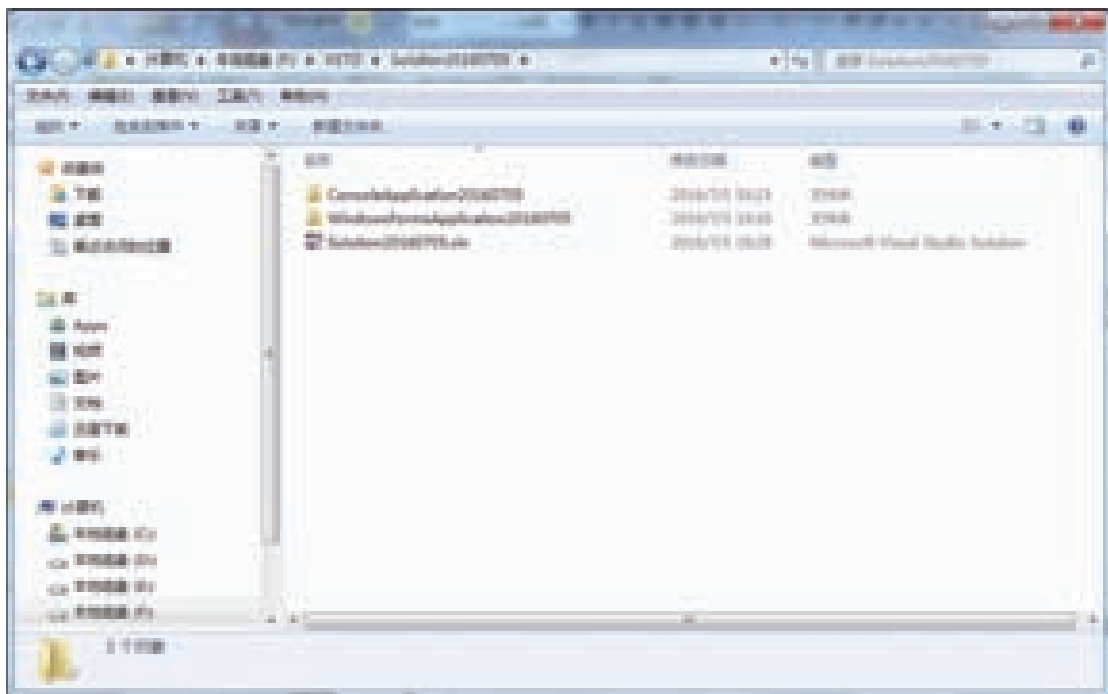


图1.18 Visual Studio文件夹

从文件夹中可以看到一个扩展名为.sln的解决方案文件，还可以看到该解决方案包含的两个项目文件夹。

注意 如果在Visual Studio中关闭了刚才的解决方案，还想再次打开编辑，请单击菜单【文件/打开项目或解决方案】，在“浏览”对话框中找到.sln文件即可。

■ 1.4.2 项目

前面提到，解决方案仅仅是若干个项目的组织框架而已，真正的代码和程序部件都包含在项目中。项目（Project）也可以称作工程，一个项目统一管理各个类模块以及项目中的所有引用。

对于项目所有的操作，可以通过单击Visual Studio的【项目】主菜单找到，也可以使用鼠标右击解决方案资源管理器中的项目节点。

如果要更改一个项目的属性，可以单击【项目/项目属性】，然后在打开的“项目属性页”中查看或修改若干属性，如图1.19所示。

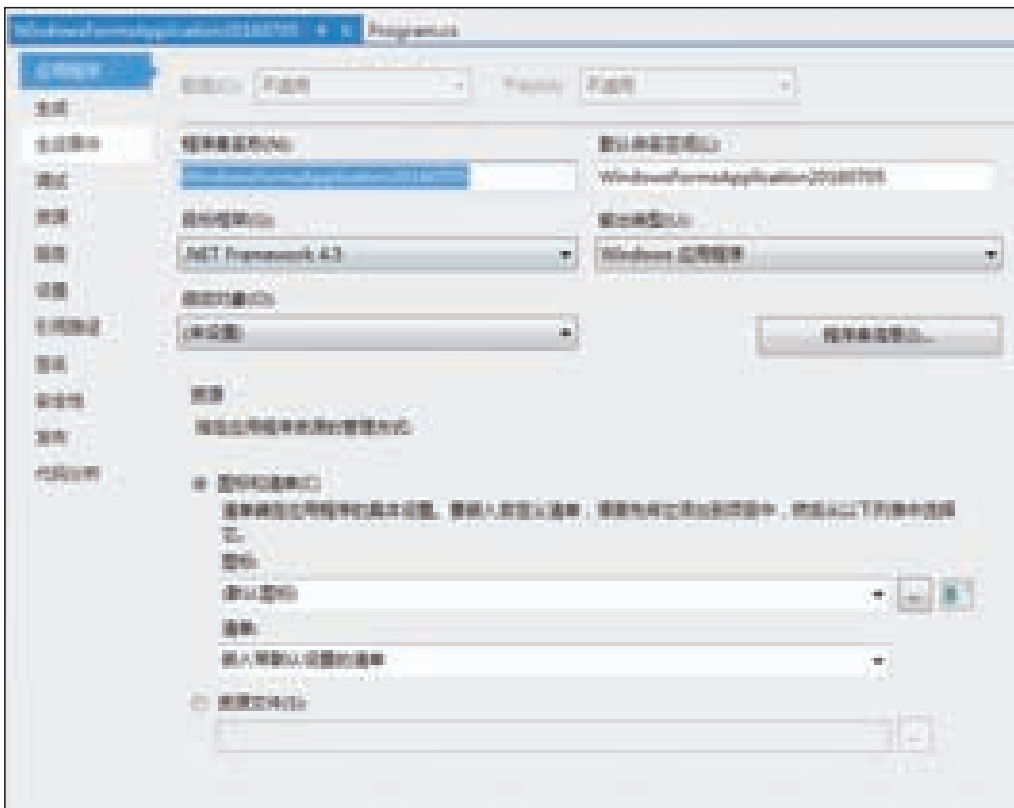


图1.19 项目属性设定

■ 1.4.3 类模块

一个项目中可以添加窗体、用户控件以及类。其实这些都可以理解为类模块，类模块的扩展名是.cs，保存于项目文件夹中。

关于类模块的具体使用方法可以参阅2.12节。

■ 1.4.4 引用管理

每一个项目都有各自的引用管理，通过单击解决方案资源管理器中项目节点的“引用”节点，可以展开该项目的引用。

项目中的引用，既可以把已有引用移除出去，也可以单击菜单【项目/添加引用】来增加新的引用。

1.5 使用帮助系统

在使用Visual Studio编程过程中遇到疑难问题时，既可以通过互联网搜索相关问题，也可以在编程环境中按下快捷键【F1】，搜索微软提供的MSDN帮助。

■ 1.5.1 设置帮助查看方式

在Visual Studio中，有以下两种查看帮助的方式：

- 在浏览器中启动。
- 在帮助查看器中启动。

可以通过单击菜单【帮助/设置帮助首选项】来选择查看方式。默认的查看方式为浏览器查看方式，也就是说，当用户在编程过程中按下【F1】键时，会启动默认的网页浏览器并跳转到相应的微软MSDN页面。

为了编程的方便，笔者建议选择帮助查看器中启动。

■ 1.5.2 下载和安装Help Viewer

如果用户选择“在帮助查看器中启动”，当在Visual Studio中按下【F1】键，或者单击菜单【帮助/查看帮助】时，则会启动Help Viewer。

■ 1.5.3 管理帮助内容

如果是初次使用Help Viewer，这个查看器中没有任何的帮助内容。为此，需要事先下载必要的帮助内容到计算机。

在Help Viewer中切换到“管理内容”选项卡，选择添加如下帮助内容：

- Visual Studio 2012：基础。
- Visual Studio 2012：Visual Studio中的Office开发。
- Visual Studio 2012：Visual Basic和Visual C#。

然后单击右下角的“更新”按钮，稍后就会自动下载这些内容到计算机，如图1.20所示。

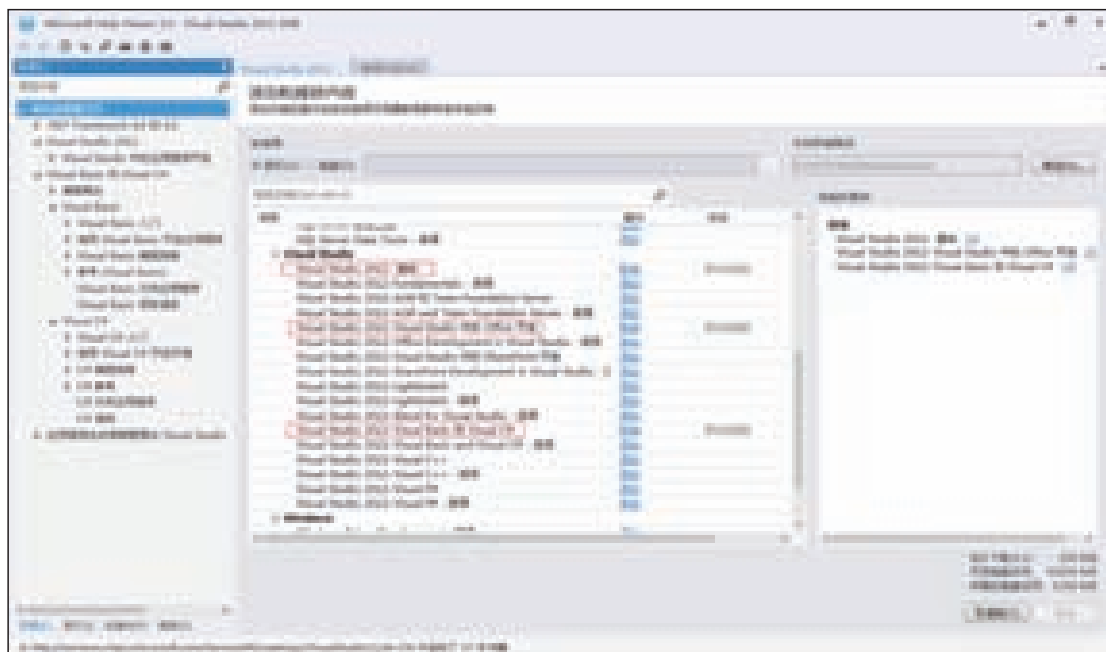


图1.20 管理帮助内容

例如，要了解MessageBox的用法，可以单击左侧导航窗格最下面的“搜索”选项卡，然后在窗格最上面的搜索框中输入关键字后按下【Enter】键，右侧自动给出相关帮助内容，如图1.21所示。



图1.21 搜索关键字

本章要点回顾

- VSTO是一种全新的Office开发技术，开发人员可以使用C#或VB.NET语言访问Office对象。
- Visual Studio是微软开发的功能强大的程序开发环境。本章讲述了在安装Visual Studio的中间步骤中，一定要勾选VSTO开发。
- 从创建最简单的C#项目到控制台应用程序，加深对Visual Studio开发环境以及C#解决方案组织结构的理解。

第2章

C#语法基础

本书介绍的是用C#语言进行VSTO开发，因此在正式进行VSTO开发之前必须先掌握一定的C#语法基础。

C#是一个简单的、现代的、通用的、面向对象的编程语言，它是由微软公司开发的。C#是基于C和C++语言的，因此如果对C和C++编程有基本的了解，将有助于您学习C#编程语言。

 本章视频：C#语法基础.wmv

2.1 变量的声明和赋值

C#代码中用到的所有变量以及对象必须先声明再使用。声明格式为：

```
变量类型 变量名 [=初值];
```

例如，“`int i=34;`”就声明了一个整型变量*i*，并且赋初值34。

注意 C#中严格区分大小写，因此变量*i*和变量*I*是不同的变量。

2.1.1 常用的数据类型

C#常用数据类型如表2.1所示。

表2.1 C#常用数据类型

类型	描述	允许的值
int	整型	负20多亿到正20多亿
long	长整型	比int范围更大
float	浮点型（常用来存储小数）	
double	双精度浮点型	比float范围更大

续表

类型	描述	允许的值
char	字符型	一个Unicode字符, 存储0~65 535之间的整数
string	字符串	
bool	布尔型	true或false

■ 2.1.2 赋值运算符

赋值语句的语法格式为:

```
变量名称=表达式;
```

表达式可以是一个常量, 也可以是一个变量或者是任意组合起来的可以计算的表达式。

除了使用=作为赋值运算符外, C#还允许的几种赋值运算符, 如表2.2所示。

表2.2 C#赋值运算符

赋值运算符	示例	含义
+=	a+=b	等价于a=a+b
-=	a-=b	等价于a=a-b
=	a=b	等价于a=a*b
/=	a/=b	等价于a=a/b
%=	a%=b	等价于a=a%b
++	i++	i自加1, 等价于i=i+1或i+=1
--	i--	i自减1, 等价于i=i-1或i-=1

■ 2.1.3 变量的作用范围

变量的作用范围和其声明的位置有关系, 可以在类模块的顶部、过程内部、语句块内部或者循环体内部声明变量。换言之, 在声明范围之外, 是不能访问和使用这个变量的。

以下代码是一个典型的使用变量的方法, 代码中先后声明了一个整型变量和一个字符串变量, 最后把两个变量连接在了一起。

注意 如果表达式中+的两端有字符串数据, 则+的含义是连接字符串, 而不是数学中的相加。

```
1      public void 变量的声明和赋值()
2      {
3          int i;
4          i = 34;
5          string s = @"e:\ab\cd.txt";
6          result = i + s;
7      }
```

相应的VBA语法为:


```

1 Public Sub 变量的声明和赋值()
2     Dim i As Integer
3     i = 34
4     Dim s As String
5     s = "e:\ab\cd.txt"
6     result = i & s
7 End Sub

```

2.2 字符与字符串处理

字符和字符串类型是编程中最常用的数据类型。编程过程中会涉及字符串的提取、拆分等操作。

2.2.1 字符变量

字符变量的声明格式为：

```
char 字符变量;
```

例如，`char MyChar = 'd'`就声明了一个字符变量并赋初值。表达一个字符常量，必须加上单引号，而不是双引号。

字符变量经常用来代表ASCII码表中的字符。上述的'd'转换为整型数就是100，ASCII码表可以用下面的公式产生。

创建一个名为“WindowsFormsApplication20160606”的窗体应用程序，在窗体上放置一个listbox列表框控件，然后在窗体的Load事件中写入如下代码：

```

1     private void Form1_Load(object sender, EventArgs e)
2     {
3         this.listBox1.Items.Clear();
4         for (int i = 0; i <= 127; i++)
5         {
6             char a = Convert.ToChar(i);
7             this.listBox1.Items.Add(i.ToString() + "\t" + a);
8         }
9     }

```

上述代码把0~127这128个字符的代码值和符号添加到列表框中，运行效果如图2.1所示。

此外，字符变量可以和整型变量直接进行运算。比如下面两行代码，变量j等于650，原因是abc是字符变量，存储了常量A，而A的ASCII码值是65，因此j是65和10的乘积。

```

1     char abc = 'A';
2     int j = abc * 10;

```

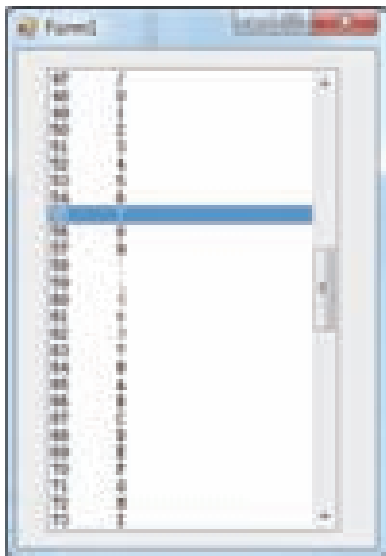


图2.1 字符变量

■ 2.2.2 字符串变量

一个字符串变量可以存储一个字符串，而字符串也可以看作是字符组成的数组。例如，下面的代码中m是一个字符串变量，而m[2]则自动转换为一个字符变量赋给c，那么c将是字符'T'。

```
1      string m = "VSTO";  
2      char c =m[2];
```

使用这个原理，就可以遍历字符串了：

```
1      public void 遍历字符串中的字符()  
2      {  
3          char c;  
4          string s = "microsoft office";  
5          for (int i = 0; i < s.Length; i++)  
6          {      //遍历字符串中的每一个字符  
7              c = s[i];  
8              result += "\n" + c;  
9          }  
10     }
```

与此对应的VBA代码如下：

```
1 Public Sub 遍历字符串中的字符()  
2     Dim c As String, s As String  
3     s = "microsoft office"  
4     result = ""
```

```
5 For i = 1 To Len(s)
6     '遍历字符串中的每一个字符
7     c = Mid(s, i, 1)
8     result = result & vbNewLine & c
9 Next i
10 End Sub
```

2.2.3 转义字符

C#中有一些特殊的转义字符。在代码中，转义字符都以反斜杠开头，如表2.3所示。

表2.3 C#转义字符

转义字符	产生的字符
\'	单引号
\"	双引号
\\	反斜杠
\n	换行
\r	回车
\t	水平制表位
\v	垂直制表位

假如要在对话框中输出两行文字，一行是Microsoft，另一行是Office，可以使用如下代码：

```
MessageBox.Show("Microsoft\rOffice");
```

运行结果如图2.2所示。

在代码中我们可以看到\r是一个转义字符，产生了回车符。再比如我们要输出计算机中的一个路径“F:\VSTO\VSTOBOOK”，直接这样输入代码中会产生错误，因此需要把源字符串中的每一个反斜杠再加一个反斜杠。



图2.2 认识转义字符

```
MessageBox.Show("F:\\VSTO\\VSTOBOOK");
```

与转义字符相反，使用@可以非转义，即原样输出。例如上述输出路径的问题，修改代码为：

```
MessageBox.Show(@"F:\VSTO\VSTOBOOK");
```

也可以输出正确的路径。

当然我们可以想到，下面的这行代码将原样输出\r，而不是换行。

```
MessageBox.Show(@"Microsoft\rOffice");
```

在编程的过程中，会经常使用\t分隔水平的数据，使用\r实现换行输出。

窗体上放置一个richtextbox文本框控件，在窗体的Load事件中输入如下代码：

```
1      string result="";
2      result += "山东省\t济南\r";
3      result += "安徽省\t合肥\r";
4      result += "黑龙江省\t哈尔滨\r";
5      result += "内蒙古自治区\t呼和浩特\r";
6      this.richTextBox1.Text = result;
```

运行结果如图2.3所示。

通过这个示例，我们可以清晰地看到转义字符所起的作用。

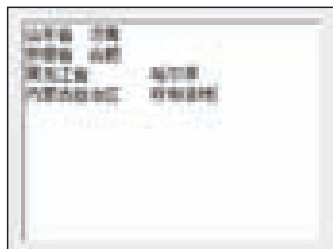


图2.3 使用制表符

■ 2.2.4 字符串连接

使用+运算符实现多个字符串的连接，如果连接的各个数
据都不是字符串，则需要事先使用ToString()方法强制转换为字符串后再连接。

```
1      string s1 = "Microsoft";
2      string s2 = s1 + "Office";
```

下面的两行代码中，s3是两个浮点数相加的结果转换为字符串，结果显示5.858，而s4是空字符串与两个数字相连，因此输出为2.7183.14。读者仔细思考其原理。

```
1      string s3 = (2.718 + 3.14).ToString();
2      string s4 = "" + 2.718 + 3.14;
```

■ 2.2.5 子字符串

在VB或VBA中使用Left、Right以及Mid函数可以提取字符串中的一部分，在C#中可以使用SubString函数达到同样的目的。SubString的用法和VB中的Mid用法非常类似。

```
1  string s1 = "Microsoft";
2  string s5= s1.SubString(3, 4);
```

上述代码中，提取字符串s1，从第三个字符起连续提取4个，因此字符串s5将会等于“roso”。

C#使用IndexOf函数可以判断字符串中是否包含另一个子字符串。

```
1  string s1 = "Microsoft";
2  int p = s1.IndexOf("so");
```

上述代码，目的是监测字符串s1中是否包含“so”，返回的结果是5，因为s1中包含这个子字符串，而且位于第5个位置。反之，如果未找到子字符串，则返回-1。

■ 2.2.6 格式化字符串

String.Format函数的语法为:

```
String.Format(要输出的字符串, 变量列表)
```

要输出的字符串中, 允许使用{数字}这样的占位符, 输出时占位符位置用变量列表中相应位置的变量来替换, 具体看下面的例子。

```
1      public void 格式化字符串()  
2      {  
3          string s0="format",s1="micro",s2="soft",s3 ="visual" ,s4="studio";  
4          result =String.Format("result is {0} {3} {4}",s0,s1,s2,s3,s4);  
5      }
```

上面这个例子依次声明并赋值了5个字符串变量。注意第4行代码, 依次把5个变量置于Format函数的参数中, 要返回的是双引号括起来的部分, 遇到{n}就表示该位置用第n个字符串变量代替, 因此result返回的结果是: result is format visual studio。

■ 2.2.7 字符串的替换

C#中使用Replace函数进行字符串的替换。

Replace函数有以下两种用法:

①字符替换。语法格式如下:

```
Replace(旧字符,新字符)
```

②字符串替换。语法格式如下:

```
Replace(旧字符串,新字符串)
```

```
1      string s = "Microsoft Office";  
2      string t = s.Replace('o', '%');  
3      string u = s.Replace("oso", "微软公司");
```

上述代码中, 变量t返回“Micr%s%ft Office”, 变量u返回“Micr微软公司ft Office”。

■ 2.2.8 字符串与数组

在编程实际应用中, 经常会将一个字符串按照某个特定的符号进行分割。C#可以使用Split函数实现字符串分割。

```
1      public void 字符串与数组()  
2      { //字符串转数组;  
3          string source = "micro soft visual studio";  
4          string[] arr = source.Split(' ');
```

```
5         foreach (string v in arr)
6         { result += v + "\n"; }
7         //数组转字符串
8         result += string.Join("+",arr);
9     }
```

上述代码中字符串数组arr用来存储使用空格分割开的source元素。反过来，可以使用Join函数把字符串数组的每一个元素用特定的符号连接起来，成为一个新的字符串。

具有同样功能的VBA代码如下：

```
1 Public Sub 字符串与数组()
2     '字符串转数组
3     Dim source As String, arr, v
4     source = "micro soft visual studio"
5     arr = Split(source, " ")
6     For Each v In arr
7         result = result & v & vbNewLine
8     Next v
9
10    Dim s(1 To 3) As String
11    s(1) = "excel"
12    s(2) = "word"
13    s(3) = "ppt"
14    result = result & Join(s, "+++")
15 End Sub
```

2.3 逻辑运算

2.3.1 布尔型变量

布尔型变量只能等于true或false这两个逻辑值。布尔型变量或常量经常用于if语句中。

```
1     public void 布尔型变量()
2     {
3         bool flag;
4         flag = (1 > -1);
5         result = flag.ToString();
6     }
```

具有同样功能的VBA代码如下：

```
1 Public Sub 布尔型变量()
2     Dim flag As Boolean
3     flag = (1 > -1)
4     result = flag
5 End Sub
```

2.3.2 比较运算符

C#中的比较运算符如表2.4所示。

表2.4 比较运算符

运算符	含义	示例
==	等于	5==3+2, 返回true
!=	不等于	5!=3+2, 返回false
<	小于	5<3+2, 返回false
>	大于	5>3+2, 返回false
<=	小于等于	5<=3+2, 返回true
>=	大于等于	5>=3+2, 返回true

注意 C#表示逻辑等于的符号是连续两个等于号。

2.3.3 多条件的与或非运算

很多情况下，需要多个逻辑表达式通过布尔运算符连接组合使用，才能表述一个实际的逻辑判断，例如判断一个年份是否是闰年。与或非运算符如表2.5所示。

表2.5 与或非运算符

运算符	含义	说明
!	非	!加在一个逻辑表达式之前，如果原表达式为true，则返回false；如果原表达式为false，则返回true
&&	与	所有表达式均为true，才返回true
	或	至少有一个表达式为true，就返回true；如果所有表达式均为false，则返回false

如果要判断一个字符是否是大写英文字母，可以使用如下代码中的逻辑表达式表述：

```
1 char var1 = 'W';
2 if (var1 >= 'A' && var1 <= 'Z')
3 { MessageBox.Show(var1 + "是一个大写字母"); }
4 else
5 { MessageBox.Show(var1 + "不是大写字母"); }
```

如果要判断一个字符是否是英文字母，需要使用或运算符：

```
1 char var1 = 'b';
2 if (var1 >= 'A' && var1 <= 'Z' || var1 >= 'a' && var1 <= 'z')
3 { MessageBox.Show(var1 + "是一个英文字母"); }
4 else
5 { MessageBox.Show(var1 + "不是英文字母"); }
```

观察下面的实例代码，思考一下result返回值是什么。

```
1 public void 多条件的与或非运算()
2 {
```

```
3         bool flag;  
4         flag = !(true && false || true);  
5         result = flag.ToString();  
6     }
```

具有同样功能的VBA代码如下:

```
1 Public Sub 多条件的与或非运算()  
2     Dim flag As Boolean  
3     flag = Not (True And False Or True)  
4     result = flag  
5 End Sub
```

2.4 不同类型的强制转换

编程过程中, 有时候需要把一个类型的数值赋值给另一个类型的变量, 如果源类型和目标类型是同种类型的, 则直接赋值即可, 这称作隐式转换。如果源类型和目标类型是不同类型的, 则需要显式转换, 否则会提示出错。

下面的4行代码可以正常运行, 因为int类型的数据可以直接赋值给long类型; float类型的数据也可以直接赋值给double类型。

```
1         int I = 32;  
2         long L = I;  
3         float FLT = 3.14F;  
4         double D = FLT;
```

2.4.1 ToString

要把一个数据显示在MessageBox对话框中, 或者显示在窗体的文本框中, 都需要事先把变量或计算结果转换为string类型才行。以下代码, 把整型变量I显示在窗体上的富文本框中。

```
1         int I = 32;  
2         this.richTextBox1.Text = I.ToString();
```

2.4.2 Parse

与ToString功能相反, Parse可以把字符串转换为其他数据类型。

```
1         string temp = "1";  
2         int i = int.Parse(temp);  
3         float f = float.Parse(temp);  
4         char c = char.Parse(temp);
```



```
5
6         string s = "false";
7         bool b = bool.Parse(s);
8
9         string d = "2016-8-8";
10        DateTime dt = DateTime.Parse(d);
```

上述代码分别把字符串类型的数据转换为整型、浮点型、字符型、布尔型以及日期时间型。

■ 2.4.3 Convert

C#还可以使用System.Convert来转换数据类型。例如下面这句代码，把字符串转换为double类型。

```
double p = System.Convert.ToDouble(“3.14”);
```

System.Convert常用数据类型转换函数如表2.6所示。

表2.6 System.Convert常用数据类型转换函数

转换函数	说明
ToString(原类型)	把原类型数据转换为字符串
ToBoolean(原类型)	把原类型数据转换为布尔型
ToDouble(原类型)	把原类型数据转换为双精度
ToInt32(原类型)	把原类型数据转换为32位整型
ToDateTime(原类型)	把原类型数据转换为日期时间类型

除了上述的转换方法外，还可以在原数据前面放一个括号，括号内是目标数据类型的名称。如下代码：

```
1         int m = 3;
2         double d =(double)m;
```

上述代码把整型数据m赋值给双精度变量d。

以下代码演示了最常用的数据类型的转换，读者自行调试体会。

```
1         public void 不同类型的强制转换()
2         {
3             int i = 30;
4             string s = i.ToString();//整型转字符串
5             s = "30.65";
6             float f = float.Parse(s); //字符串转浮点型
7             double d = (double)f; //浮点型转双精度型
8             int j = (int)d; //双精度型转整型
9             result = i + s + f + d + j + "";
10        }
```

具有同样功能的VBA代码如下：

```
1 Public Sub 不同类型的强制转换()  
2     Dim i As Integer, s As String, f As Single, d As Double, j As Integer  
3     i = 30  
4     s = CStr(i) '整型转字符串  
5     s = "30.65"  
6     f = CSng(s) '字符串转浮点型  
7     d = CDb1(f) '浮点型转双精度型  
8     j = CInt(d) '双精度型转整型  
9     result = i & s & f & d & j & "  
10 End Sub
```

2.5 使用数组

声明一个变量，只能容纳一个数据，如果有相同类型的多个数据，可以使用数组来提高编程效率。

2.5.1 数组的声明和初始化

数组的声明与变量的声明非常相似，声明一个整型变量的语法格式为：

```
int i;
```

那么声明一个整型数组的语法格式为：

```
int[] arr;
```

可以看出声明数组时，只需要在数据类型后加一对中括号即可。

声明数组的时候，也可以像变量一样，直接赋值。

```
1         int[] n1 = new int[10];  
2         int[] n2 =new int[10]{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  
3         int[] n3 = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

上述代码，声明了三个整型数组，其中数组n1只声明而没有赋给任何值；n2和n3在声明的同时，完成数组的初始化。

如果要为数组n1赋值，可以通过下标，逐个赋值，例如n1[2]=3。

2.5.2 一维数组

下面以计算1~10之间的奇数和为例，说明一维数组的用法。

```
1         int[] a= { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  
2         int total = a[0] + a[2] + a[4] + a[6] + a[8];
```

上述代码中，第1行声明了一个具有10个元素的整型数组，第2行代码累加数组a的5个

元素。C#数组的下标是从0开始的，因此上述代码相当于累加1+3+5+7+9，最后变量total返回25。

注意 C#数组的下标是从0开始的。

■ 2.5.3 数组元素的遍历

还是上面的数字加和问题为例，说明如何使用循环处理数组。

```
1      public void Sum()  
2      {  
3          int[] n = new int[10];  
4          for (int i = 0; i < n.Length; i++)  
5          {  
6              n[i] = i + 1;  
7          }  
8          int total = 0;  
9          for (int i = 0; i < n.Length; i = i + 2)  
10         {  
11             total += n[i];  
12         }  
13         MessageBox.Show(total.ToString());  
14     }
```

上述代码中，第4行到第7行，使用for循环为数组n赋初值。第8行把总和变量total初始化为0，第9行到第12行这个for循环体，步长为2，访问数组n中的奇数，累加给total，最后返回25。

代码中n.Length表示数组的长度，也就是指数组中元素个数。

为了加深对数组的理解和认识，再举一个用数组处理斐波那契数列的问题。

斐波那契数列是数学家列昂纳多·斐波那契（Leonardoda Fibonacci）以兔子繁殖为例子引入的有趣数列，故又称为“兔子数列”，指的是这样一个数列：0、1、1、2、3、5、8、13、21、34、…，可以看出该数列前两项固定为0和1，从第三项起，每一项均等于该项前面两项之和。

在数学上，斐波纳契数列以递归的方法定义： $F(0)=0$ ， $F(1)=1$ ， $F(n)=F(n-1)+F(n-2)$ ($n \geq 2$, $n \in N^*$)。

因此可以使用C#中的数组打印出斐波那契数列的前20项：

```
1      public void Fibonacci()  
2      {  
3          int[] F = new int[20];  
4          F[0] = 0;  
5          F[1] = 1;  
6          for (int i = 2; i < F.Length; i++)  
7          {
```

```

8         F[i] = F[i - 2] + F[i - 1];
9     }
10    string result = "";
11    for (int i = 0; i < F.Length; i++)
12    {
13        result += F[i].ToString()+ " ";
14    }
15    this.richTextBox1.Text =result;
16 }

```

上述代码中，第4行到第9行为数组F赋值，第11到第14行再次使用for循环遍历斐波那契数组各个元素，并且把每一个元素追加到result之后，最后在富文本框中显示如下内容：

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181

以下代码是典型的用于处理数组的示例，读者自行调试体会。

```

1     public void 一维数组()
2     {
3         string[] arr = { "ab", "cd" };
4         int[] a = new int[3];
5         a[0] = 3; a[1] = 4; a[2] = 5;
6         result = arr[0] + arr[1];
7         foreach (var v in a)//遍历数组元素
8         {
9             result += v.ToString();
10        }
11    }

```

具有同样功能的VBA代码如下：

```

1 Public Sub 一维数组()
2     Dim arr(0 To 1) As String
3     arr(0) = "ab": arr(1) = "cd"
4     Dim a(0 To 2) As Integer
5     Dim v
6     a(0) = 3: a(1) = 4: a(2) = 5
7     result = arr(0) + arr(1)
8     For Each v In a '遍历数组元素
9         result = result & v
10    Next v
11 End Sub

```

■ 2.5.4 二维数组

二维数组用来存储二维方阵数据，比如存储中国各省的省会城市，省的简称等信息。

```

1     public void province()
2     {
3         string[,] p = new string[5, 3] { { "河北省", "冀", "石家庄" }, { "河
南省", "豫", "郑州" }, { "湖北省", "鄂", "武汉" }, { "湖南省", "湘", "长沙" }, { "江苏
省", "苏", "南京" } };

```

```

4         for (int r = 0; r < 5; r++)
5         {
6             MessageBox.Show(p[r, 0] + p[r, 2]);
7         }
8     }

```

上述代码中，第3行声明了一个字符串二维数组p，并且把5个省的信息作为初值赋给p。数组p的第一维长度是5，表示存储了5个省的信息；第二维的长度是3，表示存储每个省的省名称、省简称和省会城市名。

第4行到第6行遍历每一个省，并把省名称和省会城市名在对话框中显示。

多维数组元素的表示，还是使用一个中括号，里面每一维的下标用半角逗号隔开，比如代码中的p[r,0]表示第r个省的省名称。

下面再举一个利用二维数组生成九九乘法表的经典代码：

```

1     public void multiply()
2     {
3         string[,] t = new string[9, 9];
4         string result = "";
5         int R, C;
6         for (int r = 0; r < 9; r++)
7         {
8             result += "\n";
9             R = r + 1;
10            for (int c = r; c < 9; c++)
11            {
12                C = c + 1;
13                result += R + "*" + C + "=" + (R * C) + " ";
14            }
15        }
16        MessageBox.Show(result);
17    }

```

上述代码中，第3行声明一个9行9列的二维字符串数组t，该数组每一个元素存储乘法表中每一句口诀。第5行声明了两个大写字母，用来表示乘法表中的实际数字，因为数组的下标是从0开始，而乘法表是从1开始，故此要加1。

特别注意的是代码中第10行，循环变量c是从r开始的，而不是从0开始，这样产生的乘法表将是一个上三角矩阵，而不是矩形方阵。

运行效果如图2.4所示。

遍历二维数组，就需要双层for循环才能

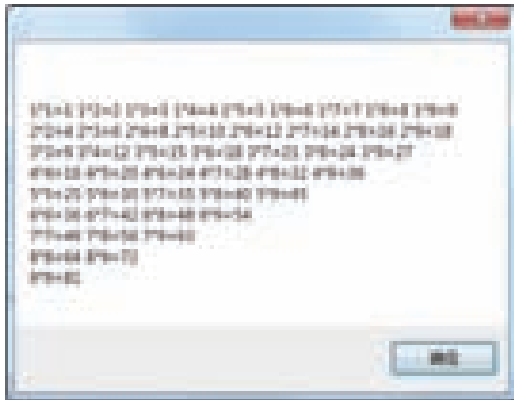


图2.4 使用二维数组生成九九乘法表

遍历到全部元素，以下是典型的处理二维数组的代码，读者自行调试体会。

```

1      public void 二维数组()
2      {
3          int[,] a = { { 1, 2, 3 }, { 4, 5, 6 } };
4          result = a[0, 0] + "\t" + a[1, 2];
5          string[,] arr = new string[2, 4];
6          for (int i = 0; i < 2; i++)//二维数组赋值
7          {
8              for (int j = 0; j < 4; j++)
9              {
10                 arr[i, j] = (i * 10 + j).ToString();
11             }
12         }
13         result += arr[1, 1];
14     }

```

具有同样功能的VBA代码如下：

```

1 Public Sub 二维数组()
2     Dim arr(0 To 1, 0 To 3) As String
3     Dim i As Integer, j As Integer
4     For i = 0 To 1 '二维数组赋值
5         For j = 0 To 3
6             arr(i, j) = (i * 10 + j)
7         Next j
8     Next i
9     result = arr(1, 1)
10 End Sub

```

2.6 条件选择语句

C#中分支控制结构有三种方式：

- 三元运算符
- If...else if...else语句
- switch语句

■ 2.6.1 三元运算符

语法如下：

```
result = condition? trueCase : falseCase
```

首先判断condition是否成立，如果成立，则把trueCase赋给result，反之把falseCase赋给result。

```

1      int q=8;
2      string result = (q % 2 == 1) ? "奇数" : "偶数";
3      MessageBox.Show(q+ "是" +result);

```

上述代码，第2行首先判断q对2求余，结果是否为1，如果是1，则为奇数，否则为偶数。运行后，对话框显示“8是偶数”。

下面再举一个简单选择语句的实例：

```

1      public void 简单选择语句()
2      { //逻辑表达式?结果1:结果2
3          int i = 11;
4          int j = 2;
5          result = i % j == 0 ? "100" : "-100";
6      }

```

运行代码后，result返回-100。

具有同样功能的VBA代码如下：

```

1 Public Sub 简单选择语句()
2     'IIF
3     Dim i As Integer, j As Integer
4     i = 11
5     j = 2
6     result = IIf(i Mod j = 0, 100, -100)
7 End Sub

```

■ 2.6.2 if语句

if语句是最常用的分支控制语句，典型的代码范例如下：

```

1      public void if语句()
2      {
3          int i = -3;
4          if (i > 0)
5              { result = "正数"; }
6          else if (i < 0)
7              { result = "负数"; }
8          else
9              { result = "零"; }
10     }

```

上述代码判断整型变量i的正负情况，如果i大于0，则只执行if后面的语句块；如果i小于0，则只执行elseif后面的语句块；所有条件都不成立，则执行else后面的语句块。

使用if的过程中，要注意if以及else if后面的条件必须用圆括号括起来，每一个分支的语句块最好用花括号括起来。

具有同样功能的VBA代码如下：

```
1 Public Sub if语句()  
2     Dim i As Integer  
3     i = -3  
4     If (i > 0) Then  
5         result = "正数"  
6     ElseIf (i < 0) Then  
7         result = "负数"  
8     Else  
9         result = "零"  
10    End If  
11 End Sub
```

为了说明if语句中花括号的功能，分析如下代码：

```
1     public void TestIf()  
2     {  
3         int i = 0;  
4         if (true)  
5         {  
6             i++;  
7             i++;  
8         }  
9         else  
10            i--;  
11            i--;  
12        MessageBox.Show(i.ToString());  
13    }
```

运行该过程，对话框会显示1，而不是2。这是因为，if语句中的条件为true，表示if分支恒成立，因此执行语句块中的两行自加代码，i变为2。比较一下代码中第10行和第11行，由于第10行属于else后的语句，因此不被执行。但是第11行不属于if…else语句结构范围，因此无论if结构哪一个分支成立，第11总是执行。为此，对话框返回1。

如果我们把第10行和第11行代码用花括号括起来，则构成了一个语句块，属于else分支，这时候对话框会返回2。

如果我们把第10~12行用花括号括起来，则这三行都属于else分支，这时候运行代码后不出现任何对话框。

■ 2.6.3 switch语句

switch语句非常类似于if语句，因为它是根据测试的值来有条件地执行代码。

下面的代码用来判断变量i的奇偶性，因此要把比较的数值放在switch后的括号内。i和下面每一个case后的数值作比较，如果相等，则执行其后的语句块。下面代码的含义是：如果i等于1或3或者5，则返回奇数；如果等于2、4、6中的任何一个，则返回偶数；如果所有的case都没匹配上，则执行default后的语句块。


```
1      public void switch语句()  
2      {  
3          int i = 3;  
4          switch (i)  
5          {  
6              case 1:  
7              case 3:  
8              case 5:  
9                  result = "奇数";  
10                 break;  
11              case 2:  
12              case 4:  
13              case 6:  
14                  result = "偶数";  
15                  break;  
16              default:  
17                  result = "不明确";  
18                  break;  
19          }  
20      }
```

具有同样功能的VBA代码如下：

```
1 Public Sub Select语句()  
2     Dim i As Integer  
3     i = 3  
4     Select Case (i)  
5         Case 1, 3, 5  
6             result = "奇数"  
7         Case 2, 4, 6  
8             result = "偶数"  
9         Case Else  
10            result = "不明确"  
11     End Select  
12 End Sub
```

2.7 循环语句

循环，就是指反复执行同一个动作。在C#中，用于循环的结构有：

- while循环
- do循环
- for循环
- foreach循环

2.7.1 while循环

while的语法结构与if语句非常类似，不同的是，if后面的语句块只执行一次，而while

后的语句块反复执行，直至while括号内条件不成立。

```
1      public void while循环()  
2      {  
3          int i = 10;  
4          while (i > 0)  
5          {  
6              result += i + "\n";  
7              i--;  
8          }  
9      }
```

上述代码中，i的初值为10，判断i是否大于0，如果条件成立不断地往result中追加i，并且i自减1。因此result最后返回的结果是：10 9 8…3 2 1，因为i为1时，仍然输出，但是自减之后，i变为0，while后的条件不再成立，跳出循环。

具有同样功能的VBA代码如下：

```
1 Public Sub while循环()  
2     Dim i As Integer  
3     i = 10  
4     While (i > 0)  
5         result = result & i & vbCrLf  
6         i = i - 1  
7     Wend  
8 End Sub
```

■ 2.7.2 do循环

与while循环相比，do循环首先执行一次其后的语句块，然后判断条件是否成立，如果成立则继续执行语句块，如果不成立，则退出do…while循环。

```
1      public void do循环()  
2      {      //至少执行一次循环体内容  
3          int i = 10;  
4          do  
5          {  
6              result += i + "\n";  
7              i--;  
8          }  
9          while (i > 0);  
10     }
```

上述代码i初值为10，首先往result后面追加10，然后自减1，然后判断i是否大于0；如果大于0，则继续执行do后的语句块；如果不大于0，则退出循环。

因此，上述代码result最终显示10，9，8，…，3，2，1这10个数字。

具有同样功能的VBA代码如下：

```

1 Public Sub do循环()
2     '至少执行一次循环体内容
3     Dim i As Integer
4     i = 10
5     Do
6         result = result & i & vbNewLine
7         i = i - 1
8     Loop While (i > 0)
9 End Sub

```

■ 2.7.3 for循环

如果要使用固定次数的循环，则该考虑使用for循环。for循环圆括号内分三个部分，依次是初值设定、条件判断和处理。

```

1     public void for循环()
2     {
3         for (int i = 1; i < 5; i++)
4         {
5             result += i + "\n";
6         }
7     }

```

上述代码中，for循环体内声明一个整型变量i，判断如果i小于5，则执行for循环体的语句块部分，接着自加1，继续判断是否小于5，如此反复执行，直至i不小于5。因此上述代码result最终结果为：1 2 3 4。

注意 for循环体内三个部分是用半角分号隔开，而不是逗号。

具有同样功能的VBA代码如下：

```

1 Public Sub for循环()
2     For i = 1 To 4 Step 1
3         result = result & i & vbNewLine
4     Next i
5 End Sub

```

下面再举一个典型的循环实例加深印象：累加从1到100之间所有的偶数之和。

```

1     public void TestFor()
2     {
3         int total = 0;
4         for (int i = 2; i <= 100; i += 2)
5         {
6             total += i;
7         }
8         MessageBox.Show("偶数和为: " + total);
9     }

```

上述代码中，total的初值为0，循环变量i从2到100，步长为2，每循环一次，就把i累加

给total，对话框显示“偶数和为2550”。

■ 2.7.4 foreach循环

foreach循环用于枚举集合类对象中的各个元素，例如列举字符串中的每个字符、数组中的每一个元素等。

```
1      public void foreach循环()  
2      {  
3          int[] arr = { 1, 3, 5, 7 };  
4          foreach (int i in arr)  
5          {  
6              result += i + "\n";  
7          }  
8      }
```

上述代码中，arr是一个整型数组，foreach循环体内使用变量i代表数组中的每一个元素。

具有同样功能的VBA代码如下：

```
1 Public Sub foreach循环()  
2     Dim arr As Variant, v  
3     arr = Array(1, 3, 5, 7)  
4     For Each v In arr  
5         result = result & v & vbCrLf  
6     Next v  
7 End Sub
```

2.8 流程控制语句

编程应用中，有时需要提前跳出循环，或提前进入下一轮循环。C#提供了4个命令用来控制循环结构：

- break语句：立即终止循环，跳到循环体外。
- continue语句：立即终止当次循环，继续执行下次循环。
- goto语句：跳出循环，跳转到指定标记好的语句行上。
- return语句：跳出循环，跳出包含该循环的过程或函数。

■ 2.8.1 break语句

```
1      public void break语句()  
2      {  
3          //跳出循环  
4          for (int i = 1; i < 5; i++)  
5          {  
6              if (i > 3)
```

```
6         { break; }
7         else
8         { result += i + "\n"; }
9     }
10 }
```

上述代码，for循环体内嵌套有if语句，当i自加大于3的时候，跳出for循环，直接跳转到代码中第10行。因此变量result最后的结果是1 2 3。

具有同样功能的VBA代码如下：

```
1 Public Sub exitfor语句()
2     '跳出循环,类似于C#中的break语句
3     Dim i As Integer
4     For i = 1 To 4
5         If (i > 3) Then
6             Exit For
7         Else
8             result = result & i & vbNewLine
9         End If
10    Next i
11 End Sub
```

■ 2.8.2 continue语句

continue语句的作用是越过循环体内continue之后的代码，直接进入下一次循环。

```
1     public void continue语句()
2     { //越过本次循环
3         for (int i = 1; i < 10; i++)
4         {
5             if (i == 3 || i == 7)
6             { continue; }
7             else
8             { result += i + "\n"; }
9         }
10    }
```

上述代码，当i循环到3或者7时，什么都不做，直接让i自加继续循环。因此代码中的result最后的结果是：1 2 4 5 6 8 9。

■ 2.8.3 goto语句

goto语句的作用是跳转到指定的标号。

```
1     public void TestGoto()
2     {
3         int i;
4         int total = 0;
```

```

5         for (i = 1; i <= 10; i++)
6         {
7             if (i > 5)
8             {
9                 goto label1;
10            }
11            else
12                total += i;
13        }
14    label1:
15        MessageBox.Show(total.ToString());
16    }

```

上述代码，循环变量*i*从1开始循环，当*i*不大于5时，把*i*累加给*total*，一旦*i*大于5，跳出for循环，跳转到标号为label1的第14行，显示一个对话框。因此对话框显示的结果为1~5的加和：15。

具有同样功能的VBA代码如下：

```

1 Sub testGoto()
2     Dim i As Integer
3     Dim total As Integer
4     For i = 1 To 10
5         If i > 5 Then
6             GoTo label1
7         Else
8             total = total + i
9         End If
10    Next i
11 label1:
12    MsgBox total
13 End Sub

```

■ 2.8.4 return语句

return语句不只用于循环体内，也可以用在一般代码中。下面的代码，当*i*等于3时，代码将直接跳转到第11行。因此result的最终结果是：1 2。

```

1     public void return语句()
2     { //return是跳出过程,break是跳出循环
3         for (int i = 1; i < 10; i++)
4         {
5             if (i == 3 || i == 7)
6             { return; }
7             else
8                 { result += i + "\n"; }
9         }
10        result += "循环体外的语句;\n";
11    }

```

具有同样功能的VBA代码如下：

```
1 Public Sub exitsub语句()  
2     '类似于C#中的return语句  
3     Dim i As Integer  
4     For i = 1 To 10  
5         If (i = 3 Or i = 7) Then  
6             Exit Sub  
7         Else  
8             result = result & i & vbNewLine  
9         End If  
10    Next i  
11    result = result & "循环体外的语句"  
12 End Sub
```

2.9 输出对话框 (MessageBox)

我们经常使用消息对话框给用户一定的信息提示，如在操作过程中遇到错误或程序异常，也会使用对话框。在C#中，MessageBox消息对话框位于System.Windows.Forms命名空间中，一般的C#项目不会自动在每一个类模块上面加上这条指令。因此，要想让MessageBox正常使用，一定要确认是否声明了这条指令。对于C#窗体应用程序或者新创建的窗体类，一般会默认具有这条指令，不需用户再加。

一般情况下，一个消息对话框包含信息提示文字内容、消息对话框的标题文字、用户响应的按钮及信息图标等内容。C#中允许开发人员根据自己的需要设置相应内容，创建符合自己要求的信息对话框。

2.9.1 MessageBox语法

MessageBox的完整语法格式如下：

```
MessageBox.Show(String, String, MessageBoxButtons, MessageBoxIcon,  
MessageBoxResult)
```

其功能是显示一个消息框，该消息框包含消息、标题栏标题、按钮和图标，并且接受默认消息框结果、遵从指定选项并返回结果。

最简单的对话框语法是MessageBox.Show(String)，也就是说，其他参数接受默认，只规定显示内容即可。

注意 显示内容这一参数必须是String类型的结果值，如果是要显示其他类型数据，请使用ToString()方法转换。

下面这个示例自定义了以下5个参数：

- 显示内容
- 标题文字
- 按钮
- 对话框图标
- 默认按钮

如下代码执行后，将出现一个含有信息图标的对话框，运行结果如图2.5所示。



图2.5 MessageBox函数用法

```
1 MessageBox.Show("显示内容", "改变标题文字", MessageBoxButtons.OKCancel,
  MessageBoxIcon.Question, MessageBoxDefaultButton.Button2);
```

我们仔细看一下，该对话框在显示内容的左侧出现一个问题图标，图标由MessageBoxIcon.Question这个参数来规定。“确定”和“取消”两个按钮由参数MessageBoxButtons.OKCancel规定。对话框启动后，默认已选中“取消”按钮，这由MessageBoxDefaultButton.Button2规定。

通过这个示例可以看出显示内容和标题文字的规定很简单，因此详细介绍一下后面三个选项的用法。

■ 2.9.2 自定义对话框的按钮

在代码的MessageBoxButton后输入小数点，可以看到如下6种样式的按钮组合，如图2.6所示。

- 取消 重试 忽略
- 确定
- 确定 取消
- 重试 取消
- 是否
- 是否 取消

在实际编程时根据需要进行设置即可。



图2.6 MessageBox函数参数提示

这里一定要思考一个问题，就是规定这些按钮后，单击不同的按钮，是否有不同的响应过程呢？稍后讨论对话框的返回值的问题。

2.9.3 自定义对话框的图标

对话框显示的图标用MessageBoxIcon枚举常量表示，如表2.7所示。

表2.7 MessageBoxIcon枚举常量

成员名称	描述
Asterisk	该符号是由一个圆圈及其中的小写字母 i 组成的
Error	该符号是由一个红色背景的圆圈及其中的白色 X 组成的
Exclamation	该符号是由一个黄色背景的三角形及其中的一个感叹号组成的
Hand	该符号是由一个红色背景的圆圈及其中的白色 X 组成的
Information	该符号是由一个圆圈及其中的小写字母 i 组成的
None	消息框未包含符号
Question	该符号是由一个圆圈和其中的一个问号组成的
Stop	该符号是由一个红色背景的圆圈及其中的白色 X 组成的
Warning	该符号是由一个黄色背景的三角形及其中的一个感叹号组成的

注意 不必死记硬背这些常量，编写代码时有自动成员提示。

2.9.4 自定义对话框默认按钮

MessageBoxDefaultButton参数规定了对话框显示后哪一个按钮是已选中按钮。

MessageBox默认按钮常量如表2.8所示。

表2.8 MessageBox默认按钮常量

名称	含义
Button1	消息框上的第一个按钮是默认按钮
Button2	消息框上的第二个按钮是默认按钮
Button3	消息框上的第三个按钮是默认按钮

2.9.5 处理对话框的用户响应

前面讲到，对话框上可以规定多个按钮，如何判断单击了哪一个按钮呢？可以把

MessageBox.Show赋给一个DialogResult类型的变量。

```
1         DialogResult v;  
2         v = MessageBox.Show("是否提交试卷?", "询问", MessageBoxButtons.YesNo,  
3         MessageBoxIcon.Question, MessageBoxDefaultButton.Button2);  
4         if (v == DialogResult.Yes)  
5             this.Text = "提交";  
6         else if (v == DialogResult.No)  
7             this.Text = "不提交";
```

上述代码执行后，会出现一个带有“是”和“否”两个按钮的询问对话框，如图2.7所示。

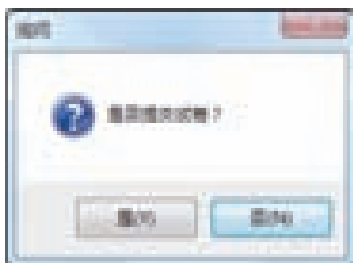


图2.7 处理对话框的用户响应

根据对话框的返回值，使用if或switch结构，分支对待所单击的按钮。上面的例子中，假设用户单击了对话框的“否(N)”按钮，变量v就等于DialogResult.No。

2.10 输入对话框 (InputBox)

有些场合下，需要弹出一个对话框，要求用户输入内容，这时可以使用输入对话框。

C#本身没有提供对话框，因此需要为项目添加“Microsoft.VisualBasic”这个引用，如图2.8所示。

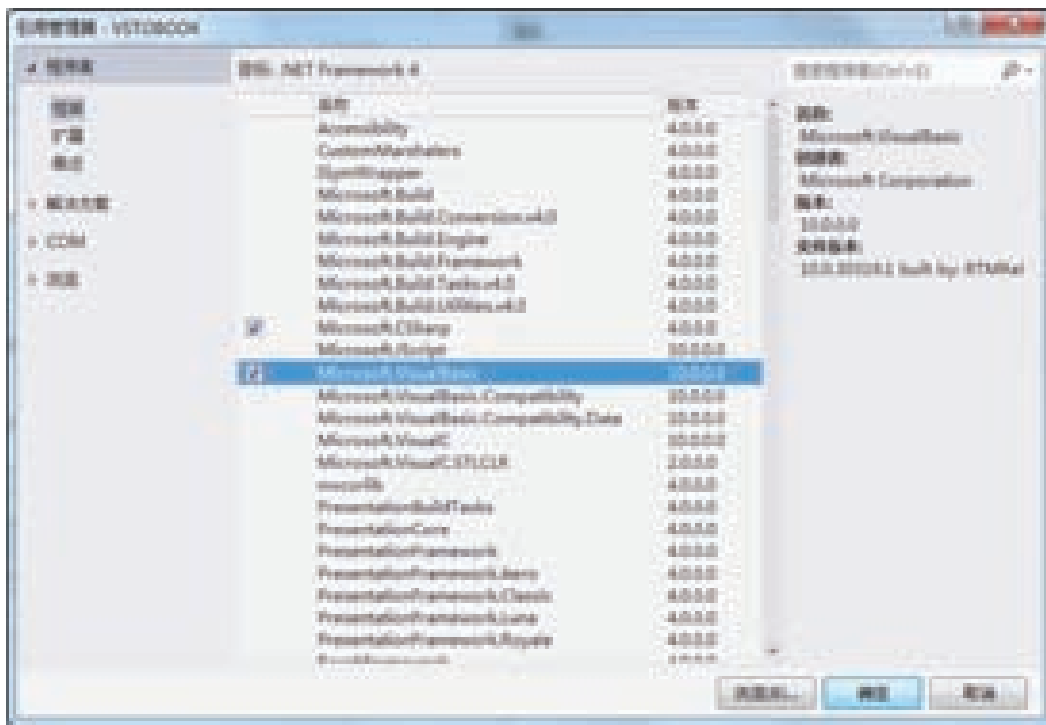


图2.8 引用Visual Basic

单击【项目/添加引用】，在引用对话框中单击【程序集/框架/Microsoft.VisualBasic】。

然后在类模块顶部使用指令“using VB = Microsoft.VisualBasic;”。

下面的代码，让用户输入姓名：

```
string result = VB.Interaction.InputBox("请输入您的姓名: ", "输入框");
```

运行结果如图2.9所示。

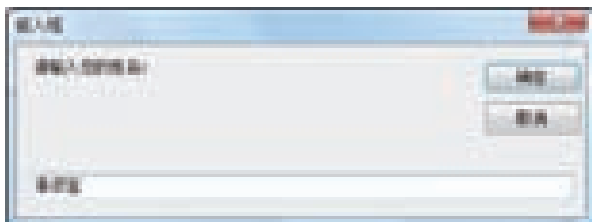


图2.9 输入框

用户输入完的内容赋给字符串变量result，以便程序使用该值。

2.11 过程与函数

C#中过程和函数没有明显的区别，两者统称为函数。一般地，我们把无返回值的称作过程，而有返回值的称作函数。过程和函数的作用是把具有特定的功能和目的的代码段放在一起的一个单元，这样做的目的是便于在其他地方调用。

2.11.1 过程与函数的定义

C#定义过程和函数的语法分为以下5个部分：

- 作用范围：public|private
- 是否静态：static
- 返回类型：void|其他类型
- 过程与函数名称
- 参数列表

下面以计算圆柱的体积为例，说明过程与函数的定义。圆柱体积的计算公式为：

$$V = \pi r^2 h$$

公式中 π 为圆周率， r 为圆柱底面半径， h 为圆柱的高。

```
1      public double V(double r, double h)
2      {
3          const double pi = 3.14159;
4          return pi*r*r*h;
5      }
```

上述代码定义的函数V，包含半径和高两个参数，由于返回的体积也是双精度类型，所以函数定义部分V前面要加上double，而不是void。代码中第1行最前面的关键词public表示这个函数是公有的，如果是private，则表示是私有的。

■ 2.11.2 过程与函数的调用

在其他地方调用过程和函数时，写上过程和函数名称以及一对括号，括号内部写入传递的参数即可。

例如，我们在窗体的Load事件中调用上节的圆柱体积函数：

```
1      private void Form1_Load(object sender, EventArgs e)
2      {
3          double vol;
4          double R = 2, H = 10;
5          vol= V(R,H);
6          MessageBox.Show(vol.ToString());
7      }
```

上述代码中，计算半径为2、高为10的圆柱的体积。其中R和H是实际参数，第5行代码调用了圆柱体积函数。当窗体启动时，对话框中显示125.6636。

调用函数时，可以在参数列表中写上形式参数，如果明确地写上形式参数名称，则各个参数的出现顺序可以不分先后。例如，“vol= V(r:R,h:H);”表示计算半径为R、高为H的圆柱的体积；而“vol= V(h:4, r:7);”则是计算半径为7、高为4的圆柱的体积。注意到这里故意把半径部分写在了后面，但是同样不影响计算结果。

如果定义的函数没有返回值，则在定义函数时要使用void类型，而且在函数体内部，不需要使用return来返回值。

下面的代码说明了无返回值函数的定义和调用：

```
1      public void 过程的定义和调用()
2      {
3          proc();//无参数过程
4          proc2(s: "hello", i: 123);//带参数过程
5      }
6      void proc()
7      {
8          result = "proc";
9      }
10     void proc2(int i, string s)
11     {
12         result = i + s;
13     }
```

具有同样功能的VBA代码如下：

```

1 Public Sub 过程的定义和调用()
2     proc '无参数过程
3     Call proc2(s:="hello", i:=123) '带参数过程
4 End Sub
5 Sub proc()
6     result = "proc"
7 End Sub
8 Sub proc2(i As Integer, s As String)
9     result = i & s
10 End Sub

```

下面的代码说明了函数的定义和调用：

```

1         public void 函数的定义和调用()
2         {
3             result = c(b: 4, a: 1).ToString();//可以颠倒参数顺序
4         }
5         int c(int a, int b)
6         { //返回2个参数的平方之差
7             return a * a - b * b;
8         }

```

具有同样功能的VBA代码如下：


```

1 Public Sub 函数的定义和调用()
2     result = c(b:=4, a:=1) '可以颠倒参数顺序
3 End Sub
4 Function c(a As Integer, b As Integer) As Integer
5     '返回2个参数的平方之差
6     c = a * a - b * b
7 End Function

```

2.12 类的创建和使用

C#中，类可以拥有自己的成员和函数，每个类以文件的形式存储于项目文件夹中。从功能上分，类还分为非静态类和静态类。

 本节视频：类的创建和使用.wmv

2.12.1 非静态类

非静态类在被调用时，需要使用new关键字实例化方可访问和使用类的成员。下面通过一个例子来说明非静态类的用法。

启动Visual Studio 2012，创建一个名为“WindowsFormsApplication20160625”的窗体应用程序，在窗体Form1上放入一个button按钮控件。

单击菜单【项目/添加类】，在出现的对话框中重命名类名为UCase.cs，如图2.10所示。


```

21         return cnt;
22     }
23 }
24 }

```

上述代码中，第11~22行为笔者添加，其余部分为模板自动生成。`countCapital`函数的实现原理是，用`foreach`循环遍历`source`字符串中的每一个字符，如果字符是大写字母，则给`cnt`自加1，最后函数返回`cnt`。

下面演示如何在窗体中调用该类中的函数。

回到窗体的设计视图，双击`button1`，修改其单击事件过程：

```

1     private void button1_Click(object sender, EventArgs e)
2     {
3         UCase U = new UCase();
4         MessageBox.Show(U.CountCapital("Microsoft Office PPT").ToString());
5     }

```

需要特别注意的是，第3行代码声明了一个`Ucase`变量`U`，那么`U`就是`UCase`类的一个实例，接下来在第4行中，使用`U.CountCapital`就调用了这个类中的函数。

启动调试，单击窗体上的按钮，对话框中出现5，这是因为字符串中含有5个大写字母。

■ 2.12.2 静态类

静态类与非静态类的区别是，在被调用时，不需要使用`new`关键字来实例化，使用引用类名称即可。在实际编程应用中，可以利用这一特点，把一些共用的变量或函数放在静态类中。

往项目中添加静态类的方法和一般的类添加步骤是一样的。下面举一个例子，通过单击窗体上的按钮，在按钮上显示已经单击了几次。

在上一小节的窗体应用程序中继续添加一个名为`Counter`的类，编辑该类中的代码为：

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace WindowsFormsApplication20160625
8  {
9      public static class Counter
10     {
11         public static int times = 0;
12     }
13 }

```

注意代码中第9行`class Counter`前面加了`public static`，这说明该类是静态类。该类中定义了一个静态的公有整型变量`times`。

在窗体的设计视图中继续添加一个名为button2的按钮控件，双击该按钮，编辑其Click事件：

```
1     private void button2_Click(object sender, EventArgs e)
2     {
3         Counter.times++;
4         this.button2.Text = "已经单击: " + Counter.times + "次。";
5     }
```

启动调试，每单击一次button2按钮，该按钮的标题文字就显示总共单击它的次数。要使用静态类中的成员，只需要使用Counter.times即可，无需实例化。

2.13 using指令

C#中的using指令写在类模块的顶部，using的作用是把外部库导入进来，从而可以访问这些外部库的成员。

比如System.Windows.Forms这个外部库，里面包含了MessageBox这个成员，也就是说，想显示一个对话框，就需要这样写代码：

```
System.Windows.Forms.MessageBox.Show("hello");
```

我们可以看出，为了显示hello这么短一个单词，代码中需要写这么长一句，因此，可以在模块顶部写上“using System.Windows.Forms;”，那么在以后写代码时，只需要写MessageBox.Show就可以显示对话框了。

此外，还可以使用简称或别名来简化。例如，在类模块顶部写上：

```
using FM = System.Windows.Forms;
```

那么在代码中，使用FM.MessageBox.Show也可以显示对话框。因为使用别名FM来代替System.Windows.Forms。

2.14 错误处理

异常是在运行期间代码中产生的错误，或者由于代码调用的函数产生的错误。

例如我们用C#制作了一个乘法计算器，用户在窗体上的两个文本框分别输入被乘数和乘数，单击一个按钮，在对话框中给出两个数的乘积。但是有些时候，用户在文本框中输入的不是数字，而是其他字符，那么造成乘法无法算出，如果不加错误处理，将会导致程序意外终止。

C#语言包含结构化异常处理的语法，用于异常处理的三个关键字是：

- try：包含抛出异常的代码。

- **catch**: 包含抛出异常时执行的代码。
- **finally**: 包含始终会执行的代码。

下面的示例, 在try块中试图提取数组i中下标为3的元素, 实际上该数组的最大下标为2, 故此一定会出错。出错后, 首先执行catch块内的代码, 然后执行finally中的代码。

```
1      public void 错误处理()  
2      {  
3          try  
4          {  
5              int[] i = { 1, 3, 5 };  
6              result = i[3].ToString(); //数组下标越界  
7          }  
8          catch (SystemException ex)  
9          {  
10             result = ex.Message;  
11          }  
12          finally  
13          {  
14             result += "\nfinally";  
15          }  
16      }
```

以下是VBA中的错误处理方式:

```
1 Public Sub 错误处理()  
2     On Error GoTo Err1:  
3     Dim i(1 To 3) As String  
4     result = i(4) '数组下标越界  
5 Exit Sub  
6 Err1:  
7     result = Err.Description  
8 End Sub
```

本章要点回顾

- C#中的变量在使用前一定要声明, 并且变量名称严格区分大小写。
- 字符串处理过程中, 掌握换行符、制表位等常用转义字符的使用技巧。理解C#字符与字符串的联系和区别。
- 掌握逻辑运算中的与或非连接运算符, 这部分知识是选择结构的基础。
- 理解数组的声明和初始化方法, 数组元素的访问和遍历。
- 选择结构与循环结构以及流程的跳转是语法中的重点知识。
- 函数的定义以及函数的调用, 要注意参数的正确传递。
- 类的使用技术, 可以在类中定义变量、函数和属性成员。注意一般类和静态类的区别。

第3章

C#进阶技术

第2章讲述了C#的语法基础，但是在实际编程应用中，还需要掌握一些相对高级一点的任务方能解决复杂一点的问题。

本章介绍如何用C#操作文件和文件夹，正则表达式和字典的运用，窗体与控件的设计技术等。

3.1 文件与文件夹操作

3.1.1 System.IO命名空间

System.IO命名空间包含允许读写文件和数据流的类型以及提供基本文件和目录支持的类型。System.IO常用的类如表3.1所示。

表3.1 System.IO常用的类

类	描述
BinaryReader	用特定的编码将基元数据类型读作二进制值
BinaryWriter	以二进制形式将基元类型写入流，并支持用特定的编码写入字符串
Directory	公开用于通过目录和子目录进行创建、移动和枚举的静态方法
DirectoryInfo	公开用于通过目录和子目录进行创建、移动和枚举的实例方法
DriveInfo	提供对有关驱动器的信息的访问
File	提供用于创建、复制、删除、移动和打开单一文件的静态方法，并协助创建 FileStream 对象
FileInfo	提供用于创建、复制、删除、移动和打开文件的属性和实例方法，并且帮助创建 FileStream 对象
FileStream	为文件提供 Stream，既支持同步读写操作，也支持异步读写操作

类	描述
FileSystemInfo	为 FileInfo 和 DirectoryInfo 对象提供基类
Path	对包含文件或目录路径信息的 String 实例执行操作
StreamReader	实现一个 TextReader, 使其以一种特定的编码从字节流中读取字符
StreamWriter	实现一个 TextWriter, 使其以一种特定的编码向流中写入字符
StringReader	实现从字符串进行读取的 TextReader
StringWriter	实现一个用于将信息写入字符串的 TextWriter
TextReader	表示可读取有序字符系列的读取器
TextWriter	表示可以编写一个有序字符系列的编写器, 此类为抽象类

上表中的每一个类都有一系列的用法, 比如File类, 这是一个处理文件的类, 文件的移动、删除、重命名、创建都可以用File类实现。而Path类和Directory类用来处理路径, 也就是文件夹。

■ 3.1.2 文件与文件夹处理实例

下面的范例首先判断了abc.ppt文件是否存在, 如果存在则删除。代码中第9行判断一个文件夹是否存在。代码中第13行用GetFiles方法列出了文件夹中所有文件的完整路径。

```
1      public void 文件与文件夹操作()  
2      {      //using System.IO  
3          bool b = System.IO.File.Exists(@"F:\abc.ppt");  
4          if (b==true)  
5          {  
6              result = b.ToString();  
7              System.IO.File.Delete(@"F:\abc.ppt"); //删除文件  
8          }  
9          bool f = Directory.Exists(@"F:\VSTO");  
10         if (f == true)  
11         {  
12             result =f.ToString();  
13             string[] arr = Directory.GetFiles(@"F:\VSTO"); //列举文件夹中所有文件  
14             路径  
15             for(int i = 0; i < arr.Length; i++)  
16             {  
17                 result += arr[i] + "\n";  
18             }  
19         }  
20     }
```

与此对应的VBA代码如下:

```
1 Public Sub 文件与文件夹操作()  
2     '引用Scripting runtime  
3     Dim FSO As New Scripting.FileSystemObject
```

```

4      Dim b As Boolean, f As Boolean
5      Dim fl As File
6      b = FSO.FileExists("E:\VBA\RegExp\百家姓.txt")
7      If b Then
8          result = CStr(b)
9          FSO.DeleteFile "E:\VBA\RegExp\百家姓.txt" '删除文件
10     End If
11     f = FSO.FolderExists("E:\VBA\RegExp")
12     Iff Then
13         result = CStr(f)
14         For Each fl In FSO.GetFolder("E:\VBA\RegExp").Files '列举文件夹中所有文件
路径
15             result = result & fl.Name & vbNewLine
16         Next fl
17     End If
18 End Sub

```

3.2 文本文件的读写

System.IO命名空间下，FileStream和StreamReader是处理文本文件的。

```

1      public void 文本文件的读写()
2      { //using System.IO
3          string path = @"F:\VSTO\a.txt";
4          FileStream fs = new FileStream(path, FileMode.Create);
5          StreamWriter sw = new StreamWriter(fs);
6          //开始写入
7          sw.WriteLine("Hello VSTO");
8          sw.WriteLine("Second Line");
9          sw.WriteLine("3th Line");
10         //清空缓冲区
11         sw.Flush();
12         //关闭流
13         sw.Close();
14         fs.Close();
15         //读入内容
16         StreamReader sr = new StreamReader(path, Encoding.Default);
17         String line;
18         result = "";
19         while ((line = sr.ReadLine()) != null)
20         {
21             result += line + "\n";
22         }
23     }

```

代码中创建了一个a.txt文本文件，写入内容后关闭文件。代码中第16行利用sr这个对象读取文本文件的每一行内容，把每次读取的内容赋给变量line。

与此对应的VBA代码如下：

```

1 Public Sub 文本文件的读写()
2     '引用Scripting runtime
3     Dim FSO As New Scripting.FileSystemObject
4     Dim path As String
5     path = "E:\VBA\RegExp\百家姓.txt"
6     Dim fs As TextStream
7     Set fs = FSO.CreateTextFile(path, True, False)
8     '开始写入
9     fs.WriteLine ("Hello VSTO")
10    fs.WriteLine ("Second Line")
11    fs.WriteLine ("3th Line")
12    fs.Close
13    '读入内容
14    Set fs = FSO.OpenTextFile(path, ForReading, True)
15    result = ""
16    While fs.AtEndOfStream = False
17        result = result & fs.ReadLine & vbNewLine
18    Wend
19    '如果一次性读取全部内容,使用: result = fs.ReadAll 替换上述While循环体
20    fs.Close
21 End Sub

```

3.3 数据库操作

编程开发中，不可避免地要接触数据库操作。下面仅举一个使用ADODB访问Access数据库的实例。

C#使用ADODB，要预先在项目中加入“Microsoft ActiveX Data Objects 2.8 Library”这个外部引用。

```

1     public void 使用ADODB访问Access数据库()
2     {
3         //添加引用Microsoft ActiveX Data Objects 2.8 Library
4         Recordset rs;
5         Connection cn;
6         cn = new Connection();
7         rs = new Recordset();
8         cn.Open(@"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=E:\
Private\ADOSQLwizard\示例数据源\Access2003数据库.mdb;Persist Security
Info=False;");
9         rs.Open(@"Select * from NewID", cn, CursorTypeEnum.adOpenKeyset,
LockTypeEnum.adLockOptimistic, (int)CommandTypeEnum.adCmdText);
10        foreach (Field fd in rs.Fields)
11        {
12            result += fd.Name + "\n";
13        } //遍历字段名称

```

```

14         }
15         while (rs.EOF == false)
16         {
17             //遍历记录中的"性别"字段
18             result += rs.Fields["性别"].Value+ "\n";
19             rs.MoveNext();
20         }
21         rs.Close();
22         cn.Close();
23     }

```

代码中第8行打开了一个Access2003数据库，第9行执行了一条SQL查询，从NewID表中查询所有字段的所有记录，然后遍历每一个字段的名称，遍历所有记录的“性别”属性值。

与此对应的VBA代码如下：

```

1  Public Sub 使用ADODB访问Access数据库()
2      '添加引用Microsoft ActiveX Data Objects 2.8 Library
3      Dim rs As ADODB.Recordset
4      Dim cn As ADODB.Connection
5      Dim fd As ADODB.Field
6      Set cn = New Connection
7      Set rs = New Recordset
8      cn.Open ("Provider=Microsoft.ACE.OLEDB.12.0;Data Source=E:\VBA\
ADOSQLwizard\示例数据源\Access2003数据库.mdb;Persist Security Info=False")
9      rs.Open "Select * from Tb2", cn, CursorTypeEnum.adOpenKeyset,
LockTypeEnum.adLockOptimistic
10     For Each fd In rs.Fields
11         result = result & fd.Name & vbNewLine
12         '遍历字段名称
13     Next fd
14     While (rs.EOF = False)
15         '遍历记录中的"性别"字段
16         result = result & rs.Fields("性别").Value + vbNewLine
17         rs.MoveNext
18     Wend
19     rs.Close
20     cn.Close
21 End Sub

```

3.4 使用资源文件

在C#项目中，可以添加资源文件，资源文件用来存储字符串或图片等信息，方便程序调用。对于比较短的字符串，直接写在代码中即可，而行数很多的字符串，放在代码中就很不明智了，资源文件是个不错的选择。

下面通过一个实例理解一下资源文件的使用过程。

■ 3.4.1 添加资源文件

启动Visual Studio 2012，新建一个名为“WindowsFormsApplication20160523”的窗体应用程序。

单击菜单【项目/添加新项】，在对话框中选择“资源文件”，重命名为“Libai.resx”，如图3.1所示。

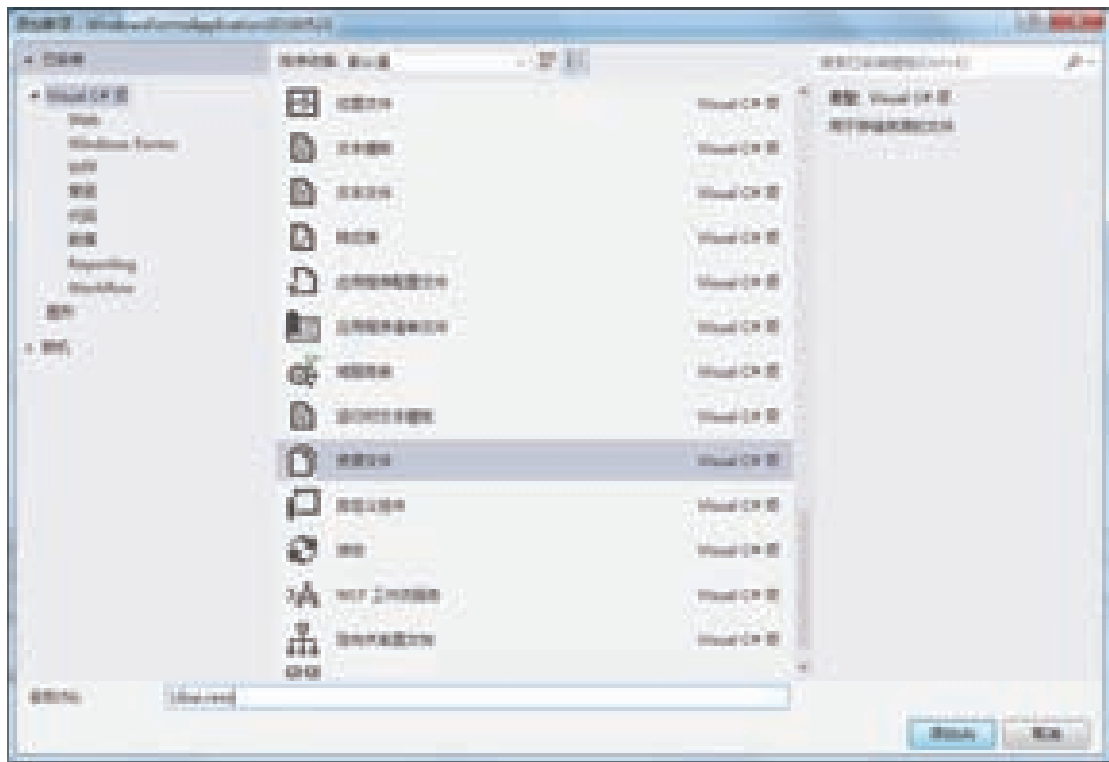


图3.1 添加资源文件

■ 3.4.2 资源文件中的字符串

编辑Libai.resx资源文件，出现的是一个数据表界面，在“名称”列中输入李白的两首古诗名称，在“值”列中输入每首古诗的全文，效果如图3.2所示。

回到Form1的设计视图，在窗体上放入一个RichTextbox控件，然后在窗体的启动事件中写入：

```
1     private void Form1_Load(object sender, EventArgs e)
2     {
3         this.richTextBox1.Text = Libai.白马篇;
4     }
```

当窗体启动后，文本框中出现“白马篇”的全文。

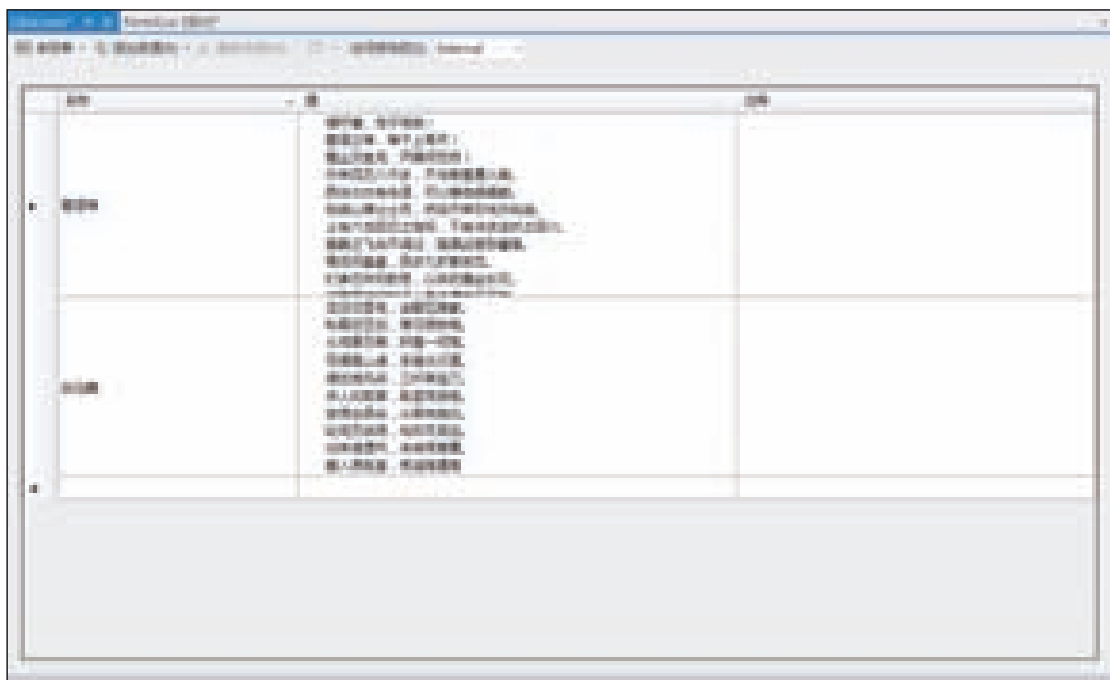


图3.2 编辑资源文件

3.4.3 资源文件中的图像

在网页上搜索李白的照片，另存到计算机，图片名称改为“李白.jpg”。在刚才的资源文件的设计视图中，通过左上角的按钮切换到“图像”，然后单击【添加资源/添加现有文件】，在浏览对话框中找到李白的照片文件，如图3.3所示。



图3.3 为资源文件添加图片

资源添加成功后，回到Form1的窗体设计视图，放入一个PictureBox控件，按下【F4】键进入图片框的属性窗口。设置PictureBox1的SizeMode属性为“AutoSize”，意思是控件尺寸适应图片大小。然后单击PictureBox1的Image属性后的按钮，出现“选择资源”对话框。

在该对话框中，选中“项目资源文件”单选按钮，选择“Libai.resx”，在下面的列表

以上介绍的是窗体在运行前，为图片框指定图片，实际上也可以在窗体启动后，用代码为其指定图片：

让图片框的Image属性等于资源文件的路径即可。

忠与过思之，出师五拜辞。
 新霜时正烈，霜刃响春雷。
 斗地康方鼎，封疆一牧童。
 气雄南山虎，手握大牙旗。
 遇君犹见其，立野再宣力。
 醉人如醉舞，醉后即醉舞。
 我思见君时，从君到海隅。
 我思见君时，从君到海隅。
 我思见君时，从君到海隅。
 我思见君时，从君到海隅。

图3.5 运行效果

3.5 使用正则表达式

在实际的编程应用中，经常会遇到验证字符串是否符合某一规则，或者从字符串中把自己需要的信息提取出来，这就需要用到正则表达式（Regex）。和正则表达式运用有关的术语有以下几个：

- 模式（Pattern）：匹配规则。
- 选项（RegexOptions）：规定是否区分大小写，是否多行等。
- 源字符串（Source）：要匹配的对象。
- 方法：验证、替换、提取、分割。

3.5.1 创建Regex对象

在C#中使用正则表达式，只需要在类模块的using指令中加入“using System.Text.RegularExpressions;”即可。声明一个Regex的语法规则是：

```
Regex obj =new Regex(Pattern,RegexOptions);
```

Pattern就是指匹配字符串，例如，我们在计算机中搜索音乐文件，可以限定文件名模式为“*.mp3”。还有，在Word文档中进行查找和替换时，也是搜索框中输入匹配字符串。

3.5.2 元字符

那么在C#的正则表达式中，Pattern中可以使用元字符或者限定符，如表3.2所示。

表3.2 正则表达式元字符

字符	描述
\	转义字符，将一个具有特殊功能的字符转义为一个普通字符，或反过来
^	匹配输入字符串的开始位置
\$	匹配输入字符串的结束位置
*	匹配前面的零次或多次的子表达式
+	匹配前面的一次或多次的子表达式
?	匹配前面的零次或一次的子表达式
{n}	n是一个非负整数，匹配前面的n次子表达式
{n,}	n是一个非负整数，至少匹配前面的n次子表达式
{n,m}	m和n均为非负整数，其中n<=m，最少匹配n次且最多匹配m次
?	当该字符紧跟在其他限制符（*, +, ?, {n}, {n,}, {n, m}）后面时，匹配模式尽可能少地匹配所搜索的字符串
.	匹配除“\n”之外的任何单个字符
(pattern)	匹配pattern并获取这一匹配

字符	描述
(?:pattern)	匹配pattern但不获取匹配结果
(?=pattern)	正向预查，在任何匹配pattern的字符串开始处匹配查找字符串
(?!pattern)	负向预查，在任何不匹配pattern的字符串开始处匹配查找字符串
x y	匹配x或y。例如，'z food'能匹配"z"或"food"。'(z f)ood'则匹配"zood"或"food"
[xyz]	字符集合。匹配所包含的任意一个字符。例如，'[abc]'可以匹配"plain"中的'a'
[^xyz]	负值字符集合。匹配未包含的任意字符。例如， '[^abc]'可以匹配"plain"中的'p'
[a-z]	匹配指定范围内的任意字符。例如， '[a-z]'可以匹配'a'到'z'范围内的任意小写字母字符
[^a-z]	匹配不在指定范围内的任意字符。例如， '[^a-z]'可以匹配不在'a'~'z'内的任意字符
\b	匹配一个单词边界，指单词和空格间的位置
\B	匹配非单词边界
\d	匹配一个数字字符，等价于[0-9]
\D	匹配一个非数字字符，等价于[^0-9]
\f	匹配一个换页符
\n	匹配一个换行符
\r	匹配一个回车符
\s	匹配任何空白字符，包括空格、制表符、换页符等

说明：由于在正则表达式中“\”、“?”、“*”、“^”、“\$”、“+”、“(”、“)”、“|”、“{”、“[”等字符已经具有一定特殊意义，如果需要用它们的原始意义，则应该对它进行转义，例如希望在字符串中至少有一个“\”，那么正则表达式应该这么写：\\+。

■ 3.5.3 正则表达式选项

正则表达式对象，除了规定其模式字符串外，还可以使用RegexOptions的枚举常量来规定更多的匹配规则，例如是否区分大小写以及是否多行等。正则表达式选项如表3.3所示。

表3.3 正则表达式选项

RegexOptions枚举常量	描述
Compiled	表示编译此模式
CultureInvariant	表示不考虑文化背景
ECMAScript	表示符合ECMAScript，这个值只能和IgnoreCase、Multiline、Compiled连用
ExplicitCapture	表示只保存显式命名的组
IgnoreCase	表示不区分输入的大小写
IgnorePatternWhitespace	表示去掉模式中的非转义空白，并启用由#标记的注释
Multiline	表示多行模式，改变元字符^和\$的含义，它们可以匹配行的开头和结尾
None	表示无设置，此枚举项没有意义

RegexOptions枚举常量	描述
RightToLeft	表示从右向左扫描、匹配，这时，静态的Match方法返回从右向左的第一个匹配
Singleline	表示单行模式，改变元字符的意义，它可以匹配合换行符

如果一个正则表达式同时用到上述两个以上常量，那么常量之间按位或组合即可。例如，`RegexOptions.Compiled | RegexOptions.IgnoreCase`，中间的分隔符是管道符。

■ 3.5.4 正则表达式方法

正则表达式的常用方法有验证、替换、提取和分隔。为了便于讲解每一个方法的使用技巧，创建一个名为“WindowsFormsApplication20160524”的Windows窗体应用程序。

1. 验证 (IsMatch)

窗体上放置一个按钮控件，按钮的单击事件写入如下代码：

```

1      private void button1_Click(object sender, EventArgs e)
2      {
3          string Source = "My Phone:13810872535.";
4          Regex obj = new Regex(@"[\d]{11}", RegexOptions.None);
5          bool result = obj.IsMatch(Source);
6          if (result == true)
7          {
8              MessageBox.Show(Source + "\n包含一个手机号码。");
9          }
10         else
11         {
12             MessageBox.Show(Source + "\n不包含手机号码。");
13         }
14     }

```

代码中，第4行创建了一个新的正则表达式对象，模式是“`[\d]{11}`”，意思是连续的11位数字，第5行使用`IsMatch`方法验证`Source`中是否包含一个手机号。

运行结果如图3.6所示。

可以看出这个例子是部分匹配，也就是说包含即可。还有一种应用是完整匹配，即检测一下源字符串中是否仅仅是一个手机号，没有其他字符。这就需要修改代码中的`Pattern`了。

代码中修改了`Source`字符串，而且`Pattern`改为“`^[^d]{11}$`”，前面的`^`和后面的`$`表示单词边界，也就是说，模式要求必须是以11位数字开头，并且是数字结尾的才返回`true`。如果`Source`中除了手机号，还有其他字符，会返回`false`。

再次运行如下代码，运行结果如图3.7所示。

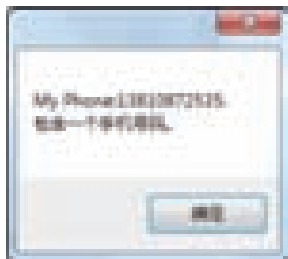


图3.6 使用正则表达式验证字符串



图3.7 字符串的严格匹配

```

1      private void button1_Click(object sender, EventArgs e)
2      {
3          string Source = "13810872535";
4          Regex obj = new Regex(@"^\d{11}$", RegexOptions.None);
5          bool result = obj.IsMatch(Source);
6          if (result == true)
7          {
8              MessageBox.Show(Source + "\n恰好是一个手机号码。");
9          }
10         else
11         {
12             MessageBox.Show(Source + "\n不是11位手机号码。");
13         }
14     }

```

2. 替换 (Replace)

下面一段话是关于中国的介绍。尝试把里面的英文china、cHina、CHINA、China全部替换为“中国”，这就需要用到正则表达式中的不区分大小写了。

china是文明古国之一。古代CHina有着与西方国家不同的科技传统。古代CHINA为世界贡献了诸多发明创造，而且在天文、数学、医药、机械、冶金、陶瓷、纺织、建筑等众多方面发展出了独具特色的先进成果。进入近代社会以后，China学习西方文明，科学研究不断发展。2003年开始的神舟系列飞船的成功发射标志着China成为继苏联及美国之后，第三个有能力独自将人送上太空的国家。

在窗体上放置一个TextBox文本框控件，把控件的Multiline属性设置为true，适当调整其高度和宽度，设置其Text属性，把上面一段文字粘贴进去。再次编辑按钮的单击事件：

```

1      private void button1_Click(object sender, EventArgs e)
2      {
3          string Source = this.textBox1.Text ;
4          Regex obj = new Regex(@"China", RegexOptions.None);
5          this.textBox1.Text = obj.Replace(Source, "中国");
6      }

```

代码中字符串变量初始值为文本框内容，规定了正则表达式的模式为“China”，匹配选项采取默认。替换为汉语的“中国”，替换后的结果再次放回到文本框中。上述代码运

行后，发现文本框内容中前三个英文单词还在，只有后两个替换为汉语了。这是因为默认是区分大小写的，因为前三个英文单词不能匹配到“China”。

这时候，修改代码为“Regex obj =new Regex(@"China",RegexOptions.IgnoreCase);”，再次执行，发现所有单词都变为汉语了。

3. 提取 (Matches)

通过提取Matches方法可以把字符串中需要的信息提取出来。正则表达式使用Matches方法会返回一个MatchCollection的集合，集合中的每一个个体是Match对象，而每一个Match对象有如下三个重要的属性：

- Index：匹配到的内容在原文中的起始位置。
- Length：匹配到的内容的长度。
- Value：匹配到的内容。

例如，获取下面句子中的连续数字：

中国国土面积960万平方公里，56个民族，13亿人口。

修改按钮的单击事件代码：

```
1         private void button1_Click(object sender, EventArgs e)
2         {
3             string Source = "中国国土面积960万平方公里，56个民族，13亿人口。";
4             Regex obj =new Regex(@"\d+",RegexOptions.IgnoreCase);
5             MatchCollection col = obj.Matches(Source);
6             foreach (Match m in col)
7             {
8                 MessageBox.Show("\n位置: " + m.Index + "\n长度: " + m.Length +
9                 "\n值: " + m.Value );
10            }
11        }
```

代码中第5行，col这个变量存储了匹配到的内容信息，需要用foreach结构遍历每一个匹配项。

对话框中显示每一个匹配项的结果。运行结果如图3.8所示。

例如960这个数字，出现在原文的第6个位置，因为第一个字的位置是0。

4. 分隔 (Split)

和提取很类似，Split方法可以把源字符串按照指定模式分隔为字符串数组。

例如，获取下面句子中的所有蔬菜名称：

胡萝卜35公斤土豆231公斤西红柿24公斤白菜1234公斤

可以看到每一个蔬菜后面跟着连续的数字还有“公斤”，显然Pattern为“\d+公斤”。

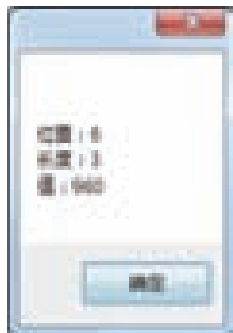


图3.8 字符串的提取

修改按钮的单击事件代码为：

```

1      private void button1_Click(object sender, EventArgs e)
2      {
3          string Source = "胡萝卜35公斤土豆231公斤西红柿24公斤白菜1234公斤";
4          Regex obj = new Regex(@"\d+公斤", RegexOptions.IgnoreCase);
5          string[] arr = obj.Split(Source);
6          string result = "";
7          foreach (string s in arr)
8          {
9              result += s + "\n";
10         }
11         MessageBox.Show(result);
12     }

```

代码中第5行，Split方法返回数组arr，接下来用foreach结构遍历数组中的每一个字符串，运行结果如图3.9所示。

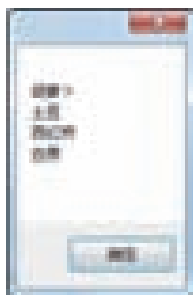


图3.9 字符串的分隔

下面是一些综合的实例代码：

```

1      public void 使用正则表达式()
2      {
3          //using System.Text.RegularExpressions;
4          Regex exp = new Regex(@"^\d{6}$");
5          bool b = exp.IsMatch("59684200");//匹配该字符串是不是恰好6个数字
6          result = b.ToString();
7          exp = new Regex("[a-z]");
8          result += exp.Replace("Visual Studio 2012", ""); //替换每一个小写字母为*
9          exp = new Regex("[a-z]", options: RegexOptions.IgnoreCase | RegexOptions.Multiline); //忽略大小写,多行
10         result += exp.Replace("Visual Studio 2012", ""); //替换每一个字母为*
11         //搜索符合模式的子字符串
12         MatchCollection Col=Regex.Matches("Address:No206,Email:32669315@qq.com,phone:136,etc.",@"\d+");
13         //匹配所有连续的数字
14         for (int i = 0; i < Col.Count; i++)
15         {
16             result += Col[i].Value + "\n";
17         }
18     }

```

与此对应的VBA代码如下：

```

1 Public Sub 使用正则表达式()
2     '添加Microsoft VBScript Regular Expressions 5.5
3     Dim exp As New RegExp
4     exp.Pattern = "^\\d6$"
5     Dim b As Boolean
6     b = exp.Test("59684200") '匹配该字符串是不是恰好6个数字
7     result = CStr(b)
8     exp.Pattern = "(a-z)"
9     result = result & exp.Replace("Visual Studio 2012", "*") '替换每一个小写字母为*
10    '搜索符合模式的子字符串
11    Dim All As VBScript_RegExp_55.MatchCollection, Every As VBScript_RegExp_
12    Const Source As String = "Address:No206,Email:32669315qq.
13    With exp
14        .Global = True
15        .Pattern = "\\d+"
16        Set All = .Execute(Source)
17        For Each Every In All '遍历每一个子结果的信息
18            With Every
19                Debug.Print .Value, .FirstIndex, .Length
20            End With
21        Next Every
22    End With
23 End Sub

```

■ 3.5.5 正则表达式测试器

笔者制作了分别用于VB/VBA（如图3.10所示）和C#（如图3.11所示）的正则表达式测试器。使用测试器可以不写代码就体会到正则表达式的强大。

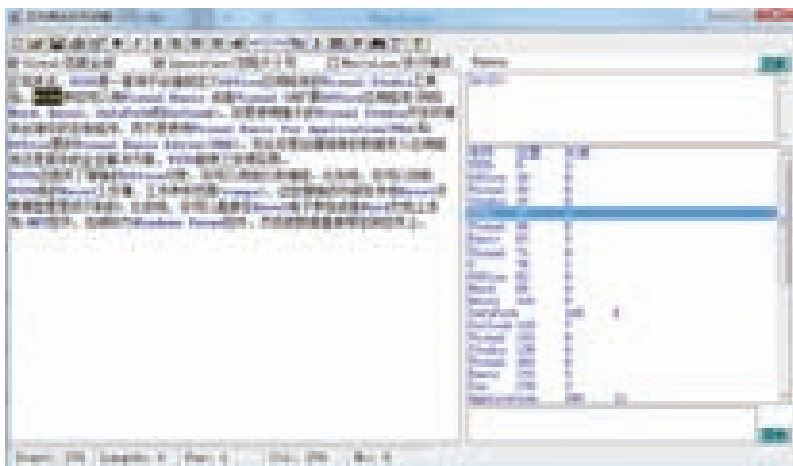


图3.10 VB版正则表达式测试器

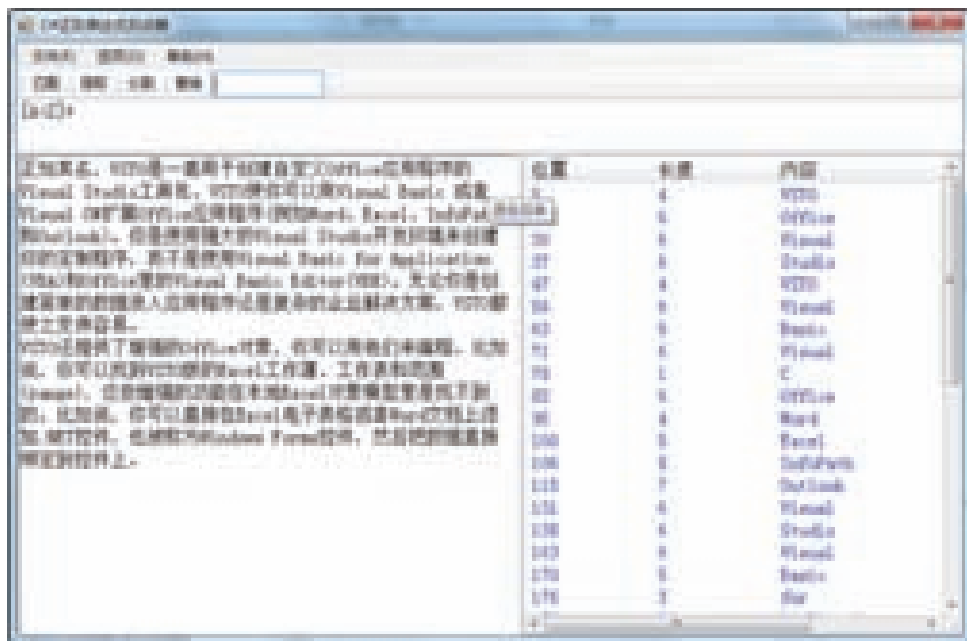


图3.11 C#版正则表达式测试器

3.6 使用字典

字典（dictionary）是一种非常有用的对象，字典用于存储数对（pair），每一个数对包括键（key）和值（value），在形式上和二维数组类似。使用字典可以方便地查询是否存在某个键，查询一个键对应的值等。

3.6.1 字典对象的创建

C#中的字典处于System.Collections.Generic命名空间。

在Visual Studio中创建一个名为“WindowsFormsApplication20160525”的窗体应用程序。打开Form1.cs代码窗口，确认类模块顶部是否已有“using System.Collections.Generic;”指令。如没有，请手动添加这条指令。

下面创建一个字典对象dic，用于存储各国国土面积，因此，字典的键类型是string类型，而值是int或double均可。

窗体Form1上放置一个按钮控件，修改单击事件过程为：

```
1 private void button1_Click(object sender, EventArgs e)
2 {
3     Dictionary<string, int> population = new System.Collections.
    Generic.Dictionary<string, int>();
```

```
4         population.Clear();//清除所有项
5         population.Add("Russia", 1707);
6         population.Add("Canada", 997);
7         population.Add("China", 960);
8         population.Add("America", 936);
9         population.Add("Brazil", 854);
10
11        population.Remove("Canada");//移除键为“Canada”的项目
12
13        int cnt = population.Count;//字典中项目总数
14
15        bool HasKey = population.ContainsKey("China");//判断是否有键为“China”
    的项目
16
17        bool HasValue = population.ContainsValue(850);//判断是否有值为“850”
    的项目
18
19        foreach (KeyValuePair<string, int> kvp in population)
20        {
21            MessageBox.Show(kvp.Key + kvp.Value);
22        }
23    }
```

代码中第3行声明了一个名为`population`的字典对象，该字典键类型是`string`，用于存储国名，值类型是`int`，用于存储对应的国土面积。

第5~9行，依次往`population`中添加了5个国家的国土信息。

注意 字典中所有的键名不可重复使用，也就是说，已经添加过的键不能再添加。键是唯一的。

如果要移除一条已有的项，使用`Remove`方法。代码中第11行，移除了键为“Canada”的项。因此现在只有4个国家的信息。

第15行和第17行，分别判断字典中是否包含特定键和特定值的项。

第19~22行，用`foreach`结构遍历了`population`值的每一项，在对话框中显示每一项的键名和值。

■ 3.6.2 根据键检索值

字典最重要的一个应用就是根据键名找到它对应的值。

检索值的语法为：

```
Value = dic[Key]
```

例如，要查找美国的国土面积，可以使用：`int i = population[“America”]`，`i`这个变量就等于936。这样就根据键名把值取出来了。

如果要更改字典中某项的值，使用语法：

```
dic[Key] =NewValue
```

例如，想把巴西的面积修正为其他数值，使用：`population[“Brazil”] = 888`，这样，字典中该项的值就被修改了。

■ 3.6.3 遍历所有键名

```
1         foreach (string key in population.Keys)
2         {
3             MessageBox.Show(key);
4         }
```

对话框中给出population中所有国名。

■ 3.6.4 遍历所有值

```
1         foreach (int val in population.Values)
2         {
3             MessageBox.Show(val.ToString());
4         }
```

对话框中给出所有国土面积。

注意 遍历所有值的时候，由于population的值类型是int，所以在foreach结构中循环变量val也是int类型。

■ 3.6.5 去除重复

字典还有一大应用是去除重复项。字典对象有一个ContainsKeys方法，用于判断一个键是否已存在。下面的例子，数组arr里包含一些人名，但是存在重复项，使用字典去除重复。

```
1         private void button2_Click(object sender, EventArgs e)
2         {
3             string[] arr = {"李白","杜甫","白居易","杜甫","李商隐","贺知章","白居易",
4                             "孟浩然","李商隐"};
5             Dictionary<string, string> dic = new Dictionary<string, string>();
6             foreach (string s in arr)
7             {
8                 if (dic.ContainsKey(s)==false)
9                 {
10                    dic.Add(s,"");
11                }
12            }
13            foreach (string k in dic.Keys)
```

```

14         {
15             MessageBox.Show(k);
16         }
17     }

```

代码中第3行arr数组中包含一些人名。

第4行声明了一个空的字典对象。

第5~11行，遍历arr数组每一项，如果dic中不含有这个人名，就把这个人名添加到字典中；反之，如果已有该项，则什么也不做。这样就达到了去重的目的。

第13~16行，遍历字典的所有键名，由于字典的键不允许有重复值，所以对话框中每个人的人名仅出现一次。

当然，去重后的结果也可以存入其他对象中加以进一步应用。

以下是关于字典的一些典型应用：

```

1     public void 使用字典()
2     {
3         //添加引用Microsoft Scripting Runtime
4         //using Scripting
5         Dictionary<int, string> dic = new Dictionary<int, string>();
6         dic.Add(1, "one");
7         dic.Add(3, "three");
8         dic.Add(5, "five");
9         dic.Add(7, "seven");
10        bool exist = dic.ContainsKey(4);
11        result = exist.ToString();
12        dic[7] = "Ten";
13        result += dic[7];
14        foreach (string v in dic.Values)
15        {
16            result += v + "\n";
17        }
18        foreach (KeyValuePair<int, string> p in dic)
19        {
20            result += p.Key + "\t" + p.Value + "\n";
21        }
22    }

```

与此对应的VBA代码如下：

```

1 Public Sub 使用字典()
2     '添加引用Microsoft Scripting Runtime
3     Dim dic As New Scripting.Dictionary
4     dic.Add 1, "one"
5     dic.Add 3, "three"
6     dic.Add 5, "five"
7     dic.Add 7, "seven"
8     Dim exist As Boolean

```

```
9      exist = dic.Exists(4) '判断键是否存在
10      result = CStr(exist)
11      dic(7) = "Ten" '值替换
12      result = result & dic(7)
13      For Each v In dic.Items '遍历值
14          result = result & v & vbCrLf
15      Next v
16      For Each v In dic.Keys '遍历键
17          result = result & v & vbCrLf
18      Next v
19 End Sub
```

3.7 窗体设计技术

本节介绍C#窗体与控件的编程技术。

 本节视频：窗体和控件的设计技术.wmv

3.7.1 窗体的显示

窗体（Form）也是一种类对象，使用一个窗体必须创建新实例。

在Visual Studio中创建一个名为“WindowsFormsApplication20150526”的Windows窗体应用程序项目，无需进行任何设置和编码，按下【F5】键就可以看到Form1显示了。这是因为窗体应用程序项目默认有一个窗体，而且该窗体作为项目的启动对象。

如果程序中还需要其他的窗体，就需要用代码来创建并显示它。

第1步：在默认的Form1设计视图中添加一个按钮控件button1。

第2步：单击菜单【项目/添加Windows窗体】，使用默认名称“Form2.cs”。

程序的意图是单击Form1的按钮，显示出Form2来。

第3步：双击Form1的button1按钮，编辑其单击事件过程：

```
1      private void button1_Click(object sender, EventArgs e)
2      {
3          Form2 fm2 = new Form2();
4          fm2.Show();
5      }
```

代码中第3行创建了一个新的窗体实例fm2，那么在代码里操作fm2即可。

注意 绝对不可以使用“Form2.Show();”直接显示窗体2。

此时，启动调试，屏幕上首先出现Form1，然后鼠标单击上面的按钮，接着出现Form2。我们发现在不关闭Form2的情况下，鼠标可以在两个窗体自由切换，这是因为此时两个窗体没有依赖关系，是相对独立的。

1. 作为子窗体显示

实际上，窗体的Show方法可以接受一个参数，如果把上述代码中第4行改为：

```
fm2.Show(this);
```

参数this就是指Form1本身，再次启动调试，发现Form2总是显示在Form1的窗体区域内部，而且总挡在Form1的前面，当Form1最小化的时候，Form2随之最小化。

2. 作为对话框窗体

在实际编程过程中，有时候需要设计成子窗体在打开的情况下，不允许操作母窗体。把上面代码第4行改为：

```
fm2.ShowDialog();
```

这样，当Form2显示出来后，鼠标不能操作Form1，除非把Form2关闭。

■ 3.7.2 窗体的卸载

卸载并关闭一个窗体除了鼠标单击窗体右上角的“关闭”按钮外，还可以使用Dispose方法关闭。

```
1 namespace WindowsFormsApplication20160526
2 {
3     public partial class Form1 : Form
4     {
5         Form2 fm2;
6         public Form1()
7         {
8             InitializeComponent();
9         }
10
11         private void Form1_Load(object sender, EventArgs e)
12         {
13
14         }
15         private void button1_Click(object sender, EventArgs e)
16         {
17             fm2 = new Form2();
18             fm2.Show();
19         }
20         private void button2_Click(object sender, EventArgs e)
21         {
22             fm2.Dispose();
23         }
24     }
25 }
```

代码中第5行声明了一个Form2变量fm2，但是没有实例化。button1的单击事件中，创建新实例并显示Form2。第22行，当单击button2时，会自动关闭Form2。

■ 3.7.3 窗体与控件的事件

在窗体的设计视图中，不仅可以在属性窗口中设置窗体与控件的属性，而且可以通过属性窗口设置事件。通常情况下，双击窗体或控件会自动进入其默认事件的代码中，例如，双击窗体会自动进入Form_Load事件过程，双击按钮会自动进入Button_Click事件。

当然，对于每一种控件的事件不只一个，例如，我们要设计窗体的Resize事件，就需要通过属性窗口实现了。在窗体的设计视图中，鼠标选中窗体，在属性窗口切换到“事件”，找到Resize事件，双击后面的组合框，就自动在代码区域创建了窗体的Resize事件过程，如图3.12所示。

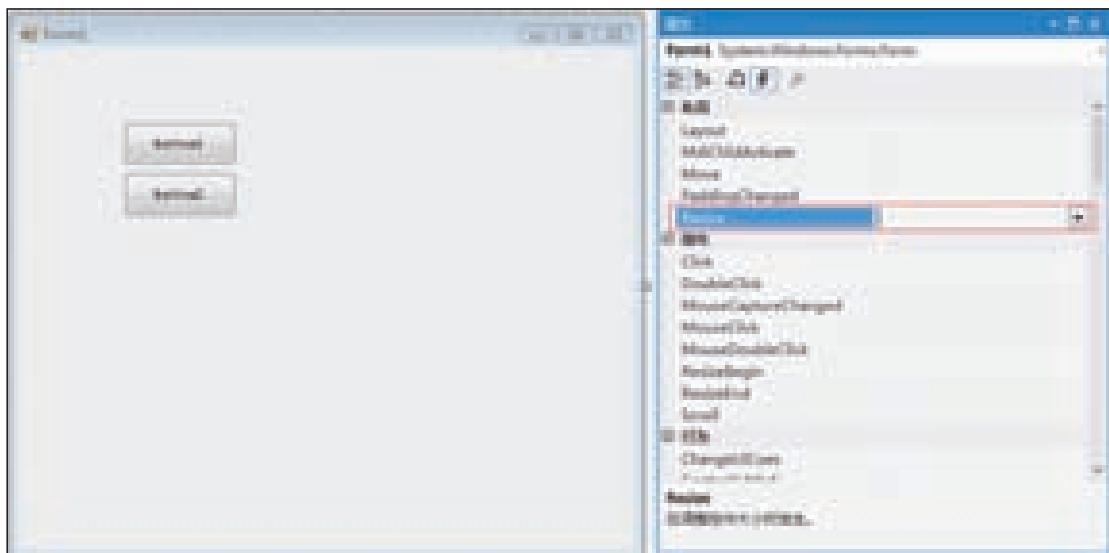


图3.12 设置窗体与控件的事件

适当修改代码为：

```
1     private void Form1_Resize(object sender, EventArgs e)
2     {
3         this.Text = this.Width + " " + this.Height;
4     }
```

Resize事件是指窗体或控件的尺寸发生改变时自动引发的过程。

启动并调试，当鼠标改变窗体的宽度和高度时，窗体的标题文字会显示窗体现在的宽度和高度。

对于按钮、列表框、文本框等常用控件的各种事件，都可以使用属性窗口来创建事件过程，这里不再一一列举。

1. 运行过程中创建事件

通常情况下，从控件工具箱往窗体上放置控件后，该控件没有任何的事件过程。事件过程的设计除了前面所述的在运行前设计期间设置外，还可以在窗体启动后，在运行期间增加或移除事件。

在窗体上放置两个Button控件，一个ListBox控件。两个Button控件的Text属性设置为“创建事件”和“移除事件”，在ListBox的Items属性中，预设列表框中的部分条目。

此时启动调试，当鼠标单击ListBox中的条目时，没有任何响应，这是因为没有为其创建事件过程。

关闭窗体终止调试，双击button1和button2编辑它们的Click事件过程：

```
1      private void button1_Click(object sender, EventArgs e)
2      {
3          this.listBox1.SelectedIndexChanged += new EventHandler(ListBox_
Event); //创建事件
4      }
5      private void ListBox_Event(object sender, EventArgs e)
6      {
7          MessageBox.Show(this.listBox1.SelectedIndex + " " + this.listBox1.
Text);
8      }
9
10     private void button2_Click(object sender, EventArgs e)
11     {
12         this.listBox1.SelectedIndexChanged -= new EventHandler(ListBox_
Event); //移除事件
13     }
```

代码中第3行，使用代码为listBox1创建了一个SelectedIndexChanged事件过程，事件过程名为“ListBox_Event”，为此，在第5行创建这个过程，以便调用。

与此相反，事件过程既可以用代码创建，也可以用代码移除。代码中第12行移除名称为“ListBox_Event”的事件过程。

启动调试，直接单击列表框，不会有任何响应；当单击一次button1后，再去单击列表框，就跳出对话框，显示当前条目的序号和内容。当再单击一次button2后，单击列表框不会有任何响应。运行结果如图3.13所示。

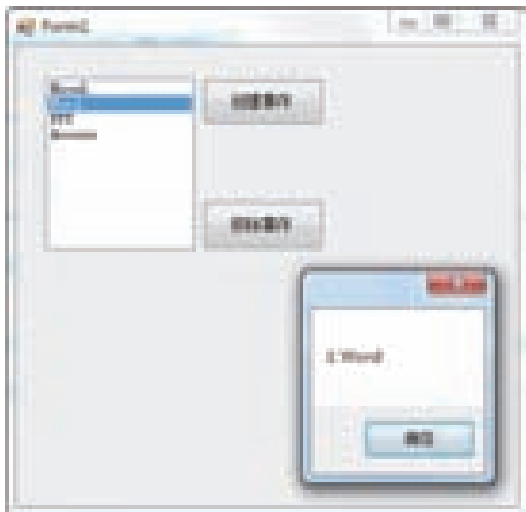


图3.13 事件的添加和移除

2. 响应鼠标单击的事件

控件响应鼠标的事件主要有：

- **MouseDown**：单击控件时引发事件和Click事件的区别是，MouseDown返回的信息参数比较丰富。
- **MouseDown**：按下鼠标按键时引发。
- **MouseUp**：抬起鼠标按键时引发。

MouseDown是指按下鼠标引发，而MouseDown要求按下后并且抬起时才引发，这是这两个事件的区别。

在窗体上放置一个新的ListBox控件，适当调整大小，在属性窗口中，切换到“事件”，双击MouseDown事件，修改代码如下：

```
1      private void listBox1_MouseDown(object sender, MouseEventArgs e)
2      {
3          if (e.Button == MouseButtons.Left)
4              MessageBox.Show("你单击了鼠标左键");
5          else if (e.Button == MouseButtons.Right)
6              MessageBox.Show("你单击了鼠标右键");
7          else if (e.Button == MouseButtons.Middle)
8              MessageBox.Show("你单击了鼠标中间滚轮");
9
10
11      }
12      MessageBox.Show("鼠标单击位置: " + e.Location);
```

可以看到MouseDown事件的参数中，有一个MouseEventArgs e，这个参数e主要有如下的鼠标信息返回：

- Button: 返回按下了鼠标的哪一个键, 可以和MouseButtons的枚举常量做比较。
- X: 返回鼠标在控件中的横坐标值。
- Y: 返回鼠标在控件中的纵坐标值。
- Location: X和Y的合体。

注意 坐标的参考基准点是控件的左上角, 而不是窗体的左上角。也就是说, 当恰好单击了控件的左上角时, Location应返回 (X=0, Y=0)。

启动窗体后, 运行结果如图3.14所示。

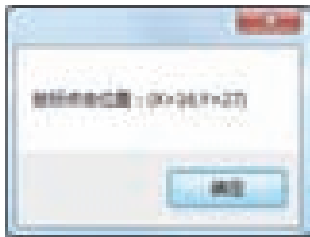


图3.14 控件的MouseDown事件

3. 移动鼠标时的事件

当鼠标在控件上方移动时, 将引发MouseMove事件。

```
1     private void listBox1_MouseMove(object sender, MouseEventArgs e)
2     {
3         this.Text = e.Location.ToString();
4     }
```

MouseMove事件和MouseDown事件非常类似, 也能返回鼠标的按键状态以及鼠标在控件中的位置。不同的是, MouseDown事件要求必须按下鼠标的某个键才引发, 而MouseMove事件要求只要鼠标在移动就引发。

启动调试后, 当鼠标在列表框上方移动时, 窗体的标题随时更新当前坐标值。

4. 响应按键的事件

响应键盘按键的事件有:

- KeyDown: 按下按键引发的事件。
- KeyUp: 抬起按键时引发的事件。

以上两个事件都会返回KeyEventArgs e, 参数e的主要成员如表3.4所示。

表3.4 按键事件的返回参数

名称	描述
Alt	获取一个值, 该值指示是否曾按下 Alt 键
Control	获取一个值, 该值指示是否曾按下 Ctrl 键

名称	描述
Handled	获取或设置一个值，该值指示是否处理过此事件
KeyCode	获取 KeyDown 或 KeyUp 事件的键盘代码
KeyData	获取 KeyDown 或 KeyUp 事件的键数据
KeyValue	获取 KeyDown 或 KeyUp 事件的键盘值
Modifiers	获取 KeyDown 或 KeyUp 事件的修饰符标志。这些标志指示按下 Ctrl、Shift 和 Alt 键的组合
Shift	获取一个值，该值指示是否曾按下 Shift 键
SuppressKeyPress	获取或设置一个值，该值指示键事件是否应传递到基础控件

窗体上放置一个TextBox控件，创建文本框控件的KeyDown事件，代码如下：

```
1 private void textBox1_KeyDown(object sender, KeyEventArgs e)
2 {
3     if (e.Control == true && e.KeyCode == Keys.F3)
4     {
5         MessageBox.Show("你一定按下了组合键【Ctrl+F3】");
6     }
7 }
```

代码中第3行，e.Control返回一个布尔值，能监测到是否按住了Ctrl键，e.KeyCode能监测出按住了其他的键。上述代码，鼠标焦点放在文本框中，只有当同时按下【Ctrl+F3】组合键，才会出现对话框；按下其他键没有任何响应。

也可以使用Modifiers来监测按键：

```
1 private void textBox1_KeyDown(object sender, KeyEventArgs e)
2 {
3     if (e.KeyCode == Keys.F1 && e.Modifiers==Keys.Alt)
4     {
5         MessageBox.Show("你一定按下了组合键【Alt+F1】");
6     }
7 }
```

上述代码中，用e.Modifiers==Keys.Alt来判断是否按下了Alt键。

■ 3.7.4 使用窗体菜单

一个专业的窗体应该包含一个主菜单。C#窗体中创建菜单栏，需要用到MenuStrip控件。

在窗体的设计视图中，从控件工具箱中选择【菜单和工具栏/MenuStrip】，拖动这个控件到窗体上，然后鼠标移动到窗体顶端，依次编辑各项菜单的标题文字，如图3.15所示。

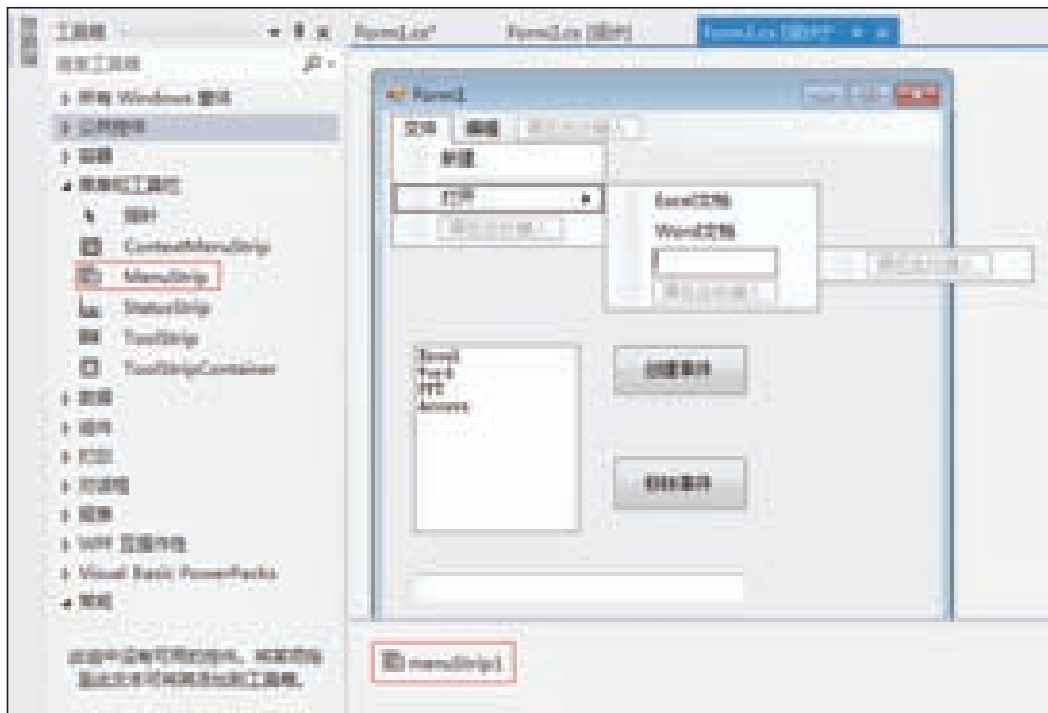


图3.15 使用MenuStrip菜单控件

如图所示，【文件】主菜单的【打开】是一个级联菜单，下面还包含两个子项。

如果要为菜单中某项上面加上分隔线，在设计视图中右击菜单项，选择【插入/Separator】即可在该项上方出现横线，如图3.16所示。

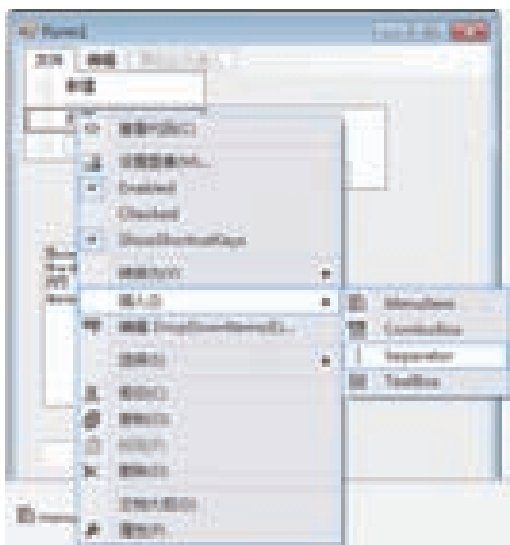


图3.16 插入分隔符

1. 菜单项事件

为菜单项添加事件过程非常简单，只需要在设计视图中双击该菜单项，就自动进入其事件代码中。当然菜单项的事件过程不只Click一个，它和窗体控件一样，也有诸多事件，Click事件是其默认事件。如果要用到其他事件，在设计视图中选择该菜单项，在属性窗口中设定事件过程也可以。

现在回到设计视图，双击窗体的菜单项【文件/新建】，编辑其Click事件过程：

```
1 private void 新建ToolStripMenuItem_Click(object sender, EventArgs e)
2 {
3     MessageBox.Show("你单击了: " + this.新建ToolStripMenuItem.Text);
4 }
```

2. 为菜单项指定快捷键

可以为菜单项指定快捷键，当窗体显示后，除了用鼠标单击菜单项之外，还可以直接按下响应的快捷键来激活菜单项的事件。

在窗体的设计视图中，鼠标选中“新建”菜单项，按下【F4】键显示属性窗口，在属性窗口中找到“ShortcutKeys”，设置其快捷键为【Ctrl+N】，如图3.17所示。

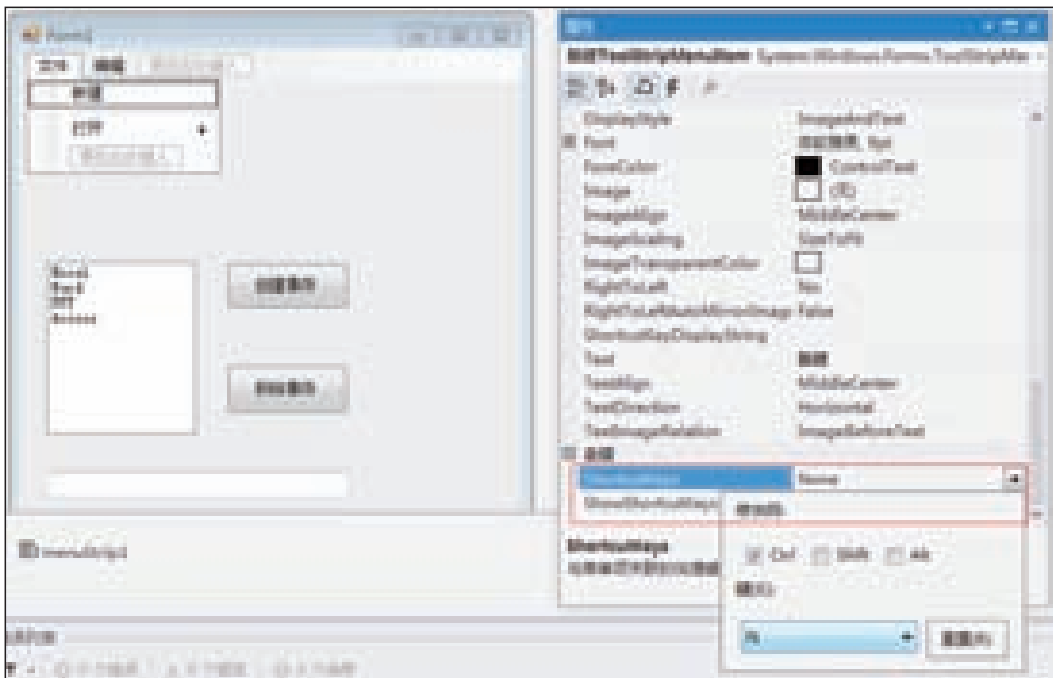


图3.17 为菜单项指定快捷键

再次启动窗体，运行效果如图3.18所示。

当在窗体中按下【Ctrl+N】组合键时，会弹出对话框。

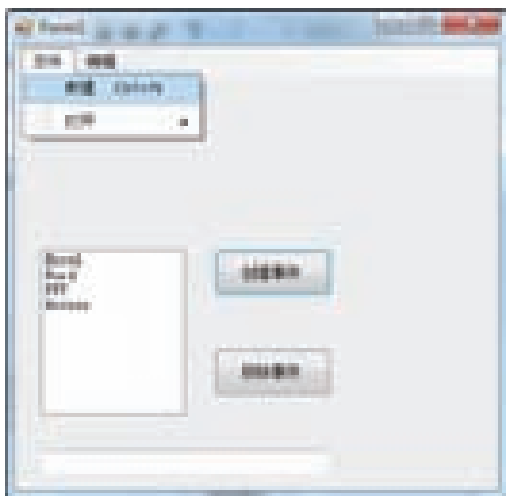


图3.18 菜单的运行效果

■ 3.7.5 使用工具栏

C#提供了ToolStrip控件，用于创建窗体的工具栏。从控件工具箱拖动一个ToolStrip控件到窗体中，就可以编辑工具栏，如图3.19所示。

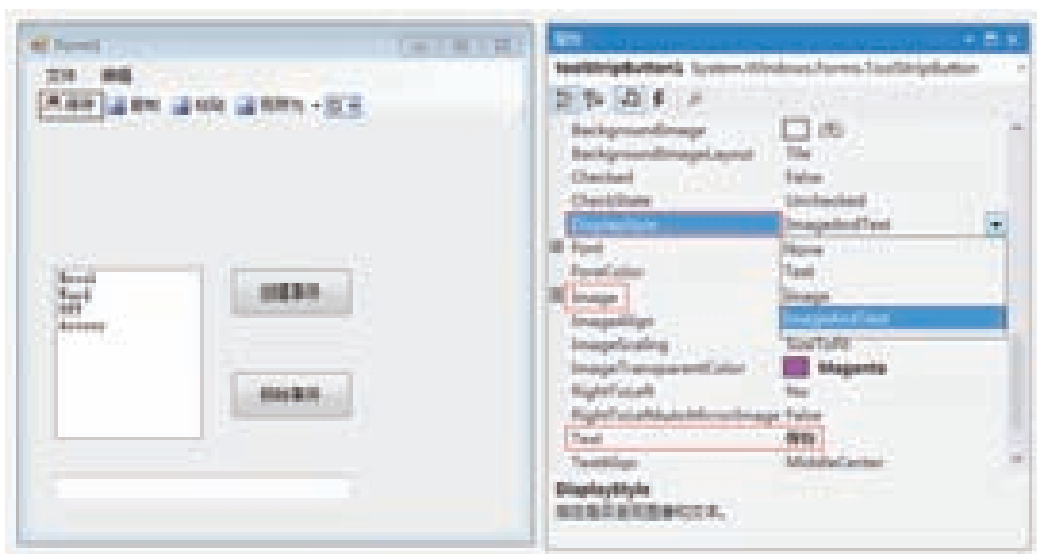


图3.19 使用工具栏控件

对于工具栏按钮的属性设定，主要设定如下几个重要属性：

- DisplayStyle: 显示样式。默认是只显示图标，根据需要可以选择“ImageAndText”。
- Image: 允许自定义图标，可以选择计算机中的jpg等图片作为图标。
- Text: 按钮的标题文字。

工具栏中的弹出式菜单

右击一个工具栏的菜单项，选择【插入/DropDownButton】，如图3.20所示，就可以创建多级弹出式菜单，然后在工具栏设计视图中，依次填写各个子菜单的属性内容。

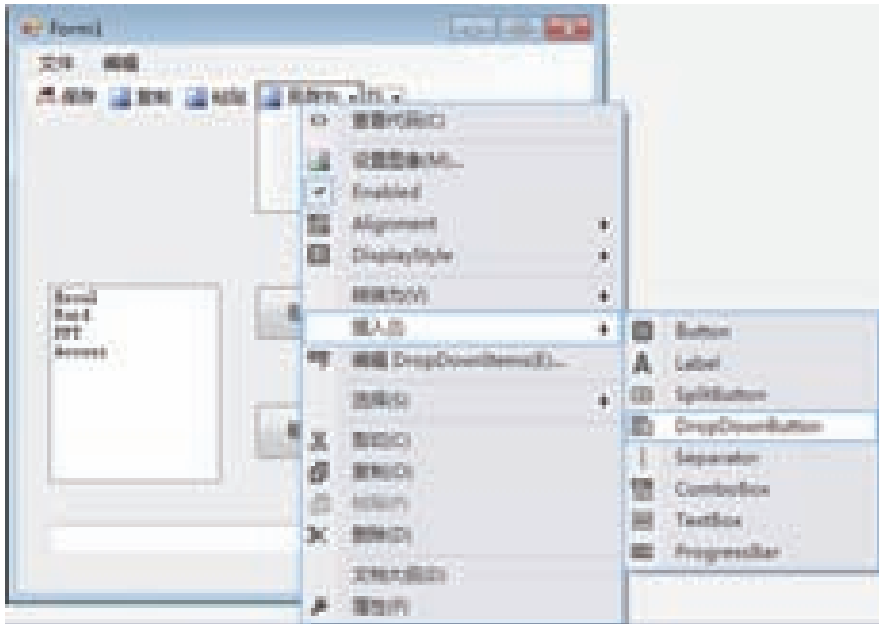


图3.20 工具栏上的级联菜单

启动调试，最终的工具栏运行效果如图3.21所示。



图3.21 运行效果

■ 3.7.6 使用右键菜单

在实际应用中，鼠标右击控件对象时，出现自定义的右键菜单。这需要用到C#中的ContextMenuStrip控件。

在窗体上放置一个ContextMenuStrip控件，然后鼠标选中这个控件，并编辑该右键菜单的每一项，如图3.22所示。

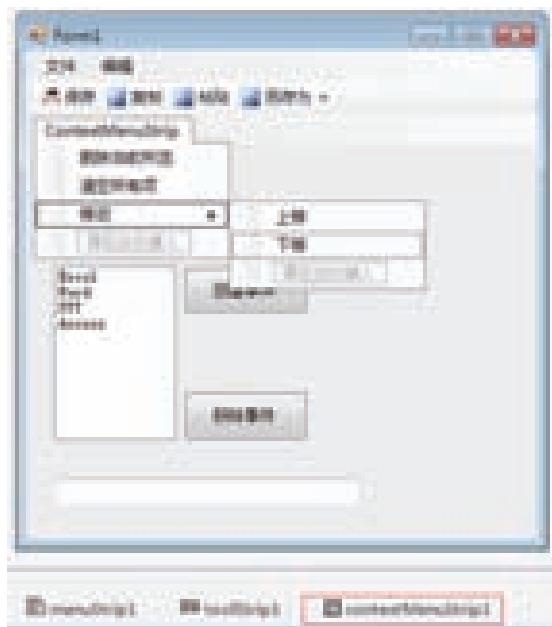


图3.22 使用右键菜单

接下来为控件指定右键菜单。鼠标选中listBox1这个列表框控件，在属性窗口中，设置该控件的ContextMenuStrip属性，默认是“无”，现在改为contextMenuStrip1，如图3.23所示。

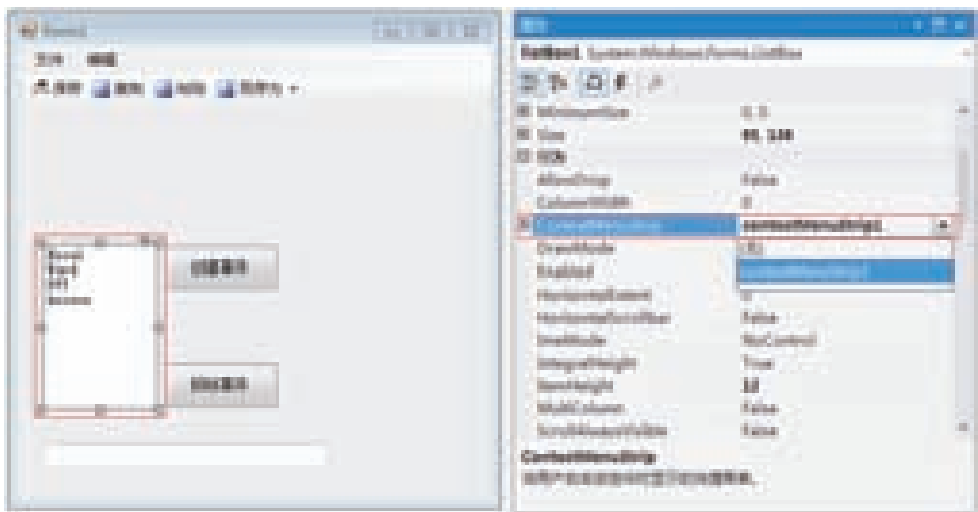


图3.23 为控件指定右键菜单

启动调试，当右击列表框控件时，运行效果如图3.24所示。

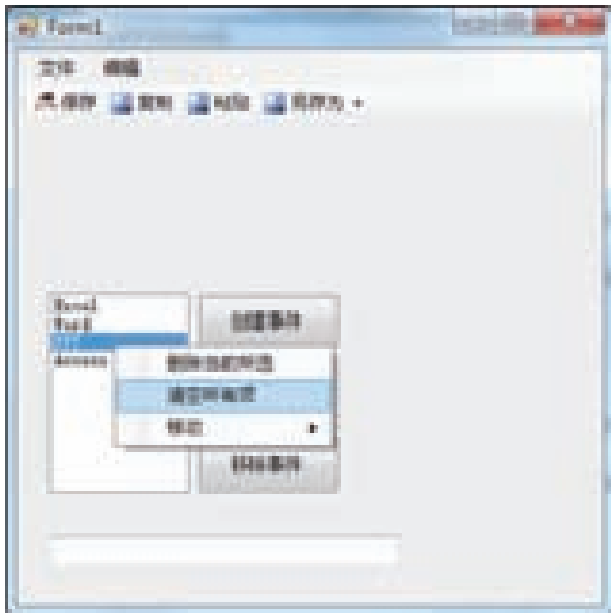


图3.24 运行效果

现在单击右键菜单中的每一项，还没有任何响应，这需要设置其事件过程。终止调试，在设计视图中双击ContextMenuStrip中的“删除当前所选”菜单项，进入其事件过程：

```
1      private void 删除当前所选ToolStripMenuItem_Click(object sender,
EventArgs e)
2      {
3          this.listBox1.Items.RemoveAt(this.listBox1.SelectedIndex);
4      }
5
6      private void 清空所有项ToolStripMenuItem_Click(object sender, EventArgs
e)
7      {
8          this.listBox1.Items.Clear();
9      }
```

再次启动调试，当单击“删除当前所选”菜单项时，列表框会自动移除鼠标选中的条目。代码中第3行，RemoveAt方法的参数是一个序号，其含义是移除第几个条目。而代码中第8行Items.Clear()是清空列表框中所有条目。

■ 3.7.7 使用状态栏

C#中提供了StatusStrip控件来设计窗体左下角的状态栏。

从控件工具箱中拖动一个StatusStrip控件到窗体中，鼠标选中该控件，根据业务需要为状态栏添加面板，如图3.25所示。

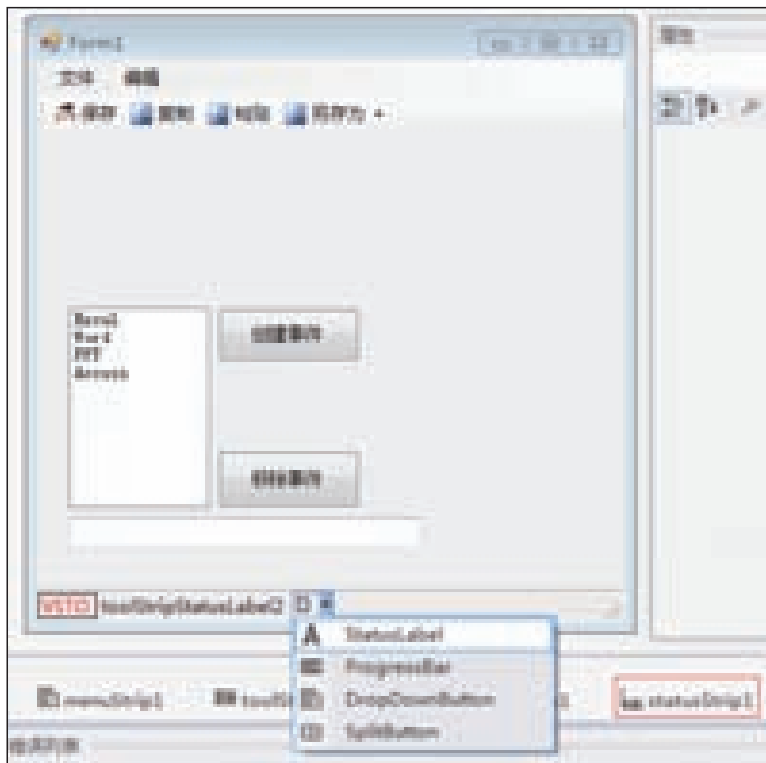


图3.25 使用状态栏

一般情况下，面板中加入StatusLabel控件即可，对于每一个面板内容，可以通过属性窗口进行详细设定。当然这些属性也可以在运行期间使用代码修改。

双击窗体Form1，编辑其Load事件过程：

```
1     private void Form1_Load(object sender, EventArgs e)
2     {
3         this.toolStripStatusLabel2.Text = System.DateTime.Now.
4         ToLongDateString() + " " + System.DateTime.Now.ToLongTimeString();
5     }
```

代码的含义是，当窗体启动后，修改状态栏的第2个面板内容为当前日期和当前系统时间。运行效果如图3.26所示。



图3.26 状态栏的运行效果

■ 3.7.8 使用文件选择对话框

在C#中，模仿Office组件中打开文件的操作，例如在Excel或Word中按下【Ctrl+O】会出现打开文件的对话框。

C#中可以选择控件工具箱中的【对话框/OpenFileDialog】添加到窗体中。为了显示出这个对话框，在窗体Form1的设计视图中，单击主菜单【文件/打开/Excel文档】，双击该菜单项进入事件代码中：

```
1      private void excel文档ToolStripMenuItem_Click(object sender, EventArgs  
e)  
2      {  
3          this.openFileDialog1.Filter = "Word Documents|*.doc|Excel  
Worksheets|*.xls|PowerPoint Presentations|*.ppt|Office Files|*.doc;*.xls;*.  
ppt|All Files|*.*";  
4          this.openFileDialog1.ShowDialog();  
5          this.textBox1.Text = this.openFileDialog1.FileName;  
6      }
```

代码中第3行，为对话框限定了过滤器，让用户只能选择Office的相关文档。第4行使用ShowDialog()方法显示出来。第5行把选择了的文件完整路径放入文本框中。

运行效果如图3.27所示。

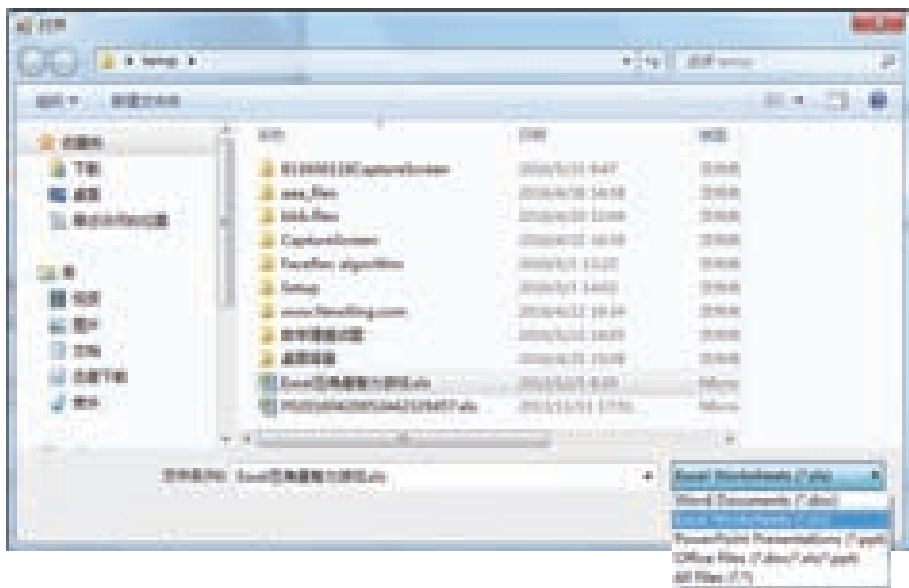


图3.27 文件选择对话框

注意 选择文件后，并不会对该文件进行任何操作，也不会真正打开。只是提取到文件的路径而已。

C#的对话框控件除了上述的文件打开对话框外，还有如下几个重要对话框：

- ColorDialog：颜色选择对话框。
- FolderBrowserDialog：文件夹选择对话框。
- FontDialog：字体选择对话框。
- SaveFileDialog：保存文件对话框。

这些对话框的设计方法和OpenFileDialog对话框类似，限于篇幅，读者自行尝试。

■ 3.7.9 运行期间动态增删控件

除了在运行前设计视图中手工添加创建窗体和控件外，还可以在运行期间用代码动态添加控件，同理，也可以动态删除已有的控件。

关于窗体控件的添加和删除，使用Form.Controls来实现，Controls表示的是一个窗体内的所有控件，以下是它的一些重要方法：

- Controls.Add(control)：增加一个控件。
- Controls.AddRange(control[])：增加一个控件数组。
- Controls.Remove(control)：移除一个控件。

1. 添加控件

在Visual Studio中创建一个名为“WindowsFormsApplication20160527”的窗体应用程

序项目，从控件工具箱拖动一个Button按钮控件到窗体上，button1的标题文字修改为“增加列表框”，双击button1编辑其单击事件过程代码：

```
1 namespace WindowsFormsApplication20160527
2 {
3     public partial class Form1 : Form
4     {
5         ListBox LB;
6         public Form1()
7         {
8             InitializeComponent();
9         }
10
11        private void button1_Click(object sender, EventArgs e)
12        {
13            LB= new ListBox();
14            LB.Left = 30;
15            LB.Top = 50;
16            LB.Visible = true;
17            LB.Items.Add("张三");
18            LB.Items.Add("李四");
19            this.Controls.Add(LB);
20        }
21    }
22 }
```

代码中第5行，声明了一个ListBox的控件对象LB，在button1的Click事件代码中，实例化LB，然后设置了LB的部分属性值，而且为该列表框预先添加了2个条目。第19行，“this.Controls.Add(LB);”这句是关键代码，把LB添加到了Form1中。

启动调试，窗体显示出来后，并没有列表框在上面，当单击了按钮后，运行效果如图3.28所示。

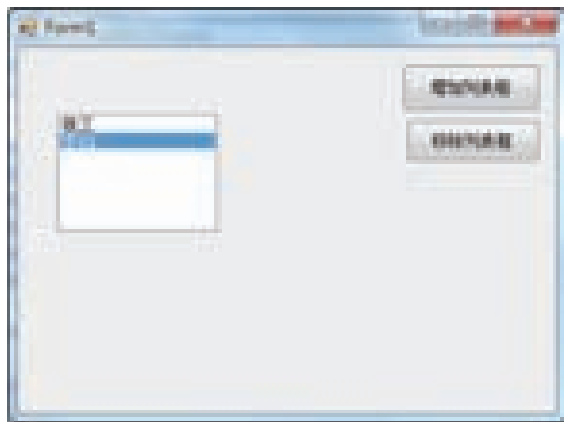


图3.28 运行效果

2. 创建和移除新控件的事件

用Forms.Controls.Add添加事件后，新控件没有任何事件过程，也无法使用属性窗口设置其事件。因此事件的增删也通过代码来实现。

修改button1的单击事件如下：

```
1         private void button1_Click(object sender, EventArgs e)
2         {
3             LB= new ListBox();
4             LB.Left = 30;
5             LB.Top = 50;
6             LB.Visible = true;
7             LB.Items.Add("张三");
8             LB.Items.Add("李四");
9             this.Controls.Add(LB);
10            LB.DoubleClick += new EventHandler(List_DblClick);
11        }
```

代码中最后一行为LB创建了一个双击事件List_DblClick，为此还需要在下面加上这个事件过程：

```
1         private void List_DblClick(object sender, EventArgs e)
2         {
3             MessageBox.Show(LB.Text);
4         }
```

当窗体启动后，单击button1创建列表框控件，当双击该列表框内容时，出现对话框。如果要用代码移除该事件，使用“LB.DoubleClick -= new EventHandler(List_DblClick);”即可。

3. 添加控件数组

有些时候，窗体上需要放置很多同样类型的控件，使用控件数组的方式最为便利。C#使用Form.Control.AddRange方法添加控件数组。下面的代码自动往窗体上添加了6个文本框控件。

```
1         private void button3_Click(object sender, EventArgs e)
2         {
3             TextBox[] tb=new TextBox[6];
4             for (int i = 0; i < 6; i++)
5             {
6                 tb[i] = new TextBox();
7                 tb[i].Text = "文本框" + i;
8                 tb[i].Left = 30;
9                 tb[i].Top =i*30;
10                tb[i].ForeColor =System.Drawing.Color.Red;
11            }
12            this.Controls.AddRange(tb);
13        }
```

代码中，第3行声明了一个文本框数组tb。接下来在for循环中依次设置每一个文本框的部分属性。第12行往窗体中添加tb这个控件数组。

启动窗体后，单击“增加控件数组”按钮，窗体左侧出现6个红色字体的文本框，运行效果如图3.29所示。

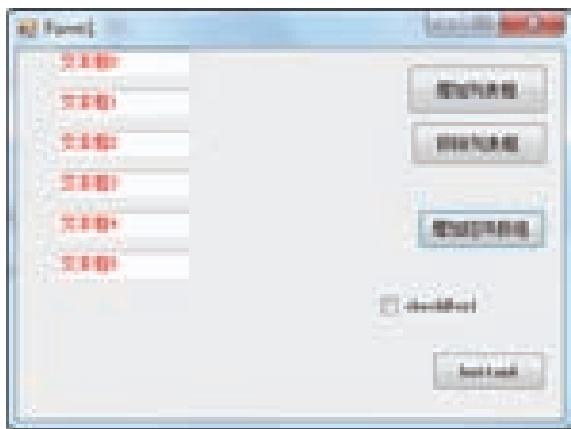


图3.29 增加控件数组

4. 移除控件

从控件工具箱中拖动一个CheckBox复选框控件到Form1中，再放置一个按钮button4，编辑button4的单击事件过程：

```
1      private void button4_Click(object sender, EventArgs e)
2      {
3          this.Controls.Remove(this.checkBox1);
4      }
```

移除一个控件只需要一行代码。当单击button4时，复选框控件从窗体上消失。

3.8 使用Windows API函数

API全称为Application Programming Interface，即应用程序接口。从意义上来说，API是一个操作系统或某个程序本身提供给其他程序使用的函数。在Windows操作系统中，有成千个Windows的函数提供给应用程序使用，本节所说的API就是指这些函数。

在实际的编程运用中，有时候仅仅靠程序中定义的函数和过程并不能完成任务，例如，在C#中如何自动操作计算器，如何自动关闭计算机上运行的一个程序或窗口，如何获得屏幕上某像素的颜色值，如何自动控制鼠标和键盘，等等。这些特殊的需求借助API函数可以实现。

系统中的API函数非常多，用途也十分广泛，本节仅仅举几个常见的例子，以便说明如何在C#中运行API函数。

■ 3.8.1 窗口类名和句柄

其实，计算机上的各种窗口都有各自的类名（ClassName）和句柄（Handle）。

1. 句柄

打个比方，当我们启动了计算机中的计算器（见图3.30），系统就会自动为计算器这个窗口分配一个编号，这个编号是不重复使用的，我们把这个唯一识别的编号叫窗口的句柄值。不难想到，屏幕上的每一个窗口都有各自的句柄值，事实上，窗口里的子窗口以及窗口中的控件也都有它们的句柄值。



图3.30 计算器

那么句柄有什么作用呢？其实API函数控制窗口的依据就是句柄，很多API函数的参数中包括句柄。当然我们平时办公的时候，无法知道一个窗口的句柄值是多少，可以借助一些现成的工具列举出窗口的各种信息。

2. 类名

窗口和控件不仅有句柄，而且还有类名。类名是指一类窗口，同种类型窗口的类名是相同的，例如，我们在屏幕上，同时打开了多个记事本文档，虽然每一个记事本文档窗口的句柄各不相同，但是它们的类名是不变的，永远是“Notepad”。那么类名有什么作用呢？实际编程过程中，可以通过类名去查找窗口的句柄，这个查找方法稍后讲。

下面举一个实例来说明C#中如何调用API函数。程序的意图是把屏幕上的计算器窗口的默认标题文字“计算器”，修改为C#窗体上文本框内的自定义文字。程序的具体步骤是，先用API查找到计算器窗口的句柄，然后使用API函数修改这个窗口的标题。这里要用到FindWindow和SetWindowText两个API函数。

API函数的查询工具有很多，图3.31是ApiViewer 2004的界面。

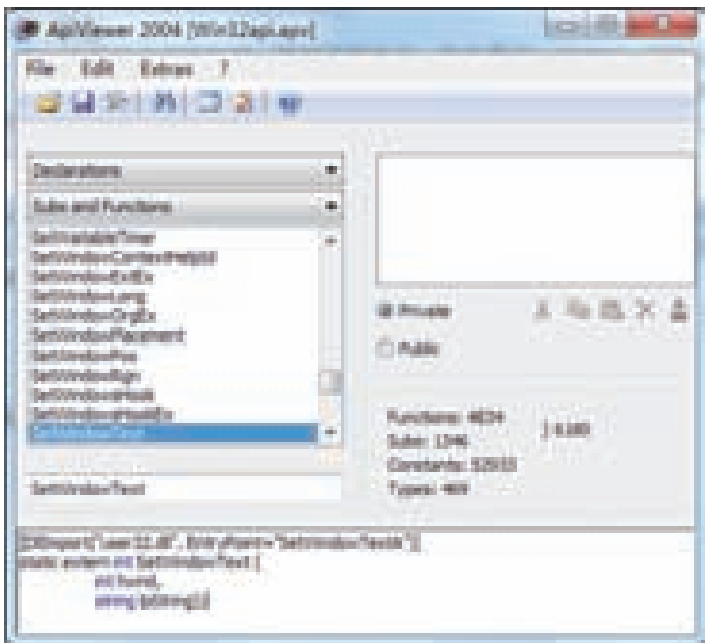


图3.31 ApiViewer 2004

第1步：创建一个名为“WindowsFormsApplication20160528”的窗体应用程序，在窗体上放置一个按钮和一个文本框控件。

第2步：双击按钮自动打开Form1.cs代码窗口，要使用API函数，必须在类模块顶部加入“using System.Runtime.InteropServices;”这条指令。

第3步：在类顶部声明两个API函数：

```
1 [DllImport("user32.dll", EntryPoint = "FindWindowA")]
2 static extern int FindWindow(string lpClassName, string lpWindowName);
3
4 [DllImport("user32.dll", EntryPoint = "SetWindowTextA")]
5 static extern int SetWindowText(int hwnd, string lpString);
```

第4步：编辑button1的单击事件代码为：

```
1 private void button1_Click(object sender, EventArgs e)
2 {
3     int h;
4     h = FindWindow("CalcFrame", null);
5     SetWindowText(h, this.textBox1.Text);
6 }
```

第5步：手动打开计算机中的计算器。

现在启动调试，在文本框中输入任意的文字，然后单击“更改标题文字”按钮，会看

到计算器窗口的标题文字发生改变。运行效果如图3.32所示。

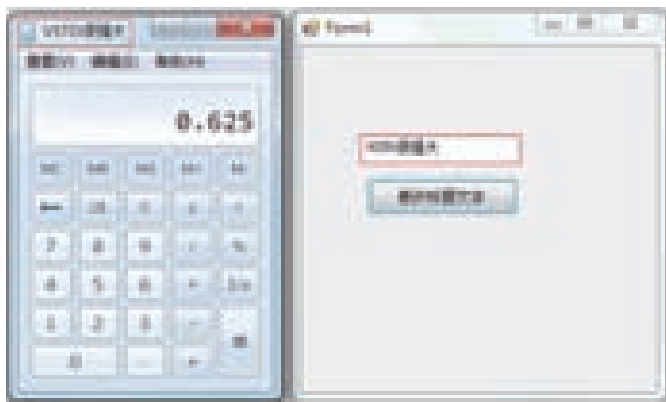


图3.32 更改窗口标题文字

Form1.cs类模块中的完整代码为：

```

1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10 using System.Runtime.InteropServices;
11 namespace WindowsFormsApplication20160528
12 {
13     public partial class Form1 : Form
14     {
15         [DllImport("user32.dll", EntryPoint = "FindWindowA")]
16         static extern int FindWindow(string lpClassName, string lpWindowName);
17
18         [DllImport("user32.dll", EntryPoint = "SetWindowTextA")]
19         static extern int SetWindowText(int hwnd, string lpString);
20         public Form1()
21         {
22             InitializeComponent();
23         }
24
25         private void button1_Click(object sender, EventArgs e)
26         {
27             int h;
28             h = FindWindow("CalcFrame", null);
29             SetWindowText(h, this.textBox1.Text);
30         }
31     }
32 }

```

代码中第10行是使用API的using指令。第15~19行是两个API函数的声明部分。

我们重点关注一下代码中第25~30行，整型变量h用来存储计算器窗口的句柄。

“h = FindWindow(“CalcFrame”, null);”这句代码很关键，FindWindow函数需要提供两个参数，一个是窗口的类名，另一个是窗口的标题文字，这是因为光靠句柄不能获得唯一的窗口，如果同时打开多个计算器窗口，就没法定位了。为此可以用标题文字加以区分。对于标题文字变化的窗口，这个参数设置为null即可。

第29行SetWindowText函数需要两个参数，一个是窗口的句柄，另一个是窗口新的标题文字。

与SetWindowText函数对应的还有一个GetWindowText函数，是用来获取一个窗口的标题文字的，读者自行尝试。

注意： CalcFrame是计算器的窗口类名。对于屏幕上打开的窗口如何获得它们的类名和句柄呢？可以借助一些Spy工具来获取，在这里笔者推荐使用Visual Studio自带的Spy++工具。

■ 3.8.2 使用Spy++

单击【开始/所有程序/Microsoft Visual Studio 2012/Visual Studio Tools/Spy++】，就会打开Spy++工具，如图3.33所示。

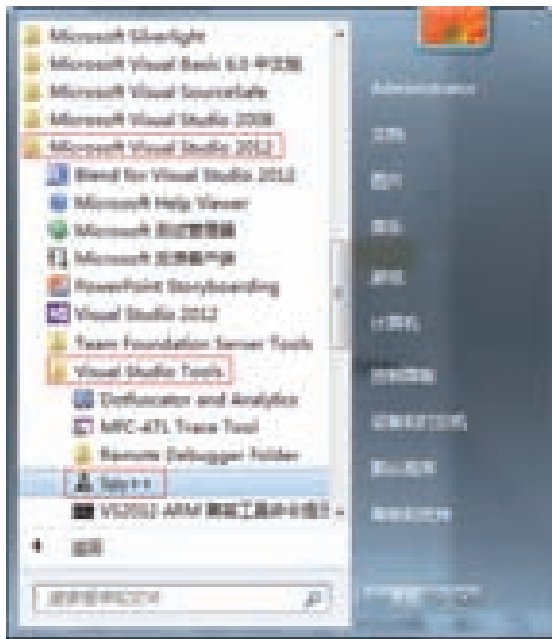


图3.33 启动Spy++

Spy++启动后，会列举出当前屏幕上所有打开的窗口信息，右击任一窗口条目，选择

右键菜单中的“属性”，出现详细的信息，如图3.34所示。

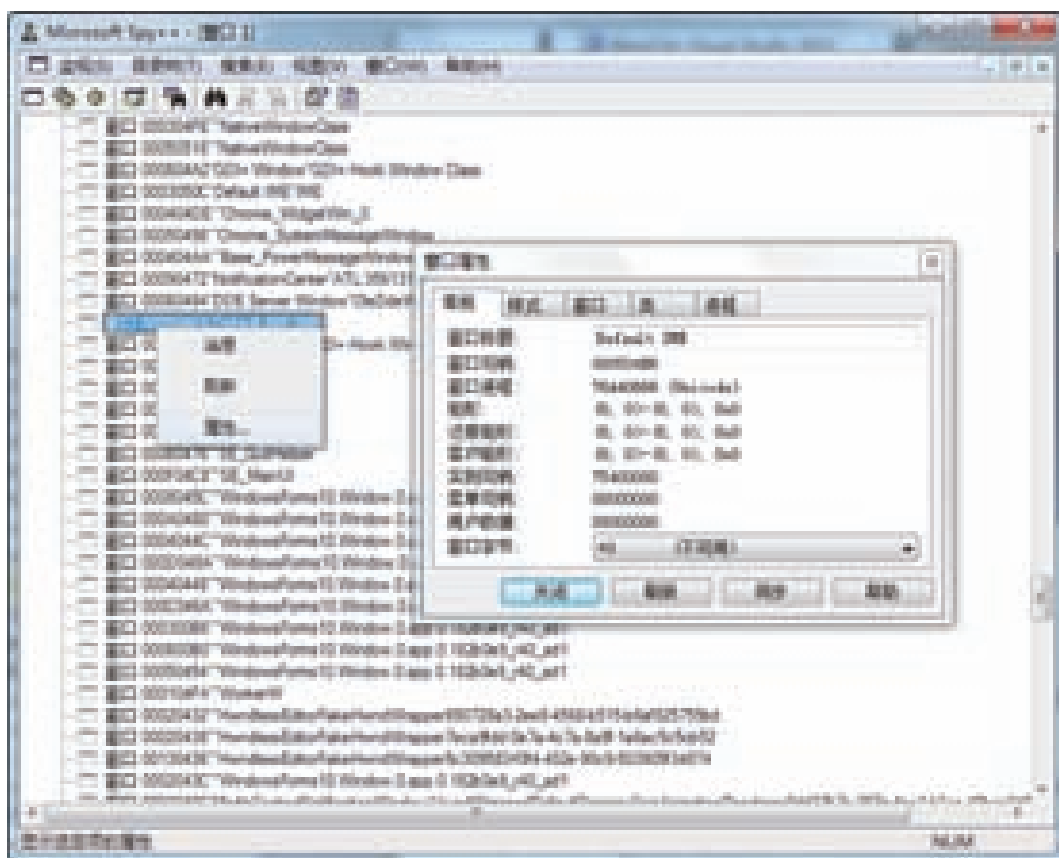


图3.34 在Spy++中查看窗口信息

Spy++还有一个重要功能是抓取窗口。单击Spy++的菜单【搜索/查找窗口】，拖动靶形按钮到目标窗口上松开鼠标，会抓取到该窗口的句柄等信息，如图3.35所示。

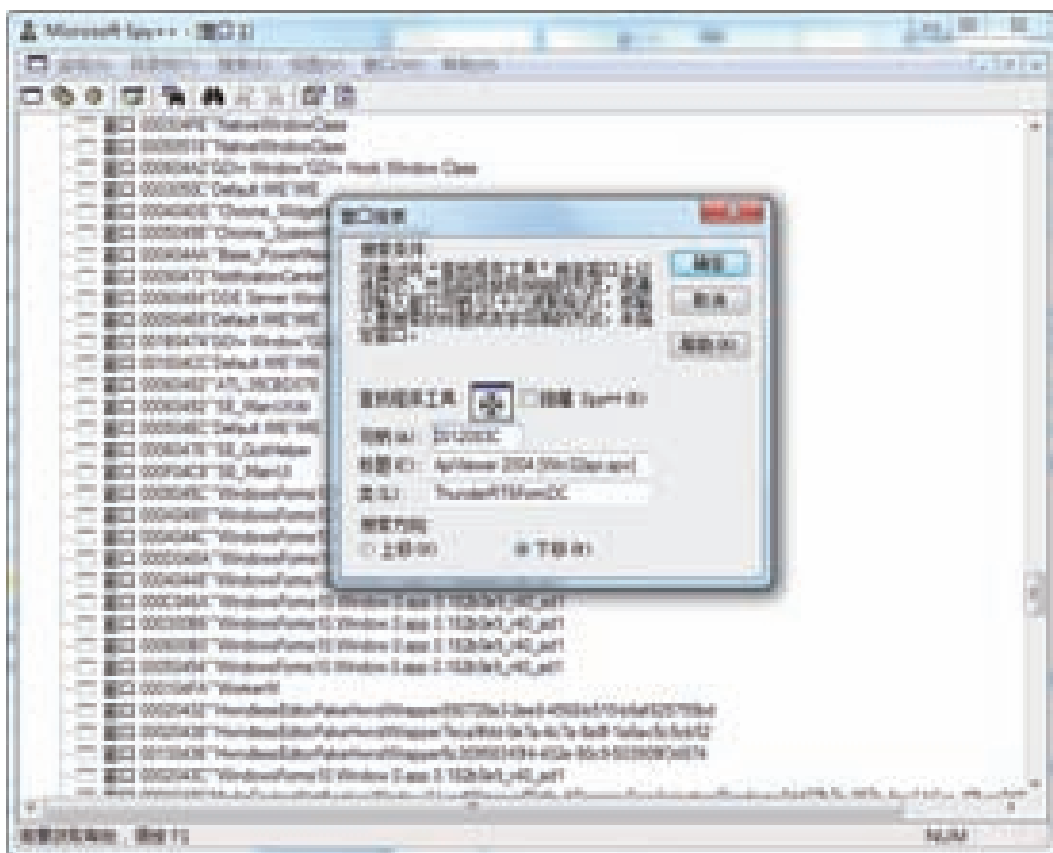


图3.35 搜索窗口

3.8.3 使用UseAPI

笔者开发了一个和Spy++功能很类似的UseAPI工具，如图3.36所示。

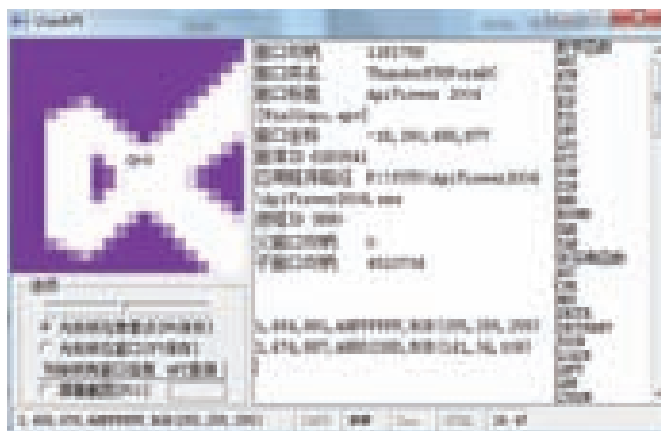


图3.36 UseAPI

该工具可以实时给出鼠标所在像素点的颜色值，还能抓取窗口信息。

■ 3.8.4 获取光标位置

API函数中有一个GetCursorPos函数，可以用来返回鼠标在屏幕上的位置（不是在窗体上的位置）。函数的声明方式为：

```
1      [DllImport("user32.dll")]
2      private static extern int GetCursorPos(out Point lpPoint);
```

设置窗体的MouseMove事件，编辑代码为：

```
1      private void Form1_MouseMove(object sender, MouseEventArgs e)
2      {
3          Point p = new Point();
4          GetCursorPos(out p);
5          this.textBox1.Text = (p.ToString());
6      }
```

当鼠标在窗体上移动时，文本框中显示当前光标在屏幕的坐标值。

本章要点回顾

- 本章讲述了C#中相对高级的实战技术，在实际的开发过程中，经常会用到这些技术，读者可以根据需要选择学习其中的知识点。
- 通过本章各节的学习，可以进一步熟悉C#程序的特点，有助于理解和掌握C#语法。
- 窗体与控件技术是界面开发的重要组成部分，理解掌握菜单和工具栏、状态栏和对话框的使用技术。

第4章

C#操作Excel对象

VSTO开发的最终目的就是使用C#语言操作和控制Office对象，为了实现对Office对象的全面控制，除了后面要讲到的制作Office外接程序和文档自定义项之外，还可以使用一般的Windows窗体应用程序来访问Office对象。因为Windows窗体应用程序项目是最简单的C#项目，而且调试过程也比Office外接程序容易。因此本章内容是后面制作更专业的VSTO项目的必不可少的预备知识。通过本章的学习，读者可以了解到Office对象模型，以及在C#中对Office对象属性的读写、事件过程等知识。

 本章视频：C#操作Excel对象.wmv

4.1 Excel对象模型概述

可以从微软提供的Office对象模型参考，或者从Office在线帮助系统获取Office组件的对象模型图。

Excel对象模型如图4.1所示。

Excel的主要对象有：

- Application：Excel应用程序对象，最顶层的对象。
- Workbook：工作簿对象。
- Worksheet：工作表对象。
- Range：单元格区域对象。
- Window：窗口对象。

还有下面一些常用对象：

- ActiveWorkbook：活动工作簿。
- ActiveSheet：活动工作表。

- ActiveWindow：活动窗口。
- Selection：Excel所选对象。
- Cells：工作表或区域的所有单元格。

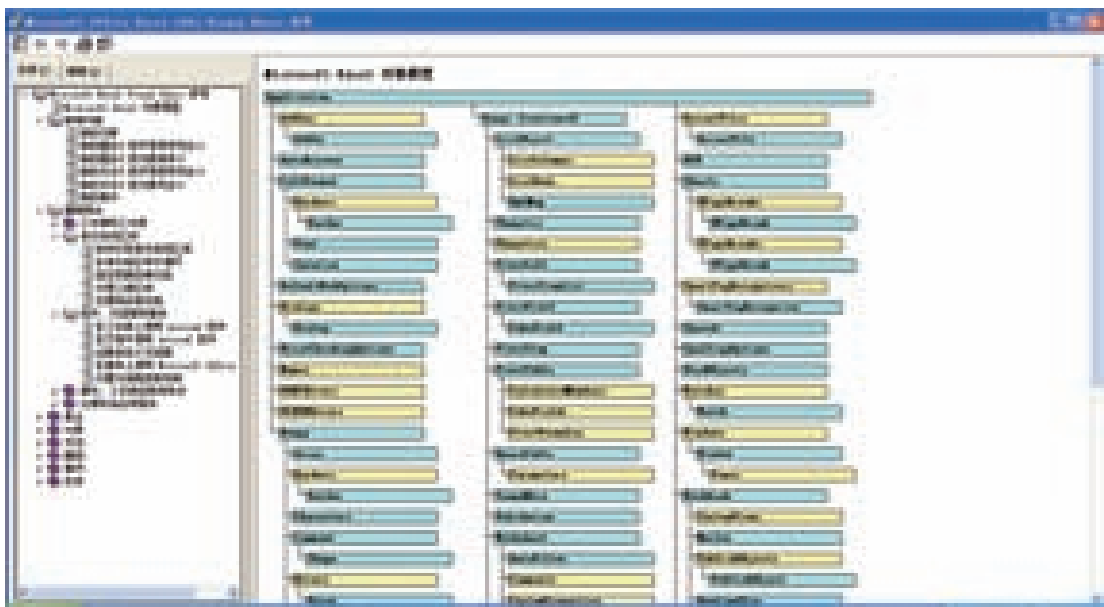


图4.1 Excel对象模型

几乎每一级对象都有其自身的子成员、属性、方法和事件。例如Worksheet对象，它有Name、Type属性，有Rows、Columns、Cells这些子成员，还有Calculate、Activate等方法，还有SelectionChange、BeforeDoubleClick等事件。我们没必要掌握每一个小的细节，但是需要了解最常用的一些代码技巧。

在开发过程中，难免会遇到自己比较陌生的对象，例如要在C#程序中操作Excel单元格的批注对象，没有现成的代码可以参考，该怎么办呢？可以使用VBA中的录制宏功能，之后把录制好的VBA代码适当修改并转换为C#代码即可。因此学习VSTO开发不能完全脱离VBA。

■ 4.1.1 Application对象

Application对象处于Excel对象的最顶端，它代表整个Excel应用程序。利用Application对象的属性可以方便地控制Excel应用程序的工作环境。例如，可以设置Excel的标题，设置是否显示状态栏、编辑栏和滚动条等。

Application对象有众多的成员、属性、方法和事件。如果要了解这些对象的详细信息，需要在Excel帮助系统中搜索。

在Excel 2010中，按下快捷键【Alt+F11】打开VBA编辑器，然后按下【F1】键或者单击菜单【帮助/Microsoft Visual Basic for Applications帮助】会打开Excel帮助界面，如图4.2所示。

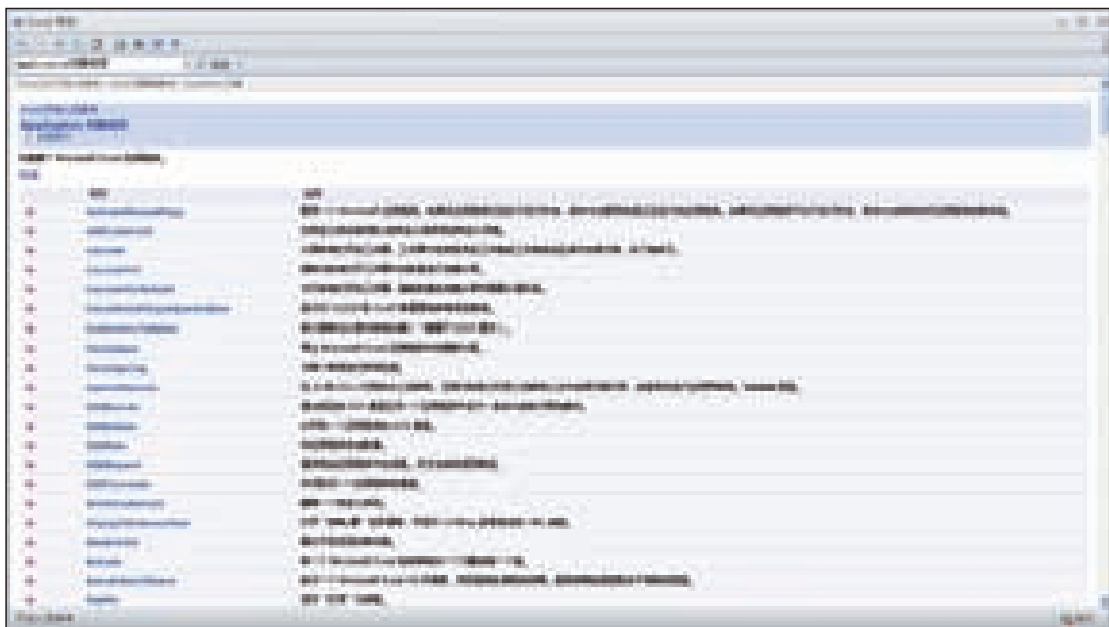


图4.2 Excel帮助界面

在搜索框中输入关键字“Application对象成员”，会看到Excel的Application对象的所有成员、方法和事件列表。当然可以继续单击任一条目，跳转到该条目的更详细的内容页面中。

■ 4.1.2 Workbook对象

工作簿集合对象Workbooks是Application的一个成员对象，这个集合对象表示应用程序所有打开的工作簿（不包含加载宏文件）。

Workbook对象则是指某一特定的工作簿，例如Workbooks[2]表示第二个工作簿。所有和工作簿有关的操作，例如打开、关闭、保存都是Workbook的方法。

■ 4.1.3 Worksheet对象

Excel的Workbook对象之下有Worksheets集合对象以及Sheets集合对象，这两个对象的区别在于，Worksheets只包括工作簿下的所有一般工作表，而Sheets则表示所有的表对象，例如Excel宏表、图表工作表等，都是Sheets的成员。

那么Worksheets的个体对象就是Worksheet，使用Worksheet对象可以对工作表进行显

示、隐藏、增加、移动和删除等操作。

■ 4.1.4 Range对象

Range对象是指单元格区域，它是Worksheet之下的一个成员对象。单元格的填充和清空、行列的插入和删除等操作都是Range对象范畴的操作。

Range对象使用单元格地址作为参数，来确定具体操作的区域。例如ActiveSheet.Range["A2:D6"]，就指代“A2:D6”这个单元格区域；Range["2:4"]表示第二到第四行。

■ 4.1.5 Window对象

我们知道，在Excel中可以创建多个不同的窗口来显示同一个工作簿，那么这些对象就是Windows对象，它是Workbook之下的一个集合对象。其中的任一窗口都是Window对象。通过改变Window对象的属性，可以设置是否显示网格线，是否显示标尺，是否显示行号、列标等。

Application之下有一个ActiveWindow对象，是指当前活动窗口。这个对象也是Window对象，在开发过程中会经常用到。

4.2 创建可以访问Excel对象的C#窗体应用程序

一般情况下，创建的C#窗体应用程序是一个独立的程序，和Excel没有任何关系。为操作和控制Excel对象，需要为项目添加关于Excel和Office的引用。如果不添加这些引用，在C#代码中当然不会出现Excel的对象成员。

启动Visual Studio 2012，创建一个名为“WindowsFormsApplication20160522”的C#窗体应用程序项目。

在Form1的设计视图放入一个Button控件。

■ 4.2.1 添加Excel 2010对象引用

单击菜单【项目/添加引用】，在“引用管理器”对话框中单击【程序集/扩展】，找到“Microsoft.Office.Interop.Excel”，由于是要操作Excel 2010，所以勾选版本号为“14.0.0.0”的那一项，单击“确定”按钮，如图4.3所示。

■ 4.2.2 添加Office 2010对象引用

由于程序中可能会用到Office对象，例如Commandbar对象、FileDialog对象，为此还需

单击“确定”按钮后，在项目的资源管理器的引用中可以看到刚才添加的两项引用，如图4.5所示。

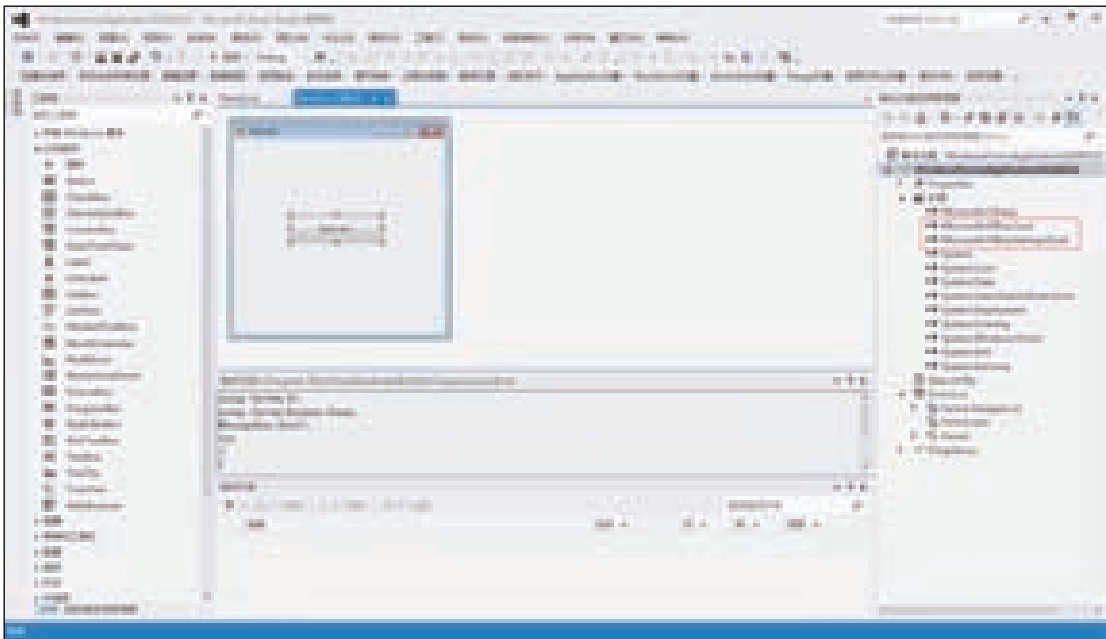


图4.5 添加引用后的效果

注意 后面讲到的VSTO创建Office外接程序项目中，这两项引用在创建项目的时候会自动添加。

4.3 操作Application对象

4.3.1 获取正在运行的Excel对象

在Form1的设计视图中，双击button1进入Form1.cs的代码编辑区：

```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Threading.Tasks;
9 using System.Windows.Forms;
10 using System.Runtime.InteropServices;
11 using Excel = Microsoft.Office.Interop.Excel;
```

```

12 using Office = Microsoft.Office.Core;
13 namespace WindowsFormsApplication20160522
14 {
15     public partial class Form1 : Form
16     {
17         Excel.Application ExcelApp;
18         public Form1()
19         {
20             InitializeComponent();
21         }
22
23         private void button1_Click(object sender, EventArgs e)
24         {
25             获取Excel对象();
26         }
27         public void 获取Excel对象()
28         {
29             //获取正在运行的Excel
30             ExcelApp = (Excel.Application)Marshal.GetActiveObject("Excel.
Application");
31             MessageBox.Show(ExcelApp.Version + "");
32         }
33     }
34 }

```

代码中第10~12行using语句是笔者加入的。

第17行声明了Excel的应用程序对象ExcelApp，button1的Click事件中调用了子过程“获取Excel对象”。

重点关注一下第30行Marshal.GetActiveObject(“Excel.Application”)，这一句是核心代码，用来获取打开了的Excel应用程序，ExcelApp从此实例化了。

对于调试本章中其他的示例过程，读者均可按照此方法，在button1的单击事件过程中调用各个过程。

```

1     public void 获取Excel对象()
2     {
3         //获取正在运行的Excel
4         Excel.Application ExcelApp = (Excel.Application)Marshal.
GetActiveObject("Excel.Application");
5         MessageBox.Show(ExcelApp.Version + "");
6     }

```

与此对应的VBA代码如下：

```

1 Public Sub 获取Excel对象()
2     Set ExcelApp = GetObject(, "Excel.Application")
3     ExcelApp.Visible = True
4     MsgBox ExcelApp.Version
5 End Sub

```

■ 4.3.2 创建新的Excel对象

如果计算机上没有打开的Excel应用程序对象，则可以使用下面的过程自动创建Excel对象。

```
1      public void 创建Excel对象()
2      {
3          Excel.Application NewApp = new Excel.Application();
4          NewApp.Visible = true;
5          NewApp.Caption = "new app";
6      }
```

与此对应的VBA代码如下：

```
1 Public Sub 创建Excel对象()
2     Set ExcelApp = CreateObject("Excel.Application")
3     ExcelApp.Visible = True
4     ExcelApp.Caption = "New APP"
5 End Sub
```

■ 4.3.3 Application对象常用属性

```
1      public void Application对象常用属性读写()
2      {
3          ((Excel.Worksheet)ExcelApp.ActiveSheet).Name = "MySheet";//重命名活
动工作表
4          Excel.Range rng = (Excel.Range)ExcelApp.Selection;//Excel所选对象赋
值给对象变量rng
5          ExcelApp.StatusBar = "My Excel Application"; //改变Excel状态栏文字
6          MessageBox.Show(ExcelApp.Workbooks.Count + "");//显示工作簿个数
7          MessageBox.Show(ExcelApp.UserName + "");//显示用户名
8          ExcelApp.DisplayFormulaBar = false;//隐藏公式编辑栏
9      }
```

与此对应的VBA代码如下：

```
1 Public Sub Application对象常用属性读写()
2     ExcelApp.ActiveSheet.Name = "MySheet"
3     Dim rng As Excel.Range
4     Set rng = ExcelApp.Selection
5     ExcelApp.StatusBar = "My Excel Application" '改变Excel状态栏文字
6     MsgBox (ExcelApp.Workbooks.Count) '显示工作簿个数
7     MsgBox (ExcelApp.UserName) '显示用户名
8     ExcelApp.DisplayFormulaBar = False '隐藏公式编辑栏
9 End Sub
```

■ 4.3.4 Application对象常用方法

```

1      public void Application对象常用方法()
2      {
3          ExcelApp.Undo();//Excel撤销,相当于按下了Ctrl+Z
4          ExcelApp.Workbooks.Close();//关闭所有工作簿
5          ExcelApp.Quit();//退出Excel
6      }

```

与此对应的VBA代码如下:

```

1  Public Sub Application对象常用方法()
2      ExcelApp.Undo 'Excel撤销,相当于按下了Ctrl+Z
3      ExcelApp.Workbooks.Close '关闭所有工作簿
4      ExcelApp.Quit '退出Excel
5  End Sub

```

■ 4.3.5 Application对象常用事件

```

1      public void Application对象常用事件()
2      {
3          ExcelApp.WorkbookBeforeClose += new Excel.AppEvents_WorkbookBeforeCloseEventHandler(ExcelApp_WorkbookBeforeClose);//创建Excel应用程序级事件过程
4          ExcelApp.SheetSelectionChange += new Excel.AppEvents_SheetSelectionChangeEventHandler(ExcelApp_SheetSelectionChange);
5      }
6      public void ExcelApp_WorkbookBeforeClose(Excel.Workbook wbk, ref bool Cancel)
7      {
8          MessageBox.Show("即将关闭:" + wbk.FullName);
9          Cancel = true;//取消关闭
10     }
11     public void ExcelApp_SheetSelectionChange(object sh, Excel.Range Target)
12     {
13         Target.Interior.Color = System.Drawing.Color.Green;//改变鼠标所选区域底纹颜色
14     }

```

代码中为ExcelApp创建了SheetSelectionChange和WorkbookBeforeClose事件,前者是工作表的单元格所选区域发生变化时引发,后者是工作簿关闭之前引发。

事件创建后,当鼠标在工作表中选择其他单元格区域时,所选区域背景色会变绿。

与此对应的VBA代码如下:

```

1  Public Sub Application对象常用事件()
2      '事件过程的定义在类模块中
3      Set Example.ExcelApp_Events = ExcelApp '赋予事件

```

```

4 End Sub
5
6 '下面是定义在类模块中的事件过程:
7 Public WithEvents ExcelApp_Events As Excel.Application
8 Private Sub ExcelApp_Events_SheetSelectionChange(ByVal Sh As Object, ByVal
Target As Excel.Range)
9     Target.Interior.Color = vbRed '改变鼠标所选区域底纹颜色
10 End Sub
11
12 Private Sub ExcelApp_Events_WorkbookBeforeClose(ByVal wb As Excel.Workbook,
Cancel As Boolean)
13     MsgBox ("即将关闭:" + wb.FullName)
14     Cancel = True '取消关闭
15 End Sub

```

■ 4.3.6 Application重要集合对象

```

1     public void Application重要集合对象()
2     {
3         MessageBox.Show(ExcelApp.Dialogs.Count + ""); //Excel所有内置对话框个数
4         foreach (Excel.AddIn adn in ExcelApp.AddIns)
5         {
6             result += (adn.Name + "\t" + adn.Installed + "\n");
7             //遍历所有加载项信息
8         }
9         MessageBox.Show(ExcelApp.CommandBars.Count + "");
10        foreach (Office.CommandBar cmb in ExcelApp.CommandBars)
11        {
12            result += (cmb.Name + "\t" + cmb.Type + "\n");
13            //遍历所有工具栏信息
14        }
15    }

```

代码中使用了foreach结构遍历对象中的成员，并把每一个成员的属性追加到result这个字符串变量中。

与此对应的VBA代码如下：

```

1 Public Sub Application重要集合对象()
2     Dim adn As Excel.AddIn
3     MsgBox (ExcelApp.Dialogs.Count) 'Excel所有内置对话框个数
4     result = ""
5     For Each adn In ExcelApp.AddIns
6         result = result & (adn.Name & vbTab & adn.Installed & vbNewLine)
7         '遍历所有加载项信息
8     Next adn
9 End Sub

```


4.4 操作Workbook对象

4.4.1 Workbook对象常用属性

本节我们学习一下C#中使用名称以及数字序号引用成员的语法：

```

1      public void Workbook对象常用属性读写()
2      {
3          Excel.Workbook wbk;
4          wbk = ExcelApp.ActiveWorkbook; //wbk为活动工作簿
5          wbk = ExcelApp.Workbooks[1];      //wbk为第1个工作簿
6          wbk=ExcelApp.Workbooks["Hello.xls"]; //使用名称确定工作簿
7
8
9          result = wbk.Worksheets.Count + ""; //给出工作簿中一般工作表的个数
10         result = wbk.Sheets.Count + ""; //给出工作簿中所有类型的工作表个数
11
12         //遍历打开的工作簿
13         foreach (Excel.Workbook wb in ExcelApp.Workbooks)
14         {
15             result += (wb.Name + "\n");
16         }
17         //常用属性:
18         bool sv = wbk.Saved; //工作簿是否已保存
19         string path = wbk.Path; //工作簿文件路径
20         bool pwd = wbk.HasPassword; //是否有密码保护
21
22     }

```

注意代码中第5行和第6行，分别引用了第1个工作簿和名为“Hello.xls”的工作簿，和下面对应的VBA代码相比，可以看出C#使用中括号把名称或序号括起来，而VBA中使用的是圆括号。

与此对应的VBA代码如下：

```

1  Public Sub Workbook对象常用属性读写()
2      Dim wbk As Excel.Workbook
3      Set wbk = ExcelApp.ActiveWorkbook 'wbk为活动工作簿
4      Set wbk = ExcelApp.Workbooks(1) 'wbk为第1个工作簿
5      Set wbk = ExcelApp.Workbooks("Hello.xls") '使用名称确定工作簿
6      result = wbk.Worksheets.Count '给出工作簿中一般工作表的个数
7      result = wbk.Sheets.Count '给出工作簿中所有类型的工作表的个数
8      '遍历打开的工作簿
9      Dim wb As Excel.Workbook
10     For Each wb In ExcelApp.Workbooks
11         result = result & (wb.Name + vbNewLine)
12     Next wb
13     '常用属性:

```

```

14 Dim sv As Boolean, path As String, pwd As Boolean
15 sv = wbk.Saved '工作簿是否已保存
16 path = wbk.path '工作簿文件路径
17 pwd = wbk.HasPassword '是否有密码保护
18 End Sub

```

■ 4.4.2 Workbook对象常用方法

```

1 public void Workbook对象常用方法()
2 {
3     Excel.Workbook wbk;
4     //打开已存在的工作簿文件
5     wbk = ExcelApp.Workbooks.Open(@"C:\Documents and Settings\
Administrator\桌面\long2.xlsm");
6     result = wbk.FileFormat.ToString();
7
8     //新建工作簿
9     wbk = ExcelApp.Workbooks.Add();
10
11    //保存工作簿
12    wbk.SaveAs(@"C:\Documents and Settings\Administrator\桌面\我的工作
簿.xlsx");
13
14    //关闭工作簿
15    wbk.Close(false, Type.Missing, Type.Missing);
16 }

```

与此对应的VBA代码如下：

```

1 Public Sub Workbook对象常用方法()
2     Dim wbk As Excel.Workbook
3     '打开已存在的工作簿文件
4     Set wbk = ExcelApp.Workbooks.Open("C:\Documents and Settings\
Administrator\桌面\long2.xlsm")
5     result = wbk.FileFormat
6
7     '新建工作簿
8     Set wbk = ExcelApp.Workbooks.Add()
9
10    '保存工作簿
11    wbk.SaveAs ("C:\Documents and Settings\Administrator\桌面\我的工作簿.xlsx")
12
13    '关闭工作簿
14    wbk.Close False
15 End Sub

```

4.4.3 Workbook对象常用事件

```

1      public void Workbook对象常用事件()
2      {
3          Excel.Workbook wbk;
4          wbk = ExcelApp.ActiveWorkbook;
5          wbk.BeforeSave += new Excel.WorkbookEvents_
BeforeSaveEventHandler(wbk_BeforeSave);
6      }
7      public void wbk_BeforeSave(bool UI, ref bool Cancel)
8      {
9          MessageBox.Show("取消保存!");
10         Cancel = true;//取消保存操作
11     }

```

上述代码为ActiveWorkbook创建了一个BeforeSave事件，该事件在当前工作簿保存之前发生。当保存工作簿的时候，会调用wbk_BeforeSave这个过程，跳出对话框并显示“取消保存！”。

与此对应的VBA代码如下：

```

1 Public Sub Workbook对象常用事件()
2     '事件过程的定义在类模块中
3     Set Example.Workbook_Events = ExcelApp.ActiveWorkbook
4 End Sub
5
6 '以下代码定义在类模块中
7 Public WithEvents Workbook_Events As Excel.Workbook
8 Private Sub Workbook_Events_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As
Boolean)
9     MsgBox "取消保存!"
10    Cancel = True '取消保存操作
11 End Sub

```

4.4.4 Workbook重要集合对象

```

1      public void Workbook重要集合对象()
2      {
3          Excel.Workbook wbk = ExcelApp.ActiveWorkbook;
4          //遍历所有工作表
5          foreach (Excel.Worksheet wst in wbk.Worksheets)
6          {
7              result += (wst.Name + "\n");
8          }
9          //      foreach (Office.DocumentProperty dp in wbk.
CustomDocumentProperties)
10         //      {
11         //          result += (dp.Name + "\n");

```

```

12         //    }
13         foreach (Excel.Window w in wbk.Windows)
14         {
15             w.Caption = "Change Caption";
16             //改变工作簿各窗口的标题文字
17         }
18     }

```

与此对应的VBA代码如下：

```

1 Public Sub Workbook重要集合对象()
2     Dim wbk As Excel.Workbook
3     Dim wst As Worksheet
4     Dim w As Excel.Window
5     Set wbk = ExcelApp.ActiveWorkbook
6     '遍历所有工作表
7     For Each wst In wbk.Worksheets
8         result = result & wst.Name & vbNewLine
9     Next wst
10    For Each w In wbk.Windows
11        w.Caption = "Change Caption"
12        '改变工作簿各窗口的标题文字
13    Next w
14 End Sub

```

4.5 操作Worksheet对象

4.5.1 Worksheet对象常用属性

```

1     public void Worksheet对象常用属性读写()
2     {
3         Excel.Worksheet wst;
4         wst = (Excel.Worksheet)ExcelApp.ActiveSheet;
5         ///wst.Name="NewSheetName"//更改工作表名称
6         wst = (Excel.Worksheet)ExcelApp.ActiveWorkbook.Worksheets[1];//通
        过数字下标引用
7         wst = (Excel.Worksheet)ExcelApp.ActiveWorkbook.
        Worksheets["Sheet3"];//通过名称引用
8         MessageBox.Show(wst.Shapes.Count + "");//给出工作表中图形个数
9         wst.UsedRange.Select();//选择使用区域
10        long r = wst.Rows.Count;
11        long c = wst.Columns.Count;
12        MessageBox.Show(r + "/" + c + "");//给出工作表的行数和列数
13        wst.Visible = Excel.XlSheetVisibility.xlSheetHidden;//隐藏当前工作表
14    }

```

与此对应的VBA代码如下：

```

1 Public Sub Worksheet对象常用属性读写()
2     Dim wst As Excel.Worksheet
3     Set wst = ExcelApp.ActiveSheet
4     wst.Name = "NewSheetName" '更改工作表名称
5     Set wst = ExcelApp.ActiveWorkbook.Worksheets(1) '通过数字下标引用
6     Set wst = ExcelApp.ActiveWorkbook.Worksheets("Sheet3") '通过名称引用
7     MsgBox wst.Shapes.Count '给出工作表中图形个数
8     wst.UsedRange.Select '选择使用区域
9     Dim r As Long, c As Long
10    r = wst.Rows.Count
11    c = wst.Columns.Count
12    MsgBox (r & "/" & c) '给出工作表的行数和列数
13    wst.Visible = Excel.XlSheetVisibility.xlSheetHidden '隐藏当前工作表
14 End Sub

```

■ 4.5.2 Worksheet对象常用方法

```

1     public void Worksheet对象常用方法()
2     {
3         Excel.Worksheet wst;
4         wst = (Excel.Worksheet)ExcelApp.ActiveSheet;
5         //wst=(Excel.Worksheet)ExcelApp.ActiveWorkbook.Worksheets.
Add();//增加工作表
6         ///wst.Delete();//删除工作表
7         wst.Select();//选择工作表
8         ///wst.Activate();//激活工作表
9
10        //遍历工作表
11        foreach (Excel.Worksheet w in ExcelApp.ActiveWorkbook.Worksheets)
12        {
13            result += (w.Index + "\t" + w.Name + "\n");
14        }
15    }

```

与此对应的VBA代码如下：

```

1 Public Sub Worksheet对象常用方法()
2     Dim wst As Excel.Worksheet
3     Set wst = ExcelApp.ActiveSheet
4     Set wst = ExcelApp.ActiveWorkbook.Worksheets.Add '增加工作表
5     wst.Select '选择工作表
6     wst.Activate '激活工作表
7     wst.Delete '删除工作表
8     '遍历工作表
9     Dim w As Excel.Worksheet
10    For Each w In ExcelApp.ActiveWorkbook.Worksheets
11        result = result & w.Index & vbTab & w.Name & vbNewLine
12    Next w
13 End Sub

```

■ 4.5.3 Worksheet对象常用事件

```

1      public void Worksheet对象常用事件()
2      {
3          Excel.Worksheet wst2;
4          wst2 = (Excel.Worksheet)ExcelApp.ActiveSheet;
5          wst2.SelectionChange += new Excel.DocEvents_SelectionChangeEventH
andler(wst2_SelectionChange);
6          Excel.Worksheet wst3;
7          wst3 = (Excel.Worksheet)ExcelApp.ActiveWorkbook.Worksheets[3];
8          ///wst3.Activate();
9          ///wst3.BeforeDoubleClick+=new Excel.DocEvents_BeforeDoubleClickE
ventHandler(wst3_BeforeDoubleClick);
10         }
11         public void wst2_SelectionChange(Excel.Range Target)
12         {
13             Target.Interior.Color = System.Drawing.Color.Yellow;
14         }
15         public void wst3_BeforeDoubleClick(Excel.Range Target, bool Cancel)
16         {
17             MessageBox.Show("双击了工作表");
18         }

```

与此对应的VBA代码如下：

```

1 Public Sub Worksheet对象常用事件()
2     '事件过程的定义在类模块中
3     Set Example.Worksheet_Events = ExcelApp.ActiveSheet
4 End Sub
5 '以下代码定义在类模块中
6 Public WithEvents Worksheet_Events As Excel.Worksheet
7 Private Sub Worksheet_Events_SelectionChange(ByVal Target As Excel.Range)
8     Target.Interior.Color = vbYellow
9 End Sub

```

4.6 操作Range对象

■ 4.6.1 Range对象常用属性

```

1      public void Range对象常用属性读写()
2      {
3          Excel.Worksheet wst;
4          Excel.Range rg;
5          wst = (Excel.Worksheet)ExcelApp.ActiveSheet;
6          rg = wst.Range["A" + 1];
7          rg.Formula = "=Sum(4,5,3)";

```

```

8
9         rg = wst.Range("B2:D5");
10        result = rg.get_Address(false, false); //返回地址
11        rg.ClearContents();
12
13        rg = (Excel.Range)wst.Cells[4, 5];
14
15        rg = wst.Range("C23:H30");
16        long r = rg.Row;
17        long c = rg.Column;
18        MessageBox.Show(r + "/" + c); //返回左上角单元格行列数
19    }

```

代码中第5行使用了强制转换，把ActiveSheet转换为Excel.Worksheet，这是因为ActiveSheet泛指工作簿中所有Sheet，而不是所有Worksheet。

代码中第7行往A1单元格输入一个公式。

与此对应的VBA代码如下：

```

1 Public Sub Range对象常用属性读写()
2     Dim wst As Excel.Worksheet
3     Dim rg As Excel.Range
4     Set wst = ExcelApp.ActiveSheet
5     Set rg = wst.Range("A" & 1)
6     rg.Formula = "=Sum(4,5,3)"
7     Set rg = wst.Range("B2:D5")
8     result = rg.Address(False, False) '返回地址
9     rg.ClearContents
10    Set rg = wst.Cells(4, 5)
11    Set rg = wst.Range("C23:H30")
12    Dim r As Long, c As Long
13    r = rg.Row
14    c = rg.Column
15    MsgBox r & "/" & c '返回左上角单元格行列数
16 End Sub

```

■ 4.6.2 Range对象常用方法

```

1     public void Range对象常用方法()
2     {
3         Excel.Worksheet wst;
4         Excel.Range rg;
5         wst = (Excel.Worksheet)ExcelApp.ActiveSheet;
6         rg = wst.Range["B:D"];
7         rg.Select();
8         rg.Merge(); //合并单元格
9         rg = wst.Range["B2"];
10        Excel.Range rg2 = rg.get_Resize(2, 2); //改变Range的尺寸
11        rg2.Select();
12    }

```

与此对应的VBA代码如下：

```
1 Public Sub Range对象常用方法()  
2     Dim wst As Excel.Worksheet  
3     Dim rg As Excel.Range  
4     Set wst = ExcelApp.ActiveSheet  
5     Set rg = wst.Range("B:D")  
6     rg.Select  
7     rg.Merge '合并单元格  
8     Set rg = wst.Range("B2")  
9     Dim rg2 As Excel.Range  
10    Set rg2 = rg.Resize(2, 2) '改变Range的尺寸  
11    rg2.Select  
12 End Sub
```

■ 4.6.3 Range对象的遍历

```
1     public void Range对象的遍历()  
2     {  
3         Excel.Worksheet wst;  
4         Excel.Range rg;  
5         wst = (Excel.Worksheet)ExcelApp.ActiveSheet;  
6         rg = wst.Range["B2:D10"];  
7         int i = 0;  
8         foreach (Excel.Range rg2 in rg.Cells)  
9         {  
10            i++;  
11            rg2.Value = i;  
12        }  
13    }
```

Excel中的Range对象是单元格区域对象，通常指的是一个矩形区域，但是也可以是不连续的区域，几列、几行或者一个单元格。

编程过程中，经常需要遍历单元格区域中每一个单元格的内容，可以使用上述的foreach结构来遍历。

与此对应的VBA代码如下：

```
1 Public Sub Range对象的遍历()  
2     Dim wst As Excel.Worksheet  
3     Dim rg As Excel.Range  
4     Set wst = ExcelApp.ActiveSheet  
5     Set rg = wst.Range("B2:D10")  
6     Dim i As Integer  
7     i = 0  
8     Dim rg2 As Excel.Range  
9     For Each rg2 In rg.Cells  
10        i = i + 1
```



```

11         rg2.Value = i
12     Next rg2
13 End Sub

```

■ 4.6.4 二维数组与Range数据交换

编程过程中，经常会用到内存数组与Excel单元格区域数据的交换，当然最笨的方法就是使用循环，逐个交换。但是也可以使用批量的方法，一句代码就完成交换。

```

1     public void 二维数组与Range数据交换()
2     {
3         int[,] arr = new int[3, 4];
4         for (int i = 0; i < 3; i++)
5         {
6             for (int j = 0; j < 4; j++)
7             {
8                 arr[i, j] = i * 10 + j;
9             }
10        }
11        Excel.Range rg = ExcelApp.Range["B2:E4"];
12        rg.Select();
13        rg.Value2 = arr; //数组写入单元格区域
14        object[,] arr2;
15        arr2 = (object[,])ExcelApp.Range["B2:E2"].Value2; //行向区域写入数组
16        arr2 = (object[,])ExcelApp.Range["B2:B4"].Value2; //列向区域写入数组
17        arr2 = (object[,])ExcelApp.Range["C3:E4"].Value2; //矩形区域写入数组
18    }

```

上述代码声明了一个二维数组arr，并且使用两层for循环填充元素的值。接着使用rg变量来代表单元格区域“B2:E4”，第13行代码一次性把数组的值赋给了单元格区域。

代码中第14~17行，使用新的数组arr2来批量获取单元格区域的值。

与此对应的VBA代码如下：

```

1 Public Sub 二维数组与Range数据交换()
2     Dim arr(0 To 2, 0 To 3) As Integer
3     Dim i As Integer, j As Integer
4     For i = 0 To 2
5         For j = 0 To 3
6             arr(i, j) = i * 10 + j
7         Next j
8     Next i
9     Dim rg As Excel.Range
10    Set rg = ExcelApp.Range("B2:E4")
11    rg.Select
12    rg.Value = arr '数组写入单元格区域
13
14    Dim arr2 As Variant
15    arr2 = ExcelApp.Range("B2:E2").Value '行向区域写入数组

```

```

16     arr2 = ExcelApp.Range("B2:B4").Value '列向区域写入数组
17     arr2 = ExcelApp.Range("C3:E4").Value '矩形区域写入数组
18 End Sub

```

■ 4.6.5 一维数组与Range数据交换

```

1     public void 一维数组与Range数据交换()
2     {
3         int[] arr = new int[4] { 3, 5, 7, 9 };
4         Excel.Range rg = ExcelApp.Range["A1:D1"];
5         rg.Value2 = arr; //行向区域接受一维数组的数据
6
7         int[,] arr2 = new int[4, 1];
8         for (int i = 0; i < 4; i++)
9         {
10             arr2[i, 0] = arr[i];
11         }
12         Excel.Range rg2 = ExcelApp.Range["A1:A4"];
13         rg2.Value2 = arr2; //先将一维数组数据传递给二维数组;列向区域接受二维数组
    数据
14
15         object[,] arr3 = (object[,])ExcelApp.Range["A1:C1"].Value2; //用二
    维数组来接受单元格数据
16     }

```

代码中，arr是一个一维数组，存有4个数字。在第5行代码中，单元格区域“A1:D1”这个行向区域接受arr。

与此对应的VBA代码如下：

```

1 Public Sub 一维数组与Range数据交换()
2     Dim arr As Variant
3     arr = Array(3, 5, 7, 9)
4     Dim rg As Excel.Range
5     Set rg = ExcelApp.Range("A1:D1")
6     rg.Value = arr '行向区域接受一维数组的数据
7     Dim rg2 As Excel.Range
8     Set rg2 = ExcelApp.Range("A1:A4")
9     rg2.Value = ExcelApp.WorksheetFunction.Transpose(arr) '列向区域接受一维数
    组的数据
10 End Sub

```

4.7 操作Commandbar对象

```

1     public void 创建工具栏()
2     {

```

```
3         try
4         {
5             ExcelApp.CommandBars["cmb"].Delete();
6         }
7         catch (System.SystemException ex)
8         {
9             MessageBox.Show(ex.Message);
10        }
11        Office.CommandBar cmb = ExcelApp.CommandBars.Add("cmb", Office.
MsoBarPosition.msoBarTop, Type.Missing, true);
12        cmb.Visible = true;
13        for (int i = 1; i < 10; i++)
14        {
15            Office.CommandBarButton btn = (Office.CommandBarButton)cmb.
Controls.Add(Office.MsoControlType.msoControlButton, Type.Missing, Type.Missing,
Type.Missing, true);
16            btn.Caption = i.ToString();
17            btn.FaceId = i;
18            btn.Style = Office.MsoButtonStyle.msoButtonIconAndCaption;
19            btn.Click += new Office._CommandBarButtonEvents_
ClickEventHandler(btn_Click);
20        }
21        ExcelApp.CommandBars["Standard"].Controls[1].Visible = false;//隐藏
标准工具栏第一个控件
22    }
23    public void btn_Click(Microsoft.Office.Core.CommandBarButton Ctrl, ref
bool CancelDefault)
24    {
25        switch (Ctrl.Caption)
26        {
27            case "1":
28
29                break;
30            case "2":
31
32                break;
33            default:
34                ExcelApp.ActiveCell.Value = Ctrl.Caption;
35                break;
36        }
37    }
```

代码中第11行为ExcelApp增加一个名为“cmb”的用户自定义工具栏，然后依次添加10个工具栏按钮，并为这些按钮创建单击事件过程。

第23~35行为按钮的单击事件过程，使用switch结构判断按钮的标题文字，对每个按钮的单击分别进行处理。

与此对应的VBA代码如下：

```
1 Public Sub 创建工具栏()  
2     On Error Resume Next  
3     ExcelApp.CommandBars("cmb").Delete  
4     Dim cmb As Office.CommandBar  
5     Set cmb = ExcelApp.CommandBars.Add("cmb", msoBarTop, True)  
6     cmb.Visible = True  
7     Dim i As Integer  
8     Dim btn As Office.CommandBarButton  
9     For i = 1 To 10  
10        Set btn = cmb.Controls.Add(Office.MsoControlType.msoControlButton)  
11        btn.Caption = i  
12        btn.FaceId = i  
13        btn.Style = Office.MsoButtonStyle.msoButtonIconAndCaption  
14        btn.OnAction = "btn_Click"  
15    Next i  
16    ExcelApp.CommandBars("Standard").Controls(1).Visible = False '隐藏标准工具栏  
17 End Sub  
18 Public Sub btn_Click()  
19     '这个回调过程,需要写在VBA模块中  
20     Dim cap As String  
21     Set ExcelApp = Application  
22     cap = ExcelApp.CommandBars.ActionControl.Caption  
23     Select Case cap  
24     Case 1, 2  
25  
26     Case Else  
27         ExcelApp.ActiveCell = cap  
28     End Select  
29 End Sub
```

4.8 操作VBE工程

在C#中,也可以操作和控制工作簿的VBA工程组件。需要为项目添加“Microsoft Visual Basic for Applications Extensibility 5.3”这个外部引用。

■ 4.8.1 引用VBIDE类型库

在Visual Studio中,单击菜单【项目/添加引用】,在打开的对话框中单击【程序集/扩展】,勾选“Microsoft.Vbe.Interop”,如图4.6所示。

■ 4.8.2 允许对VBA工程访问

在Excel 2010中,激活选项卡“开发工具”,单击“宏安全性”,在打开的对话框中,勾选“宏设置”中的“信任对VBA工程对象模型的访问”,如图4.7所示。

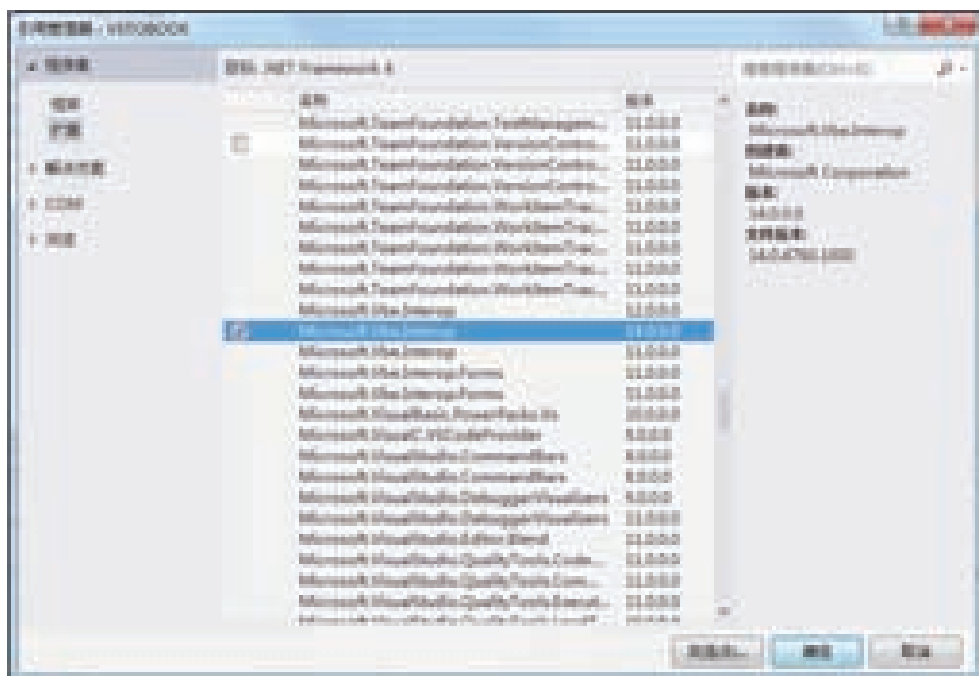


图4.6 引用VBE

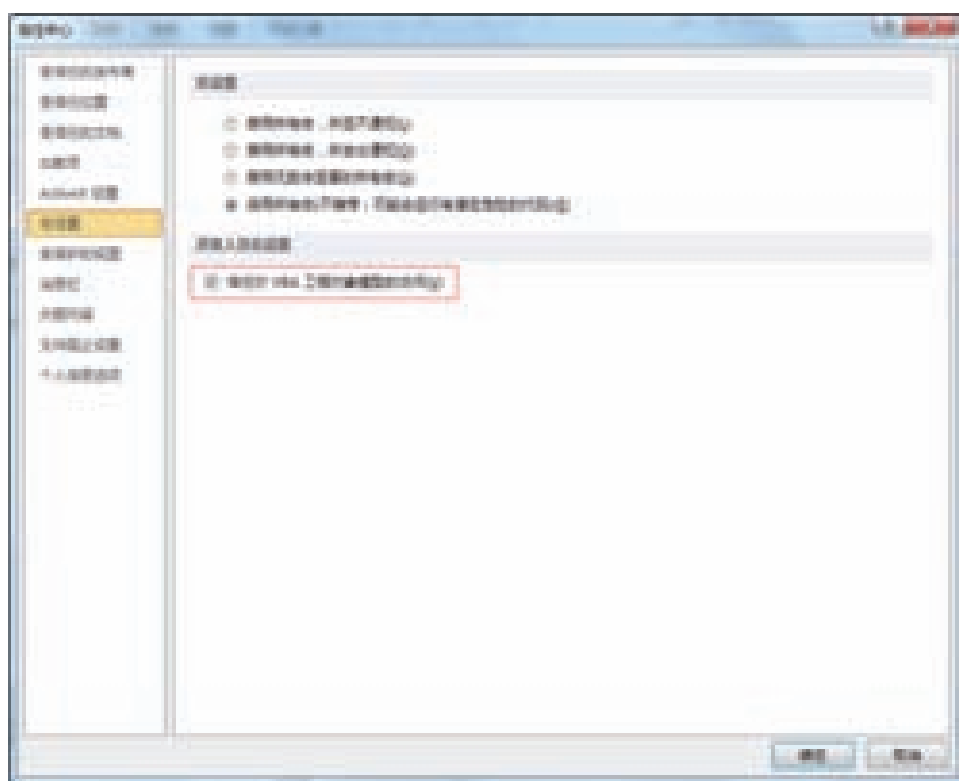


图4.7 信任对VBA工程的访问设置

■ 4.8.3 操作VBE各级对象

```

1      public void 操作VBE()
2      {      //需要引用"Microsoft Visual Basic for Applications Extensibility
5.3"
3          //必须在Excel中勾选"信任对VBA工程对象模型的访问"
4          Microsoft.Vbe.Interop.VBE vbe = ExcelApp.VBE;
5          Microsoft.Vbe.Interop.VBComponent vbc = vbe.ActiveVBProject.
VBComponents.Add(Microsoft.Vbe.Interop.vbext_ComponentType.vbext_ct_StdModule);
6          vbc.Name = "newModule";
7          //插入一个新的标准模块,并且重命名
8          vbc.CodeModule.InsertLines(1, "Sub Test()\n\tMsgBox 123\nEnd
Sub");//自动插入代码
9      }

```

上述代码中, 对象变量vbe是Excel的VBE环境的顶层对象, vbc是VBE的模块对象, 代码的第5行往VBA过程插入一个名为“newModule”的标准模块, 然后往该模块中自动写入了一个Test过程。

与此对应的VBA代码如下:

```

1 Public Sub 操作VBE()
2     '需要引用"Microsoft Visual Basic for Applications Extensibility 5.3"
3     '必须在Excel中,勾选"信任对VBA工程对象模型的访问"
4     Dim vbe As VBIDE.vbe
5     Dim vbc As VBIDE.VBComponent
6     Set vbe = ExcelApp.vbe
7     Set vbc = vbe.ActiveVBProject.VBComponents.Add(vbext_ct_StdModule)
8     vbc.Name = "newModule"
9     '插入一个新的标准模块,并且重命名
10    vbc.CodeModule.InsertLines 1, "Sub Test()" & vbNewLine & "        MsgBox 123"
& vbNewLine & "End Sub" '自动插入代码
11 End Sub

```

4.9 创建Excel自定义函数

Excel中有很多内置的函数, 例如求和、求平均、字符串操作函数、金融函数等。有些时候, 结合业务要求, 这些函数可能不能满足我们的需求。本节介绍如何用C#创建自定义函数, 并在Excel的工作表中使用。其实创建自定义函数的过程本身和Office以及Excel没有任何关系, 只是创建好的函数可以用于Excel。

下面编制一个判断是否是闰年的函数, 我们首先要明确, 该函数的参数是年份, 是int型的, 返回值是布尔型。

闰年的判断算法如下:

如果年份不是100的整数倍，则判断其能否被4整除，如果能则是闰年。

如果年份是100的整数倍，则判断其能否被400整除，如果能则是闰年。

举例说明：1996不是100的整数倍，但是可以整除4，因此是闰年；3000是100的整数倍，但是除以400会有余数，因此不是闰年。

根据上述思路在C#中不难写出IsLeap这个函数，代码如下：

```
1      public bool IsLeap(int year)
2      {
3          if (year % 4 == 0 && year % 100 != 0)
4              return true;
5          else if (year % 400 == 0)
6              return true;
7          else
8              return false;
9      }
```

■ 4.9.1 使用C#创建类库

启动Visual Studio 2012，单击菜单【文件/新建/项目】在“新建项目”对话框中，选择【Visual C#/类库】，名称中填写“UDF20160521”，如图4.8所示。

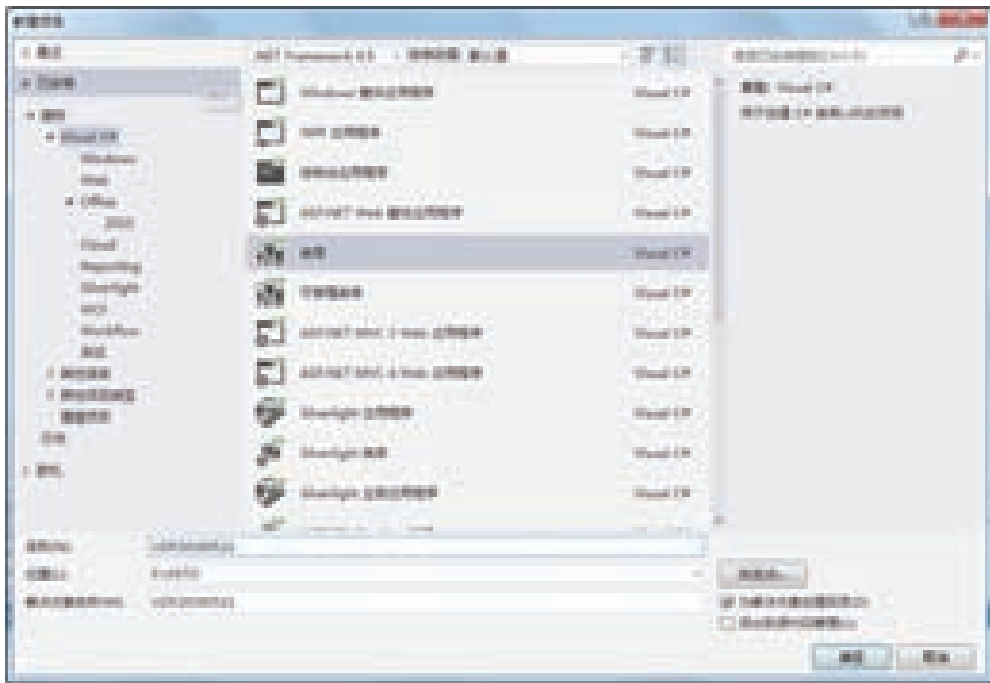


图4.8 创建类库项目

单击“确定”按钮后，在Visual Studio中自动打开Class1.cs这个类模块，全选原有代码并删除，复制如下代码到该类模块中。

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.Runtime.InteropServices;
7  using Microsoft.Win32;
8  namespace UDF20160521
9  {
10     [Guid("64F90FE5-C7F9-425B-BDA0-1A6F8F24B602")]
11     // 工具/创建GUID
12     [ClassInterface(ClassInterfaceType.AutoDual)]
13     [ComVisible(true)]
14     public class ClassLeap
15     {
16         #region COM related
17         [ComRegisterFunction]
18         public static void RegisterFunction(Type type)
19         {
20             Registry.ClassesRoot.CreateSubKey(GetSubKeyName(type,
21 "Programmable"));
22             var key = Registry.ClassesRoot.OpenSubKey(GetSubKeyName(type,
23 "InprocServer32"), true);
24             key.SetValue("", Environment.SystemDirectory + @"\mscoree.dll",
25 RegistryValueKind.String);
26         }
27         [ComUnregisterFunction]
28         public static void UnregisterFunction(Type type)
29         {
30             Registry.ClassesRoot.DeleteSubKey(GetSubKeyName(type,
31 "Programmable"), false);
32         }
33         private static string GetSubKeyName(Type type, string subKeyName)
34         {
35             var s = new System.Text.StringBuilder();
36             s.Append(@"CLSID\{");
37             s.Append(type.GUID.ToString().ToUpper());
38             s.Append(@"}\");
39             s.Append(subKeyName);
40             return s.ToString();
41         }
42         /// <summary>
43         /// 将Object类的4个公共方法隐藏
44         /// 否则将会出现在Excel的UDF函数中
45         /// </summary>
46         /// <returns></returns>
47         [ComVisible(false)]
48         public override string ToString()
49         {

```



```

48         return base.ToString();
49     }
50
51     [ComVisible(false)]
52     public override bool Equals(object obj)
53     {
54         return base.Equals(obj);
55     }
56
57     [ComVisible(false)]
58     public override int GetHashCode()
59     {
60         return base.GetHashCode();
61     }
62     #endregion
63
64     // 以下为自定义Excel函数
65     // 项目属性设置：生成/勾选“为COM互操作注册”
66     public bool IsLeap(int year)
67     {
68         if (year % 4 == 0 && year % 100 != 0)
69             return true;
70         else if (year % 400 == 0)
71             return true;
72         else
73             return false;
74     }
75 }
76 }

```

上述代码中第10行的GUID字符串需要读者重新创建并替换，具体步骤是：

第1步：在Visual Studio中单击菜单【工具/创建GUID】，如图4.9所示。



图4.9 创建GUID

第2步：“GUID格式”中选中第5个单选按钮。

第3步：单击“新建GUID”按钮，然后单击“复制”按钮，最后单击“退出”按钮。

第4步：回到类模块的第10行，删除原有GUID，按下Ctrl+V粘贴新的GUID。

注意：代码中第66~74行是用来判断是否闰年的自定义函数。如果读者还需要增加更多自定义函数，请在此处追加。

1. 修改项目生成属性

在Visual Studio中单击菜单【项目/UDF20160521属性】打开项目的属性文件，在打开的界面中，切换到“生成”选项卡，选中“为COM互操作注册”复选框，如图4.10所示。

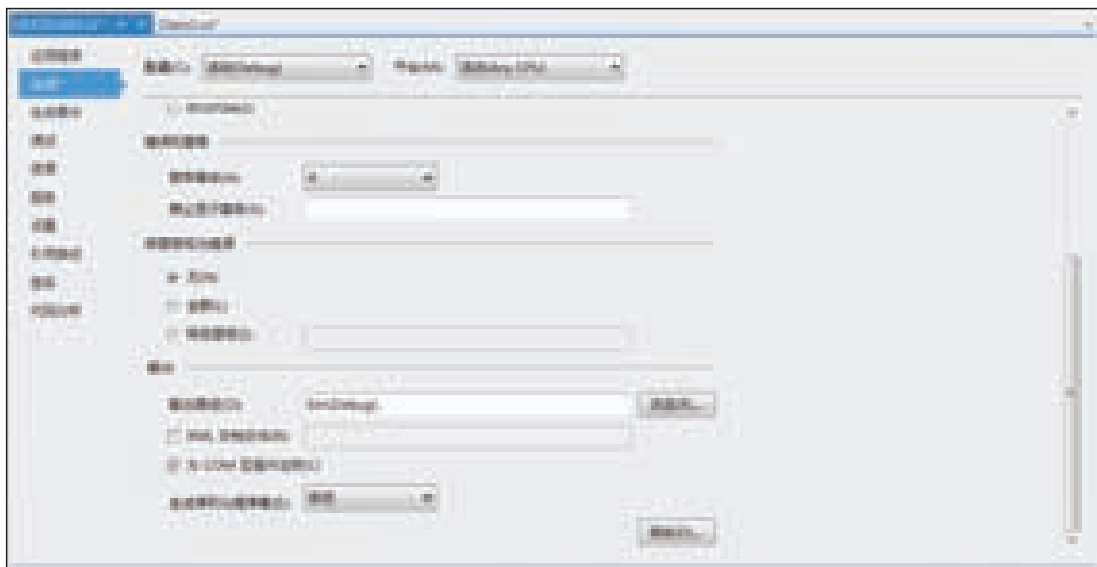


图4.10 修改项目的生成属性

然后关闭属性文件，继续切换到Class1.cs类模块中。

2. 生成自定义函数

类库的生成文件是一个dll动态链接库文件，为此，在Visual Studio中单击菜单【项目/生成解决方案】，会提示生成成功。

找到路径“F:\VSTO\UDF20160521\UDF20160521\bin\Debug”，会看到Debug文件夹中有了UDF20160521.dll这个文件。

此时可以关闭该解决方案或者完全退出Visual Studio。

4.9.2 工作表中使用C#开发的自定义公式

启动Excel 2010，单击选项卡【开发工具/加载项】，在“加载宏”对话框中，单击“自动化”按钮，在“自动化服务器”对话框中，拖动滚动条找到UDF20160521.ClassLeap

这一项，单击“确定”按钮，会在“加载宏”对话框中看到已经勾选了该项，如图4.11所示。

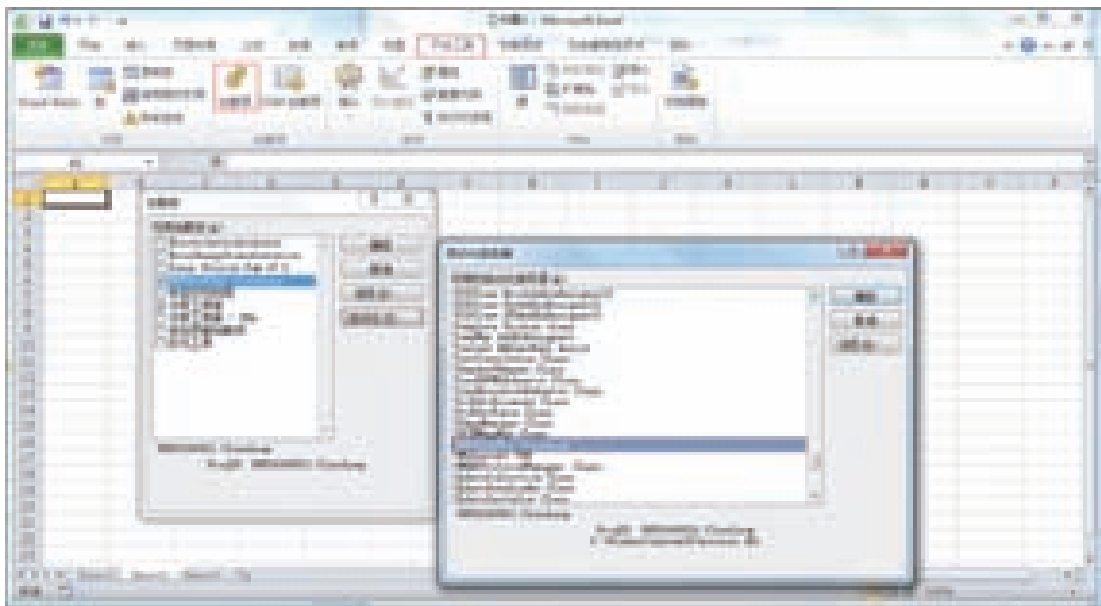


图4.11 加载自定义函数

关闭“加载宏”对话框，在工作表界面中单击公式编辑栏旁边的“fx”按钮，出现“插入函数”对话框，如图4.12所示。

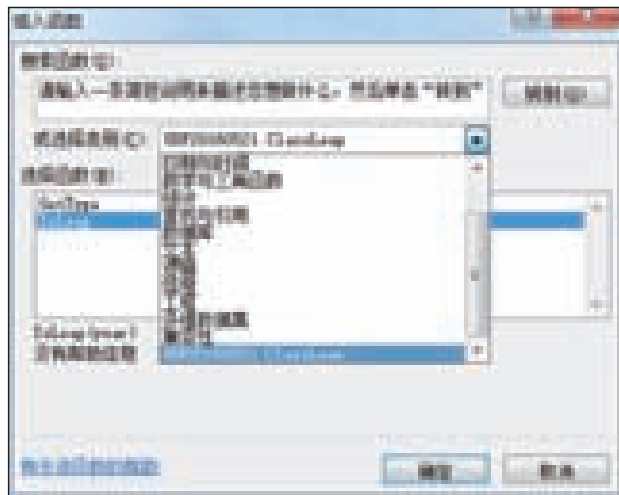


图4.12 “插入函数”对话框

在Excel中成功添加了该自定义函数，在单元格中输入公式“=IsLeap(2003)”，会看到False。

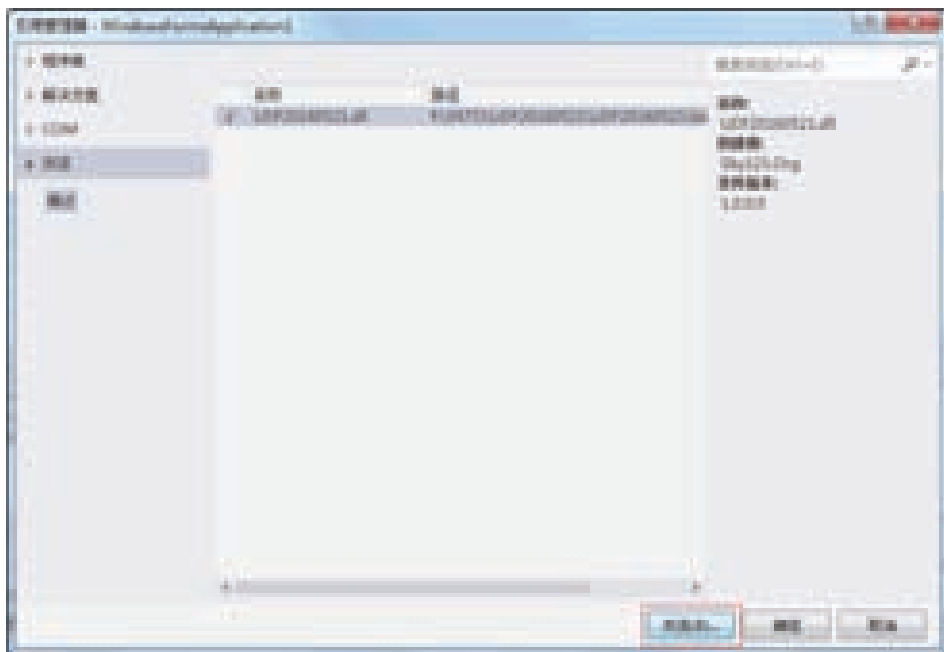


图4.14 C#调用自定义公式

在解决方案资源管理器中，可以看到引用列表中多了“UDF20160521”这一项。回到Form1的设计视图，双击Button1按钮，编辑其单击事件过程：

```

1      private void button1_Click(object sender, EventArgs e)
2      {
3          UDF20160521.ClassLeap Test = new UDF20160521.ClassLeap();
4          int year = Convert.ToInt16(this.textBox1.Text);
5          MessageBox.Show(Test.IsLeap(year).ToString());
6      }

```

启动运行，在窗体的文本框中输入年份，单击“是否闰年”按钮，会判断该年份是否为闰年，运行效果如图4.15所示。

■ 4.9.5 客户机使用C#制作的自定义函数

上述演示均是在开发机器上面进行的测试，那么创建好的自定义函数能否用在其他计算机上呢？要想在其他计算机上使用该UDF，必须在客户机中对dll文件进行注册。

在开发机器中找到路径“F:\VSTO\UDF20160521\UDF20160521\bin\Debug”，把Debug文件夹整体复制或发送到客户机。



图4.15 运行效果

在客户机中单击计算机的【开始】菜单，再依次单击【所有程序/附件/命令提示符】，找到命令提示符后，以管理员身份运行，出现如下DOS命令窗口，如图4.16所示。

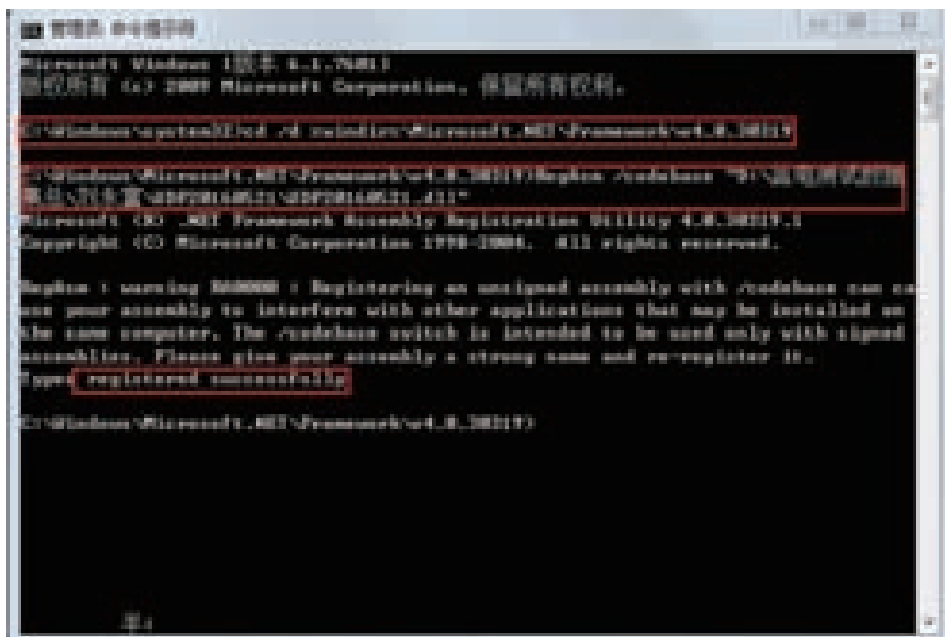


图4.16 注册自定义函数

注意 注意该窗口的标题中，有“管理员”字样。

输入`cd /d %windir%\Microsoft.NET\Framework\v4.0.30319`，按【Enter】键运行，接下来输入`RegAsm /codebase “完全路径”`，按【Enter】键，会看到提示Successfully，表示注册成功。

上面的完全路径，读者需要根据客户机dll文件的实际路径修改一下。

注册成功后，启动Excel，尝试加载该函数，这时客户机可以和开发机器一样，在工作表中使用该函数了。

本章要点回顾

- 本章介绍C#操作和控制Office对象，应学会如何向C#项目中添加Excel和Office这两个外部引用。
- 熟悉Excel对象模型，会用C#操作和访问Excel最常用的对象，会遍历集合对象中的成员。
- 区别对待Excel常用对象的属性、方法和事件，理解事件的添加和移除的原理。
- 单元格区域和C#数组之间的数据传递，这部分是基础且重要的知识。

第5章

创建Office外接程序

Office外接程序是指利用其他编程工具（如Visual Basic 6.0.C#）编写的，用于操作和控制Office对象的插件，也叫作Office COM加载项。利用传统的VBA编写的作品，程序代码寄生在Office文档中，这类程序不能叫作Office外接程序。

 本章视频：创建Office外接程序.wmv

5.1 Office COM加载项简介

可以使用一般的加载项和COM加载项来扩展Office的功能，一般加载项也可以称为加载宏，通常是用VBA写成的，文件格式类似于Office文档格式；Word、Excel、PowerPoint、Access都可以使用相应的文档另存为加载宏，Office 2003以下版本可用的加载项扩展名分别为.dot、.xla、.ppa、.mde，Office 2007以上版本扩展名可以为.dotm、.xlam、.ppam等。而COM加载项则是使用其他语言写成的用于Office的插件，通常用户可以使用VB6或VSTO开发Office的COM加载项，本书只讨论后者。

5.2 认识Office COM加载项管理对话框

Office所有的COM加载项的加载和卸载操作，均在“COM加载项”对话框中完成。对于Office 2003以下版本，Office界面上找不到COM加载项，需要从“自定义”对话框中调出。而Office 2010的COM加载项，需要显示出“开发工具”选项卡。“开发工具”选项卡中主要集中了VBA编程方面需要使用的控件，要显示该选项卡，需要单击【选项/自定义功能区】，选中“开发工具”前的复选框。

单击“开发工具”选项卡中的“COM加载项”，会看到如图5.1所示的对话框。

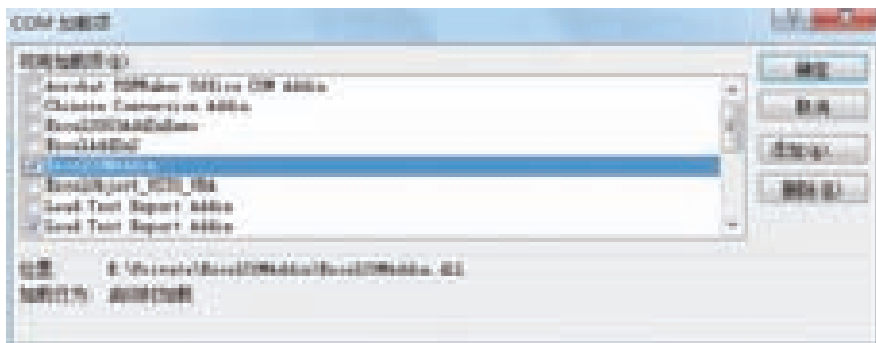


图5.1 Office的“COM加载项”对话框

当我们调试VSTO外接程序或者安装了他人创建好的COM加载项时，Office的COM加载项中就会出现相应项，通过选中和取消选中实现COM加载项的加载和卸载。

5.3 创建第一个Office外接程序项目

启动Visual Studio 2012，单击【文件/新建/项目】，出现“新建项目”对话框。左侧树形结构分别展开【模板/Visual C#/Office/2010】，右侧项目类型选择“Excel 2010外接程序”，下方的项目名称、项目路径用户自行指定，单击“确定”按钮，如图5.2所示。

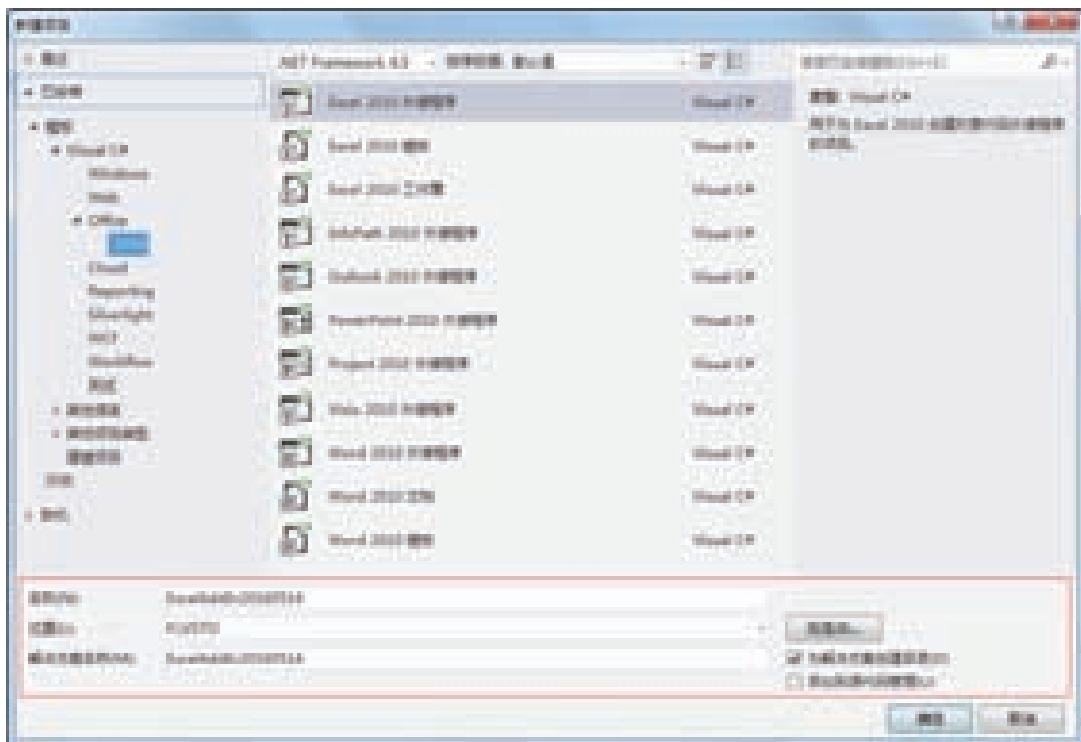


图5.2 创建VSTO项目

5.4 ThisAddIn的启动事件和卸载事件

VSTO项目创建完毕后,注意代码窗口的右侧“解决方案资源管理器”中有一个Excel/ThisAddIn.cs类模块,如图5.3所示。

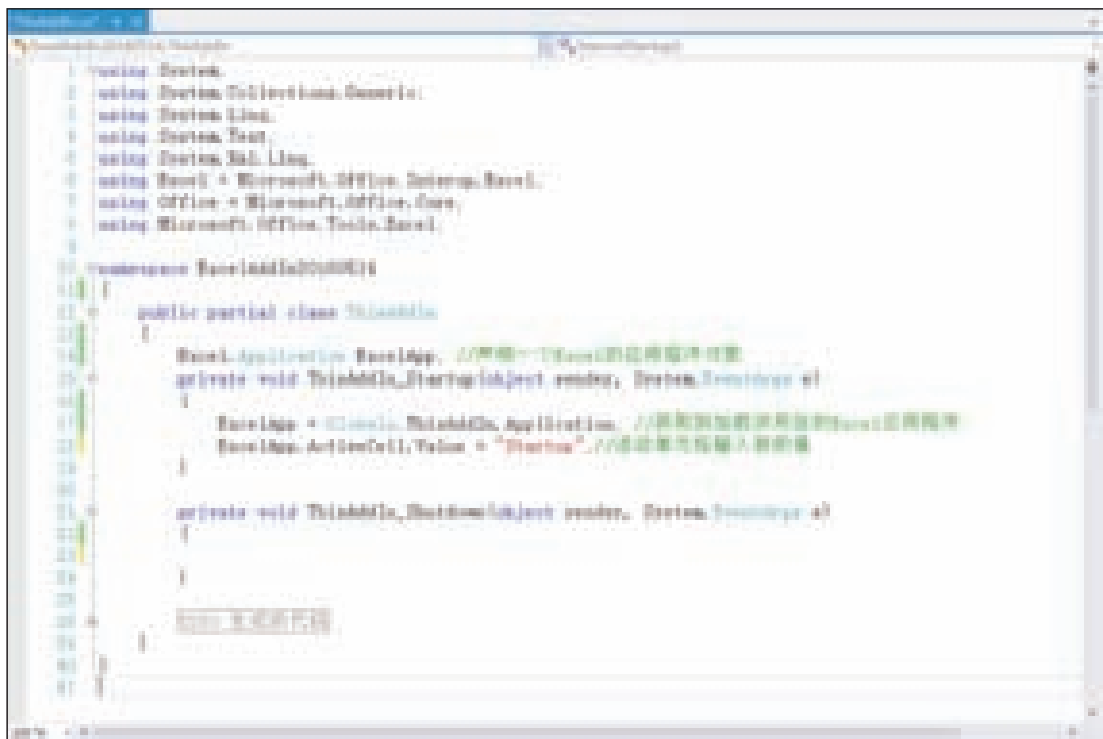


图5.3 Excel外接程序的入口函数

注意 上图带有注释的3行代码是笔者加入的,并非模板自动创建。

该类模块中,第1~8行是模块顶部的using语句部分,第6行和第7行使用Excel和Office两个缩写单词代替对Excel模型库和Office类库的引用。

第14行声明一个Excel的应用程序对象ExcelApp,在整个加载项加载期间,一直使用这个顶层对象来操作和控制Excel程序,第15~19行是加载项启动事件过程ThisAddIn_Startup。当加载项被加载时,首先运行该事件过程中的代码。此刻按下快捷键【F5】,或者单击Visual Studio工具栏中的【启动】按钮,就会看到自动启动了新的Excel,并且会看到Excel的活动单元格中输入了新的内容,说明加载项运行正常。

此时如果打开Excel的“COM加载项”对话框,会看到里面有“ExcelAddIn20160514”这一项;如果取消选中该项前面的复选框,或者完全退出Excel都将卸载该加载项,并且引发VSTO中ThisAddIn_Shutdown事件过程中的代码。

至此，一个最基本的Office外接程序创建完成，但是在Excel中没有看到任何界面的变化，只是可以使用外接程序来读写Excel对象了。以后的章节会讲到使用VSTO向Office中增加新的元素，都是基于这个最基本的VSTO项目。

本章要点回顾

- 熟练操作Office的“COM加载项”对话框，“COM加载项”对话框是连接VSTO项目与Office应用程序的桥梁，外接程序的加载和卸载都通过这个对话框来完成。
- 使用Visual Studio创建第一个Office外接程序，要弄清楚外接程序加载后的入口程序。
- Excel应用程序变量ExcelApp，在外接程序一加载，就获取到Excel对象。程序的其他位置需要操作和控制Excel对象时，都可以通过ExcelApp访问到。

第6章

自定义Office功能区

使用VSTO向Office中引入新的界面有以下种方式：

- Office中显示C#窗体。
- 自定义Office功能区。
- 创建自定义任务窗格。
- 创建自定义工具栏和控件。

这些新的界面虽然外观和行为有所不同，但是作用和意义却是一致的，都是为了操作控制Office程序，为了和Office程序有更好的交互。

6.1 CustomUI概述

自Office 2007版本，微软开始使用功能区来装载各种命令，取代了以往的工具栏菜单方式，虽然自定义工具栏界面仍然可以在【加载项】选项卡中看到，但是功能显然被弱化了。更改工具栏和控件的技术称为“自定义工具栏”，而更改Office高级版本界面的技术叫作Custom User Interface，简称CustomUI。

Office可以定制的界面有以下几部分：

- backstage视图
- 常用功能区（ribbon/tabs）
- 快速访问工具栏（qat）
- 环境选项卡（ribbon/contextualTabs）
- 对象右键快捷菜单（contextMenus）

其中，上述常用功能区和快速访问工具栏以及环境选项卡这3部分可以在Office中手工定制，而且其他部分只能使用XML代码定制。

以上部分在Excel中的位置如图6.1所示。

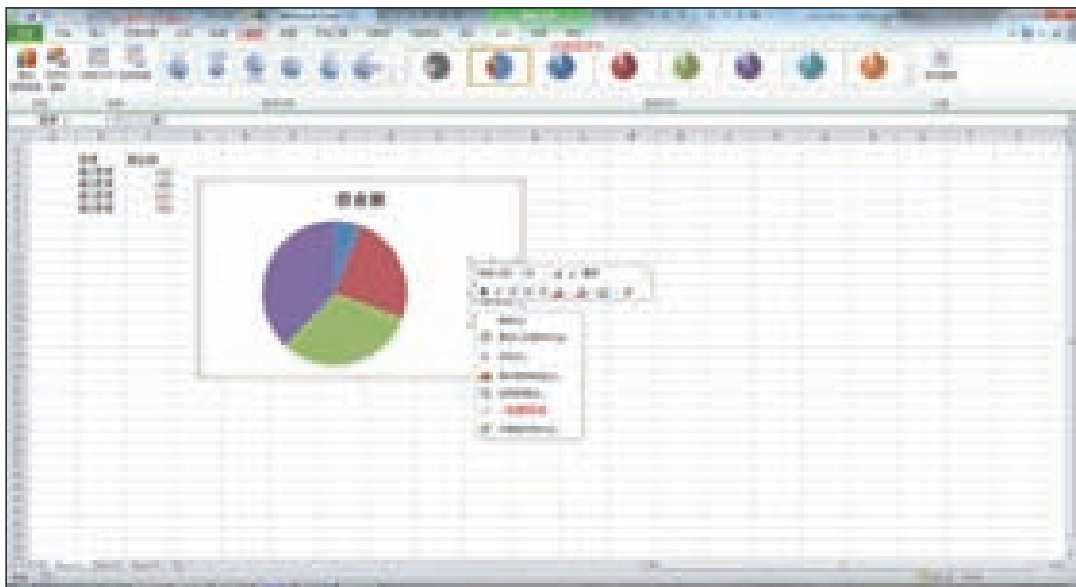


图6.1 VSTO可以定制的区域

注意 图中红色标记部分。

环境选项卡是指用户选择Excel中其他对象的时候，例如选择了一个图表，会在Excel顶部出现“图表工具”这个选项卡，如果不选择图表，这个选项卡就看不到，这样的选项卡称为“环境选项卡”。就是说它的显示、隐藏和鼠标所选对象有关。

其中backstage视图是指单击【文件】后，出现图6.2所示的界面。



图6.2 backstage视图

和自定义工具栏意义基本一样，自定义界面的意义主要有以下两点：

- 由此可见，凡是引起Office界面变化的技术都属于CustomUI。

CustomUI作用范围可以分为如下三个级别：

- 使用VSTO创建的Office外接程序，涉及的自定义界面属于应用程序级；而使用VSTO创建的Office模板或文档（在后续章节讲到）添加的自定义界面属于文档级或模板级。

启动Excel 2010，单击【文件/选项/自定义功能区】，出现如图6.3所示的对话框。



用户可以在这个对话框中,把左侧的命令追加到右侧的选项卡中,也可以隐藏部分选项卡、组,或进行删除控件等操作。对于Office的其他组件的界面定制,和在Excel中一样,请读者自行实践。

如果要自定义快速访问工具栏,单击【文件/选项/自定义快速访问工具栏】,方法类似,不再赘述。

6.2 CustomUI与XML

XML是指可扩展标记语言(EXTensible Markup Language),语法和HTML语言类似。Office高级版本的界面用XML语言表述。

6.2.1 XML语法规则

一个典型的XML由若干节点及其属性组成,节点之间可以嵌套。描述一个节点的语法规则如下:

```
<节点名称 属性1="属性值" 属性2="属性值">  
</节点名称>
```

一个节点的结束标记是</节点名称>,如果该节点包含子节点,那么该节点的结束标记应加在所有子节点之后。属性值无论是哪一种数据类型,都必须用半角双引号括起来。

```
<部门 名称="销售部" 电话="010-64268888">  
  <员工 姓名="张三" 年龄="35">  
  </员工>  
  <员工 姓名="李四" 年龄="65" 政治面貌="群众">  
  </员工>  
</部门>
```

上述XML代码中,描述了销售部的名称和电话,此外,销售部有两名员工。部门节点包含员工节点。描述嵌套的数据结构就是XML的优势之一。

如果节点不包含任何子节点,结束标记可以简化,上述代码中,张三这个员工节点可以简化为:

```
<员工 姓名="张三" 年龄="35" />
```

也就是说,在该行结尾的>前面加斜杠,省去了后续的</员工>标记。

XML严格区分大小写,比如标签<Letter>与标签<letter>是不同的。

在XML中编写注释的语法与HTML的语法很相似,输入<!--注释部分-->作为注释。

```
<部门 名称="销售部" 电话="010-64268888">
  <员工 姓名="张三" 年龄="35">
  </员工>
  <员工 姓名="李四" 年龄="65">
  </员工>
  <!--这里是注释部分-->
</部门>
```

■ 6.2.2 描述Office界面的XML

描述Office界面的XML的最顶层节点是<customUI>，下一级节点的名称与自定义的Office界面部分有关，下面举例介绍。

1. 自定义backstage

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
onLoad=" onLoad" >
  <backstage>
    <tab id="dynamicVisibilityTab"label="BackStage" insertAfterMso="TabRecent">
      <firstColumn>
        <group id="specDetailsGroup"label="Group1" helperText="帮助文字">
          <primaryItem>
            <menu id="switchMenu" label="菜单" imageMso="ControlLayoutSt
acked">
              <menuGroup id="menuGroup">
                <button id="switchGroups" label="命令1" onAction=
"SwitchGroups" imageMso="C"/>
                <button id="switchGroups2" label="命令2" onAction=
"SwitchGroups" imageMso="D"/>
              </menuGroup>
            </menu>
          </primaryItem>
          <topItems>
            <editBox id="specTitle"label="Title:" getText= "GetSpecDetailText"/>
            <editBox id="specDesigner" label=" Designer:" getText= "GetSpecDetailText"/>
            <editBox id="specEngineer" label=" Engineer:" getText= "GetSpecDetailText"/>
            <editBox id="specTeam"label="Team:" getText= "GetSpecDetailText"/>
            <editBox id="specCost" label="Cost:" getText= "GetSpecDetailText"/>
          </topItems>
        </group>
        <group id="marketingGroupDetails" label="Marketing Group">
          <primaryItem>
            <button id="marketingButton" label="Marketing"
imageMso="OutlookGlobe"/>
          </primaryItem>
          <topItems>
            <editBox id="marketingManager" label="Manager:
"getText="GetMarketingDetail"/>
            <editBox id="marketingBudget" label=" Budget:
"getText="GetMarketingDetail"/>
```

```

        <editBox id="marketingEndDate" label="Completion Date:"
getText="GetMarketingDetail"/>
    </topItems>
</group>
<group id="engineeringGroupDetails" label="Engineering Group">
    <primaryItem>
        <button id="engineeringButton" label="Engineering"
imageMso="TableDesign"/>
    </primaryItem>
    <topItems>
        <editBox id="engineeringManager" label="Manager:"
" getText="GetEngineeringDetail"/>
        <editBox id="engineeringBudget" label="Budget:"
" getText="GetEngineeringDetail"/>
        <editBox id="engineeringEndDate" label="Completion Date: "
getText="GetEngineeringDetail"/>
    </topItems>
</group>
</firstColumn>
<secondColumn>
    <taskGroup id="bidProcessTaskGroup" label="Group2">
        <category id="defineWorkScopeCategory" label="Define work and
tasks - Complete tasks in order.">
            <task id="defineScope" label="Define the Scope of Work." imageMso=
"_1" onAction="SetTaskComplete"/>
            <task id="assignTasks" label="Assign the Tasks" imageMso=
"_2" onAction="SetTaskComplete"/>
        </category>
        <category id="calculateCostsCategory" label="Calculate costs -
Complete tasks in order.">
            <task id="calcManHours" label="Calculate Total Man-Hours"
imageMso="_3" onAction="SetTaskComplete"/>
            <task id="calcOverheadCosts" label="Determine Overhead
Costs" imageMso="_4" onAction="SetTaskComplete"/>
        </category>
    </taskGroup>
    <group id="tasksCompleteGroup">
        <topItems>
            <layoutContainer id="tasksCompleteLayout" layoutChildren=
"horizontal">
                <imageControl id="tasksCompleteImage" imageMso=
"AcceptInvitation"/>
                <labelControl id="tasksCompleteLabel" label="The proposal
is ready for review."/>
            </layoutContainer>
        </topItems>
    </group>
</secondColumn>
</tab>
</backstage>
</customUI>

```


在Excel中的效果如图6.4所示。

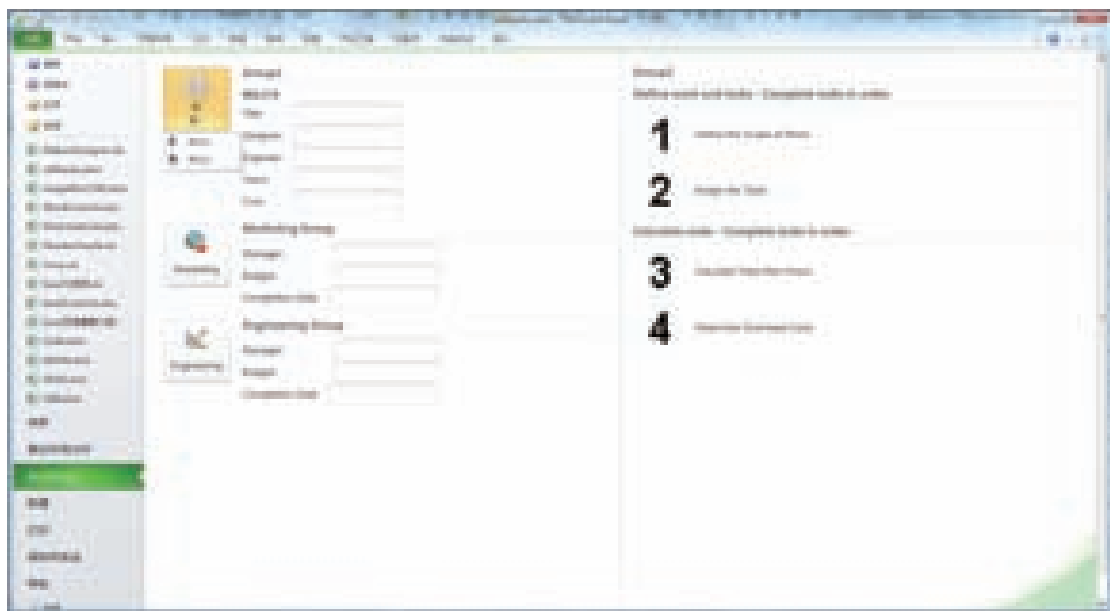


图6.4 自定义backstage视图

2. 自定义常用功能区

以下代码是自定义常用功能区的示例代码，在Excel中的效果如图6.5所示。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
onLoad="rxIRibbonUI_onLoad">
  <ribbon startFromScratch="false">
    <tabs>
      <tab id="DemoTab" label="RibbonDemo" insertBeforeMso="TabHome">
        <group id="DemoGroup1" label="New Group">
          <button id="rxbtnRollForward" label="button" imageMso="ChartTypeOtherInsertGallery"
screenTip="This is Screen Tip" supertip="This is super tip"
onAction="button_OA"/>
          <checkbox id="CKB1" label="checkBox" getPressed="rxchkR1C1_getPressed"
onAction="rxchkR1C1_click"/>
          <editBox id="rxtxtRename" label="editBox" imageMso="SignatureLineInsert"
keytip="R" sizeString="123456789" onChange="Edb_Click"/>
          <toggleButton id="IDBold" label="toggleButton" getPressed="Bold_getPressed"
imageMso="FieldList" onAction="Bold_Click"/>
          <comboBox id="rx cboSelectSheet" label="comboBox" onChange="comboBox_Click">
            <item id="item1" label="item1"/>
            <item id="item2" label="item2"/>
            <item id="item3" label="item3"/>
          </comboBox>
          <control idMso="Cut"/>
          <dropDown id="dropdown1" label="dropDown" onAction="dropDown_Click">
```

```

        <item id="item21" label="item1"/>
        <item id="item22" label="item2"/>
        <item id="item23" label="item3"/>
    </dropDown>
    <menu id="mnuResources" imageMso="HyperlinkInsert" label="menu">
        <menuSeparator id="menuS1" title="menuSeparator1"/>
        <button id="bt31" label="button1" onAction="button1_OA"/>
        <button id="bt32" label="button2"/>
        <menuSeparator id="menuS2" title="menuSeparator2"/>
        <button id="bt33" label="button3"/>
        <button id="bt34" label="button4"/>
    </menu>
    <splitButton id="split1">
        <button id="bt41" label="button41" screentip="splitButton"
imageMso="CreateReportFromWizard" onAction="button41_OA"/>
        <menu id="submenu1">
            <button id="bt42" label="button42" onAction="button42_OA"/>
            <button id="bt43" label="button42"/>
        </menu>
    </splitButton>
    <box id="box1" boxStyle="vertical" visible="true">
        <button id="bt51" screentip="box" label="button1"/>
        <button id="bt52" label="button2"/>
        <button id="bt53" label="button3"/>
        <button id="bt54" label="button4"/>
        <button id="bt55" screentip="box" label="button5"/>
    </box>
    <separator id="separator1"/>
    <buttonGroup id="buttonGroup1">
        <button id="bt61" screentip="buttonGroup" label="button1"/>
        <button id="bt62" label="button2"/>
        <button id="bt63" label="button3"/>
        <button id="bt65" keytip="G" label="button5"/>
    </buttonGroup>
    <labelControl id="LC1" label="labelControl"/>
    <gallery id="gall1" tag="large" label="Gallery" imageMso="G"
showItemLabel="true" size="large" itemWidth="32" itemHeight="32"
onAction="OnAction">
        <item id="l__3DBevelPictureTopGallery" imageMso="A"
label="A"/>
        <item id="l_BevelShapeGallery" imageMso="B" label="B"/>
    </gallery>
    <dialogBoxLauncher>
        <button id="dlgbutton" onAction="dlg_OA"
screentip="dialogBoxLauncher"/>
    </dialogBoxLauncher>
</group>
</tab>
</tabs>
</ribbon>
</customUI>

```

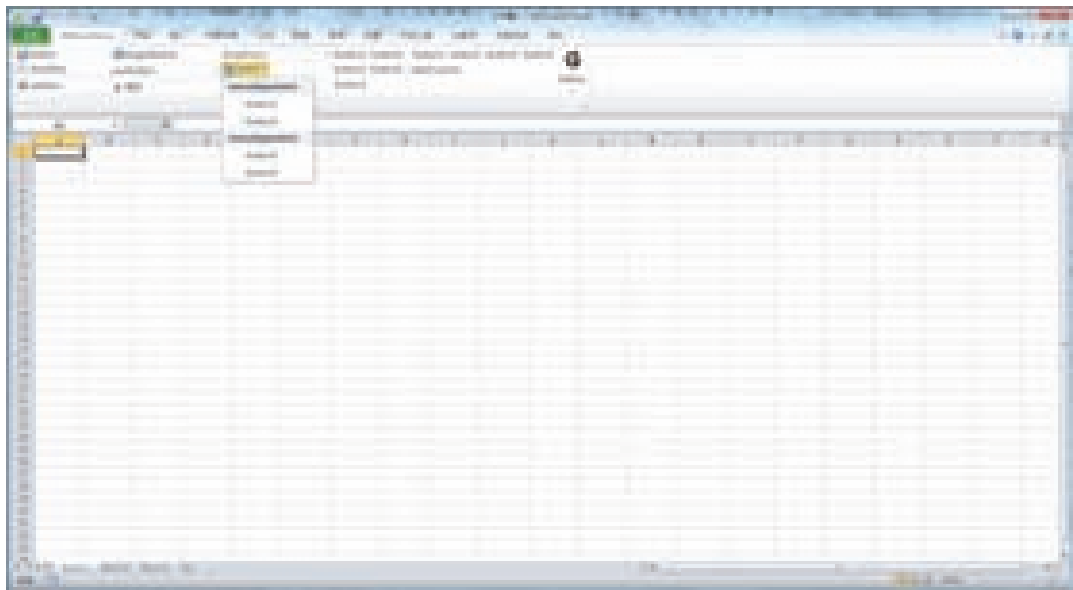


图6.5 自定义常用功能区

3. 自定义快速访问工具栏

以下代码是自定义快速访问工具栏的示例代码，对应的效果如图6.6所示。

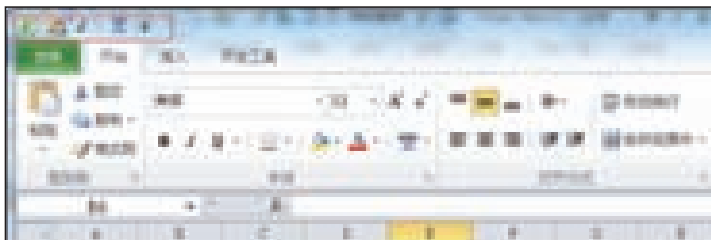


图6.6 自定义快速访问工具栏

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon startFromScratch="true">
    <!--这个属性只能是true-->
    <tabs>
      <tab idMso="TabHome" visible="true"/>
      <tab idMso="TabInsert" visible="true"/>
      <tab idMso="TabDeveloper" visible="true"/>
    </tabs>
    <qat>
      <sharedControls>
        <button idMso="Paste"/>
        <control idMso="Italic"/>
        <separator id="s1"/>
        <button id="bt1" label="新按钮" imageMso="EquationInsertNew"/>
      </sharedControls>
    </qat>
  </ribbon>
</customUI>
```

```
</qat>
</ribbon>
</customUI>
```

4. 自定义环境选项卡

以下代码是自定义环境选项卡的示例代码。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <contextualTabs>
      <!--环境选项卡的自定义。修改图片工具/格式的label, 隐藏图片排列组, 插入新组。-->
      <tabSet idMso="TabSetPictureTools">
        <tab idMso="TabPictureToolsFormat" label="图片格式">
          <group idMso="GroupArrange" visible="false"/>
          <group id="newGroup" label="ryueifu" insertBeforeMso="GroupPictureStyles">
            </group>
          </tab>
        </tabSet>
      </contextualTabs>
    </ribbon>
  </customUI>
```

使用自定义环境选项卡前默认的效果如图6.7所示。

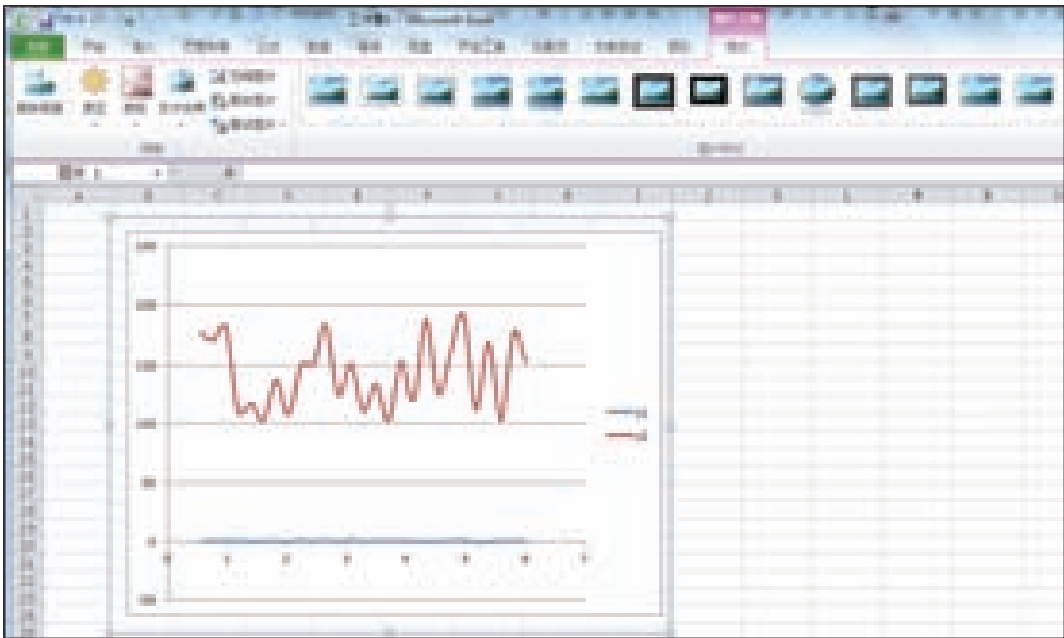


图6.7 图片的环境选项卡

使用自定义环境选项卡后的效果如图6.8所示。

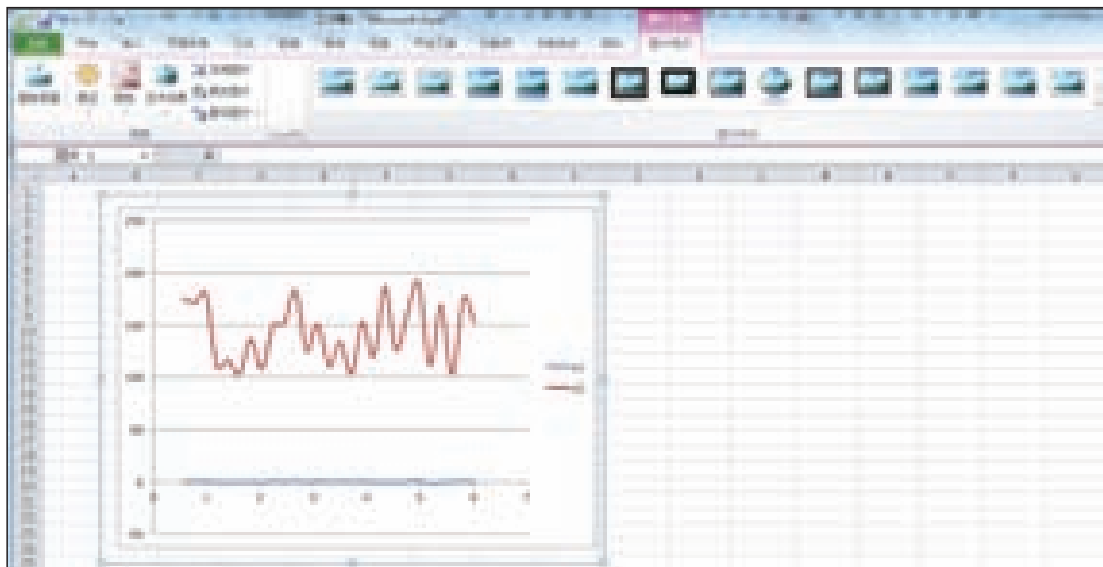


图6.8 自定义图片环境选项卡

可以看出“格式”选项卡的名称改为了“图片格式”，而且在“调整”组的右侧插入了一个空白组。

5. 自定义右键菜单

自定义右键菜单的示例代码如下：

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <contextMenus>
    <contextMenu idMso="ContextMenuCell">
      <button id="NewButton" label="π"
onAction="OA" imageMso="EquationInsertNew"
insertBeforeMso="Cut"/>
    </contextMenu>
  </contextMenus>
</customUI>
```

效果如图6.9所示。

右键菜单上方多了一个自定义按钮。

■ 6.2.3 使用Ribbon XML Editor

为了实时查看XML语句和Office界面的对应关系，笔者开发了Ribbon XML Editor，用户可以在编辑XML过程中，单击菜单【查看/Excel】，立即在Excel中看到新的界面变化。

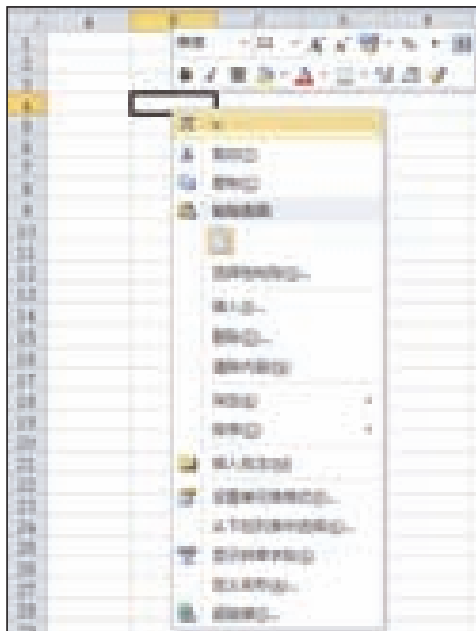


图6.9 自定义右键菜单

该工具支持Office的Access、Excel、Outlook、PowerPoint、Publisher和Word。关于该工具的其他功能，请读者下载工具后查看帮助文档。工具界面如图6.10所示。

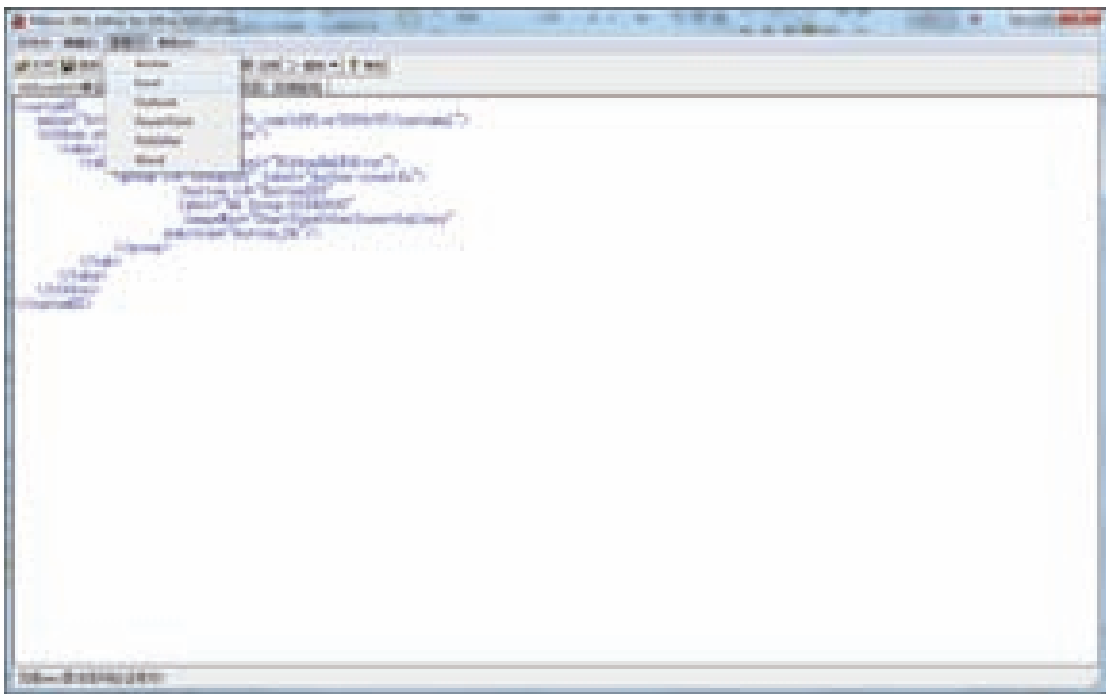


图6.10 Ribbon XML Editor

6.3 CustomUI元素详解

Office CustomUI拥有众多的成员节点和属性。本节通过典型实例重点讲述自定义常用功能区所用到的元素及其属性。

自定义常用功能区的XML层次结构为：

```
<customUI>
  <ribbon>
    <tabs>
      <tab>
        <group>
          <control/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

ribbon表示功能区，tabs表示选项卡集合，而tab表示单个选项卡，group表示选项卡中

的一个组，control泛指组中的各种类型的控件。因此常用功能区的元素可以理解为三级层次结构：tab里包含若干group，每一个group包含若干control。

■ 6.3.1 选项卡（tab）元素

tab的父级对象是tabs，子级对象是group。tab的常用属性有id（或idMso）、label、visible、insertBeforeMso（或insertAfterMso）。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon startFromScratch="false">
    <tabs>
      <tab idMso="TabFormulas" label="Equation"/>
      <tab idMso="TabData" visible="false"/>
      <tab id="CustomTab" label="CustomTab" insertAfterMso="TabDeveloper"/>
    </tabs>
  </ribbon>
</customUI>
```

上述XML代码的功能是：把Excel的内置选项卡“公式”的标题文字修改为“Equation”，把内置选项卡“数据”隐藏掉，并且新建一个自定义的空白选项卡“CustomTab”，这个新选项卡置于“开发工具”选项卡之后。

通过这个实例，可看出CustomUI的任何元素必须有id或idMso作为唯一标识，如果元素是内置的，必须使用idMso，反之，如果是用户自定义元素必须使用id，id属性值可以是不重复使用的字符串。

Excel 2010内置选项卡的label及idMso如表6.1所示。

表6.1 Excel 2010内置选项卡的label及idMso

label	idMso
开始	TabHome
插入	TabInsert
页面布局	TabPageLayoutExcel
公式	TabFormulas
数据	TabData
审阅	TabReview
视图	TabView
开发工具	TabDeveloper
加载项	TabAddins

实际上，idMso不仅用于内置选项卡，内置组、内置控件也都有相应的idMso属性。

注意 读者可以下载 Office2010ControlIDs.rar 这个资源文件了解Office所有组件的所有内置元素的idMso。

label属性是元素的标题文字，可以更改内置元素的标题文字，也可以为新的用户元素指定label属性。

visible属性是元素的可见性，visible属性几乎可以用于任何元素。

insertBeforeMso属性的含义是用户自定义元素置于内置元素之前。

读者可以把上述代码放在Ribbon XML Editor中加以验证。

■ 6.3.2 组 (group) 元素

group的父级对象是tab，group之下可以包含各种类型的control。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon startFromScratch="false">
    <tabs>
      <tab idMso="TabHome">
        <group idMso="GroupFont" visible="false">
        </group>
      </tab>
      <tab id="exampleID1" label="Demo">
        <group id="GroupID1" label="new group">
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

上述XML代码是把Excel 2010的“开始”选项卡中的“字体”内置组隐藏掉，并且新建一个“Demo”选项卡，该选项卡新建了一个“new group”，新组中无任何控件。group元素可用的属性和tab元素属性基本都一样。

此外，内置组也可以整体搬入其他选项卡中，如果把上述代码中的“group id="GroupID1"”修改为“group idMso="GroupFont"”，那么会在自定义选项卡Demo中出现字体组。

■ 6.3.3 控件 (control) 元素

功能区的组元素下可以包含的控件如表6.2所示。

表6.2 功能区组元素下包含的控件类型

控件名称	描述
框 (box)	将控件组合在一个组里。可以在框里放置任何控件，将控件按水平或垂直流向设置。必须提供boxStyle属性：vertical或horizontal。与按钮组控件不同，除了设置控件流向外，框控件不会提供可视化外观
按钮 (button)	提供基本的执行功能。单击按钮，会发生某项操作

控件名称	描述
按钮组 (buttonGroup)	将不同类型的按钮组合在一起。按钮出现在自然的框里面，并且Office将它们放置在一起表明其以某种方式联系。可以使用其组合下列控件：按钮、切换按钮、库、菜单、动态菜单和拆分按钮控件
复选框 (checkBox)	提供基本的选择功能。通过单击该控件，用户启用或禁用某选项。Office为复选框控件提供了两种上下文环境，包括作为单独的控件或者是菜单的一部分
组合框 (comboBox)	为用户显示一系列选项，使用item控件创建选项列表。每个comboBox控件必须包括至少一个item控件作为子项。当使用comboBox控件时，用户也能够输入一个不在列表中出现的值（dropDown控件需要用户选择列表中的某个项目）
通用控件（control）	代表任何类型的控件
dialogBoxLauncher	弹出对话框。组右下角的小箭头
下拉控件 (dropDown)	为用户显示一系列选项，使用item或button控件创建选项列表。列表至少应包含两类可选的控件之一。用户必须选择提供的列表中的某选项。当用户选择button而不是item时，Office执行所需求的动作，而不是选择期望的选项
动态菜单 (dynamicMenu)	定义在运行时而不是设计时所创建的菜单。菜单内容能够修改以满足特定的需要。必须包括getContent回调来使用该控件。dynamicMenu控件能够作为buttonGroup、menu或splitButton控件的一部分出现
编辑框（editBox）	让用户输入纯文本到功能区中。可以使用这项功能执行某任务，例如搜索。使用这个控件来进行任何不能通过使用其他控件来定义的输入
库（gallery）	在下拉结构中显示一组控件以节省功能区空间
标签控件 (labelControl)	在屏幕中创建标签。可以在标签控件组或者不易使其以其他形式识别的元素中使用该控件
菜单（menu）	定义在设计时创建的菜单。菜单可以包含诸如button和checkBox控件之类的控件。可以单独使用菜单，也可以作为splitButton控件的一部分。使用menuSeparator控件在菜单元素之间放置分隔条
分隔线（separator）	在任何控件组里提供分隔线
拆分按钮 (splitButton)	创建具有默认操作和一系列选择性选项的按钮。拆分按钮最好的示例之一是“开始”选项卡的“剪贴板”组中的“粘贴”按钮。必须包括button或toggleButton控件为默认控件。在menu控件里出现可选的操作，这里可以添加button或toggleButton控件
切换按钮 (toggleButton)	提供checkBox和button控件的组合。用户通过单击toggleButton选择某状态和执行某操作

group下面可以容纳16种控件，各个控件有相同的属性，也有不同的属性，限于篇幅，下面着重介绍button按钮控件的属性及用法，其他控件在后续的章节中陆续提及。

button也具有id（idMso）、visible、label等常见属性。通过下面简短的XML代码我们进一步理解其他常用属性。

```

<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon startFromScratch="false">
    <tabs>
      <tab id="exampleID1" label="RibbonXmlEditor">
        <group id="GroupID2" label="Author:ryueifu">
          <button id="ButtonID3" label="常规按钮" imageMso="ChartTypeOther
InsertGallery" onAction="button_OA"/>
          <button id="ButtonID4" label="大按钮" size="large"
imageMso="Call" supertip="supertip:ABC" screentip="screentip:DEF"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>

```

效果如图6.11所示。

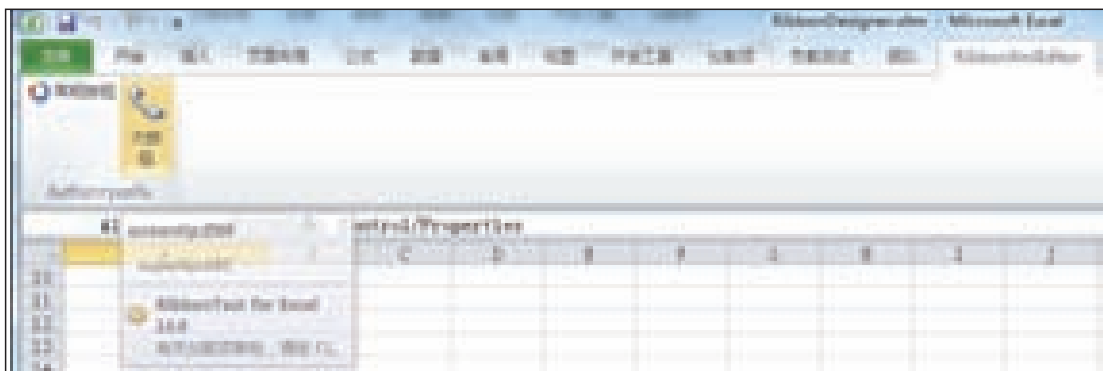


图6.11 制作功能区按钮

在上述XML代码中，button的imageMso属性用于指定按钮的图标，微软提供了7345个内置图标，每一个图标都有特定的名称，例如“Call”就表示一个电话的图形。

注意 读者可以下载名为LargeImage.rar的文件，该文件可以查询所有内置图标。

控件的supertip和screentip属性用于指定当用户把鼠标悬浮在该控件时，出现的浮动提示文字。

size属性：功能区中的控件一般是纵向最多排列3个控件，这三个控件都是常规大小的控件，如果把控件的size属性指定为large，则该控件单独占据一行。

button元素最重要的一个属性是onAction。因为CustomUI的一个目的就是要把用户的功能放置到Office界面中，因此定制按钮的作用就是用来调用加载项中的命令过程，上述XML中提到的常规按钮，onAction为“button_OA”，这就是说，当用户单击了该按钮时，会调用名为“button_OA”的宏过程，具体与制作加载项的过程有关，可以调用VBA中的过程，也可以调用其他封装程序里的过程。这部分细节，请看后面提到的处理自定义按钮的回调部分。

6.4 VSTO中使用功能区可视化设计器

VSTO提供了两种CustomUI的方式：

- 使用功能区可视化设计器。
- 使用XML。

功能区可视化设计器，就好像在窗体中设计控件那么简便，开发者不需要理解XML，就可以设计出自定义界面。

📺 本节视频：使用Ribbon设计器自定义Office功能区.wmv

下面基于5.3节创建的第一个Excel外接程序，增加自定义界面。

第1步：在Visual Studio中打开名为“ExcelAddIn20160514”的解决方案。

第2步：单击菜单【项目/添加新项】，出现如图6.12所示的对话框。

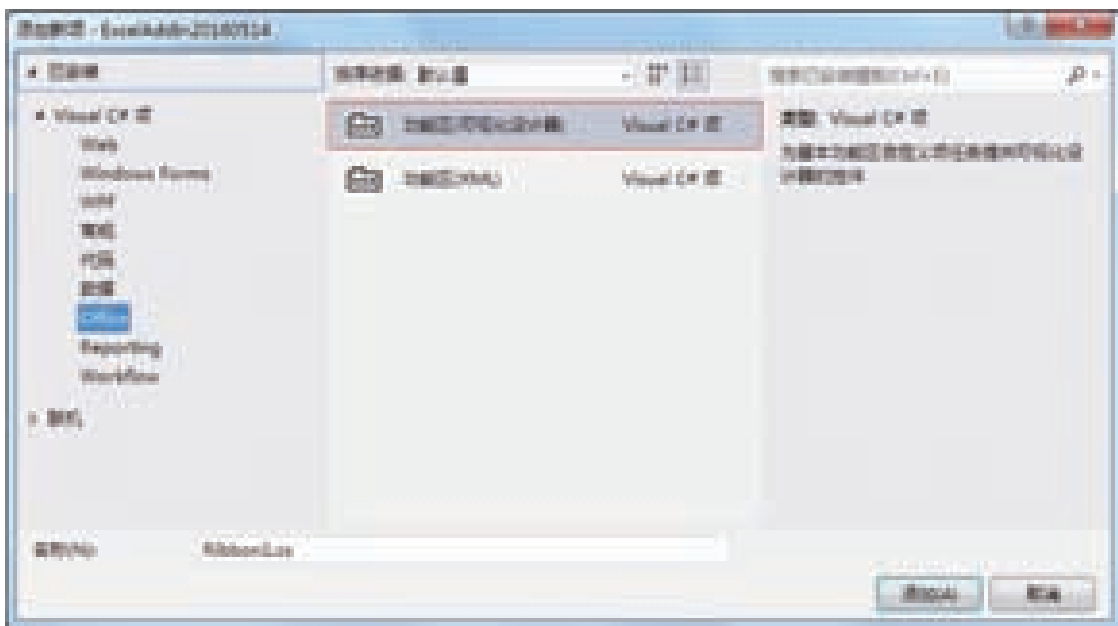


图6.12 使用功能区可视化设计器

第3步：依次单击【Visual C#项/Office】，右侧选择“功能区（可视化设计器）”。

第4步：在最下面的名称对话框中输入“Ribbon1.cs”，单击“添加”按钮。稍后会出现如图6.13所示的功能区设计界面。

鼠标选中功能区设计器的tab1选项卡在右方的属性窗口中，需要修改如下属性：

展开属性中的Position节点，PositionType规定为BeforeOfficeId，OfficeId中填写TabInsert，意思是自定义选项卡出现在Excel的“插入选项卡”之前。

另一个重要的属性是ControlIdType，默认是Office，此时我们修改为Custom。

最下面的外观，修改Label属性为“自定义选项卡”。

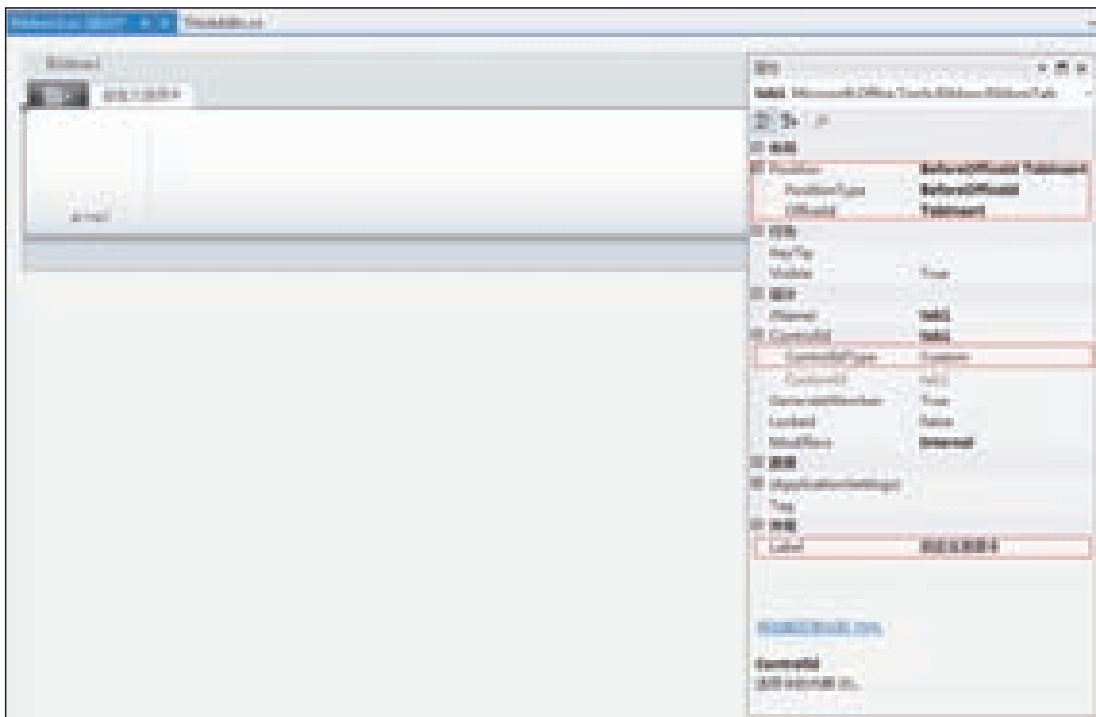


图6.13 功能区可视化设计器设计视图

类似地，我们选中功能区设计器中的group1，修改若干属性。然后单击左侧的控件工具箱中的【Office功能区控件/button】，向group1中拖放一个按钮进来，然后在属性窗格中修改button1的属性，如图6.14所示。

主要属性修改如下：

ControlSize属性修改为RibbonControlSizeLarge，这个属性等价于button的size属性。意思是以大按钮的形式出现。

Label属性修改为“第一个按钮”。

OfficeImageId属性修改为“Call”，出现一个电话的图标，这个属性等价于imageMso。

重要提示 默认情况下，功能区设计器提供了一个自定义选项卡，里面包含了一个自定义组；实际上根据需要如果需要多个自定义选项卡或多个组，鼠标再次单击左侧的工具箱，拖动tab或group到设计区即可实现。

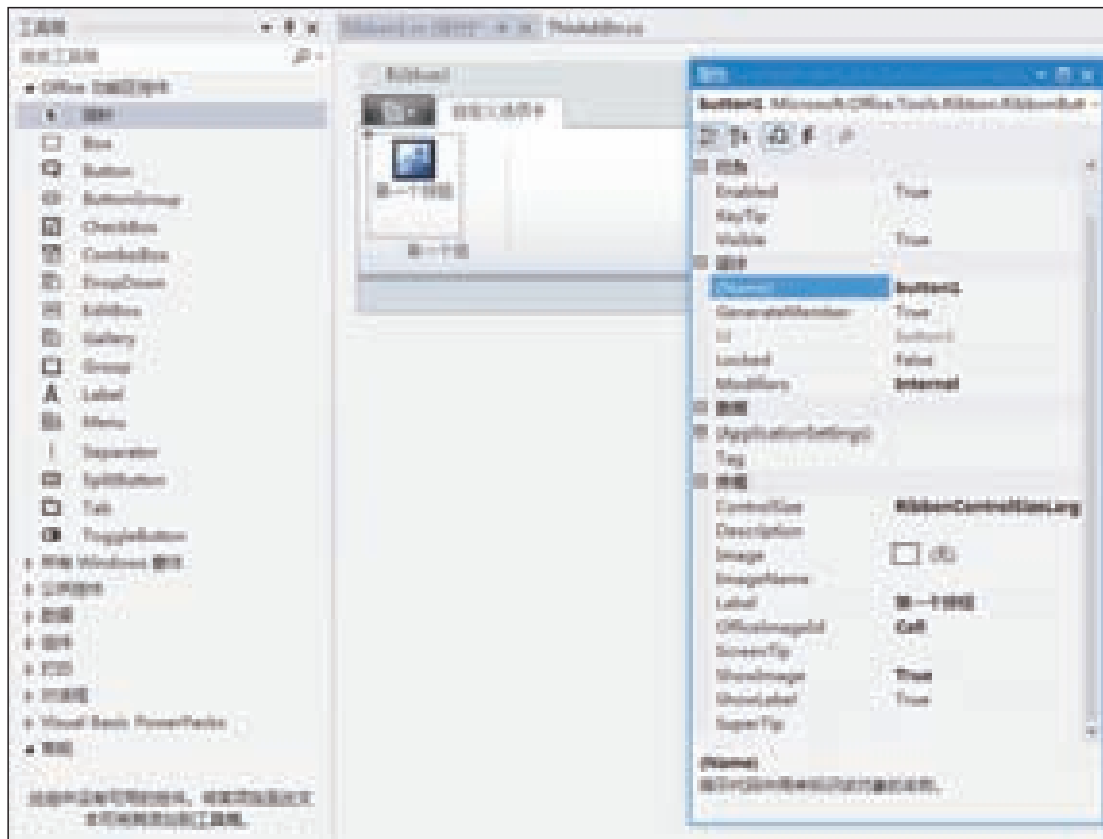


图6.14 修改功能区控件属性

6.4.1 为按钮指定回调过程

上面所述的新按钮没有任何事件过程，也就是说，即使目前启动了调试，单击该按钮也不会有任何响应。为此，在设计视图中选中该按钮，在属性窗口中切换到事件过程，双击Click即可自动进入按钮的单击事件代码区，这一点类似于C#窗体中按钮控件的单击事件过程。

Ribbon1.cs模块代码修改为：

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using Microsoft.Office.Tools.Ribbon;
6 using Excel = Microsoft.Office.Interop.Excel; //using Excel
7 namespace ExcelAddIn20160514
8 {
9     public partial class Ribbon1
```

```

10     {
11
12         private void Ribbon1_Load(object sender, RibbonUIEventArgs e)
13         {
14
15         }
16
17         private void button1_Click(object sender, RibbonControlEventArgs e)
18         {
19             Excel.Range rng;//声明一个Excel的单元格区域变量
20             rng = Globals.ThisAddIn.Application.Selection;//获得Excel所选区域
21             rng.Interior.Color = System.Drawing.Color.Yellow;//所选区域底纹颜色
22             设置为黄色
23         }
24     }

```

注意 上述代码中凡是后面带有注释的语句都是笔者加入的。

当启动调试后，在Excel中单击新建的自定义按钮，会自动把Excel的所选区域底纹颜色变黄，这就完成了VSTO中功能区和宿主程序的交互。

■ 6.4.2 Group中加入DialogBoxLauncher

DialogBoxLauncher是指组的右下角的小箭头，例如Excel的“开始选项卡”的“字体组”，它的右下角有一个小箭头，当单击这个小箭头按钮时，会出现一个对话框。这个功能在CustomUI中也能实现。

在功能区设计器视图中，鼠标选中group1，单击该组的右上角，会看到【GroupView任务/添加DialogBoxLauncher】，单击此项，如图6.15所示。然后打开属性窗口，切换到group1的事件过程，双击DialogBoxLauncherClick，自动产生该事件过程。

在这里，当用户单击组右下角的小箭头后，在Excel中出现一个C#窗体。因此，在书写group1_DialogLauncherClick事件之前，先为项目添加一个新窗体。

在Visual Studio中，单击【项目/添加Windows窗体】，在出现的对话框中，将名称修改为“Form1.cs”，单击“添加”按钮。

然后修改Ribbon1.cs中的部分代码为：

```

        private void group1_DialogLauncherClick(object sender,
        RibbonControlEventArgs e)
        {
            Form1 fm1 = new Form1();//创建新实例
            fm1.ShowDialog();//以对话框的形式，在Excel中显示Form1
        }

```

此时，按下【F5】键启动调试，会在Excel中看到新的用户选项卡，如图6.16所示。

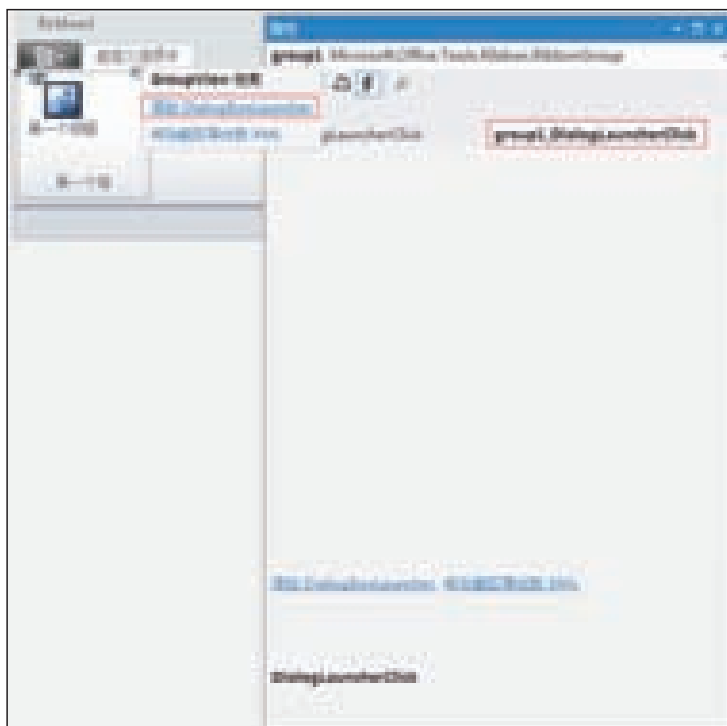


图6.15 为Group添加DialogLauncherClick

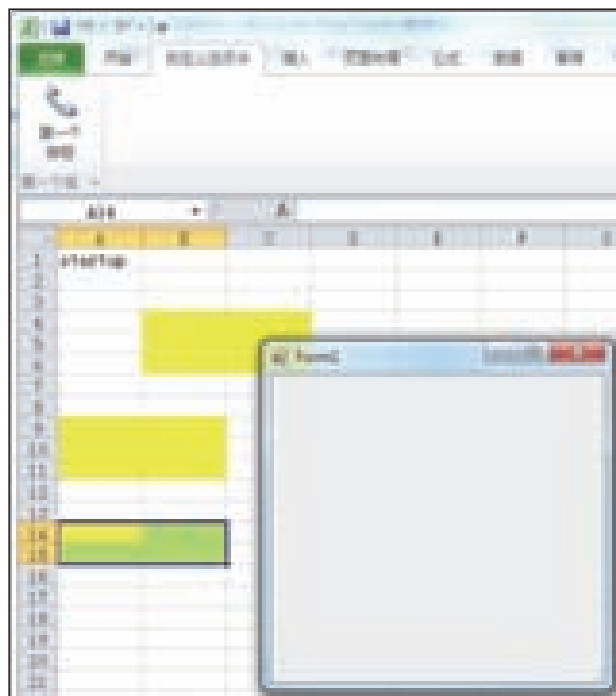


图6.16 DialogLauncher效果

单击“第一个按钮”，Excel所选区域会变黄；单击组右下角的小箭头，会出现一个C#窗体。测试完毕后，关闭C#窗体，完全退出Excel。

使用功能区可视化设计器，开发者不需要写任何XML代码即可产生界面，设计过程和C#窗体控件的设计思路几乎一样，大大减轻了开发者的难度。但是它的缺点是自定义的范围太窄，使用这项技术只能自定义常用选项卡部分。而至于backstage，还有对象右键菜单是无法在这里自定义的。为此，我们有必要进一步学习更全面的自定义知识：使用XML进行CustomUI定制。

6.5 使用XML进行CustomUI定制

在VSTO开发中，可以使用纯粹的XML代码来定制Office界面。为了便于演示，重新创建一个名为“ExcelAddIn20160515”的Excel外接程序的空白项目。创建过程和方法请读者参考前面所述。

📺 本节视频：使用XML自定义Office功能区.wmv

第1步：在Visual Studio中打开名为“ExcelAddIn20160515”的解决方案。

第2步：单击菜单【项目/添加新项】，出现“添加新项”对话框，如图6.17所示。

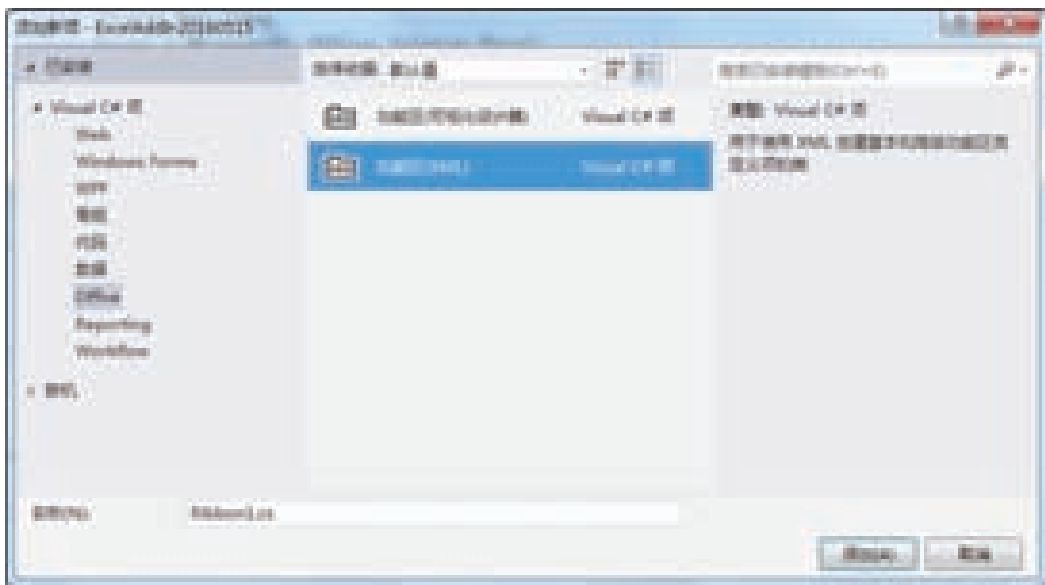


图6.17 使用XML

第3步：左侧依次选择【已安装/C#项/Office】，右侧选择“功能区（XML）”，名称更改为“Ribbon1.cs”，单击“添加”按钮。自动打开的Ribbon1.cs类模块，如图6.18所示。

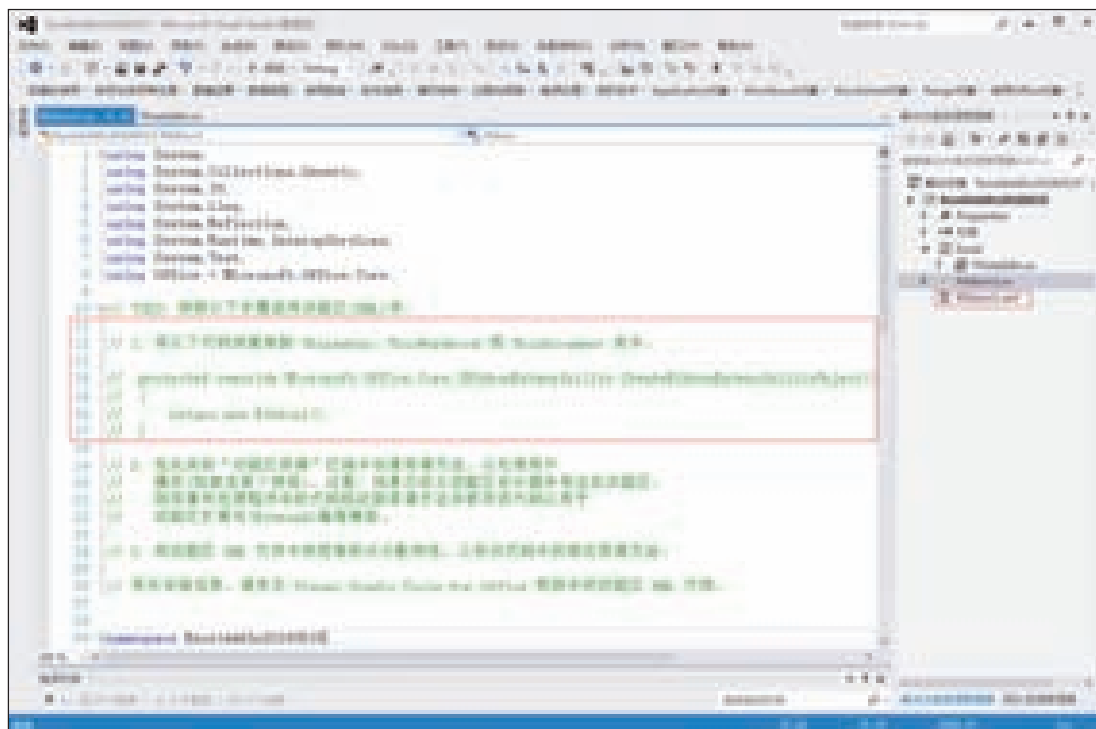


图6.18 XML操作指示

第4步：根据提示，把图6.18中框中的代码复制到ThisAddIn.cs中，取消其注释。

修改后的ThisAddIn.cs为：

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Xml.Linq;
6  using Excel = Microsoft.Office.Interop.Excel;
7  using Office = Microsoft.Office.Core;
8  using Microsoft.Office.Tools.Excel;
9
10 namespace ExcelAddIn20160515
11 {
12     public partial class ThisAddIn
13     {
14         private void ThisAddIn_Startup(object sender, System.EventArgs e)
15         {
16         }
17
18         private void ThisAddIn_Shutdown(object sender, System.EventArgs e)
19         {
20         }
21

```

```

22         protected override Microsoft.Office.Core.IRibbonExtensibility CreateRi
bbonExtensibilityObject()
23         { //该过程用于启用XML
24             return new Ribbon1();
25         }
26     }
27 }

```

第5步：双击解决方案资源管理器中的Ribbon1.xml，编辑用于CustomUI的XML代码，如图6.19所示。

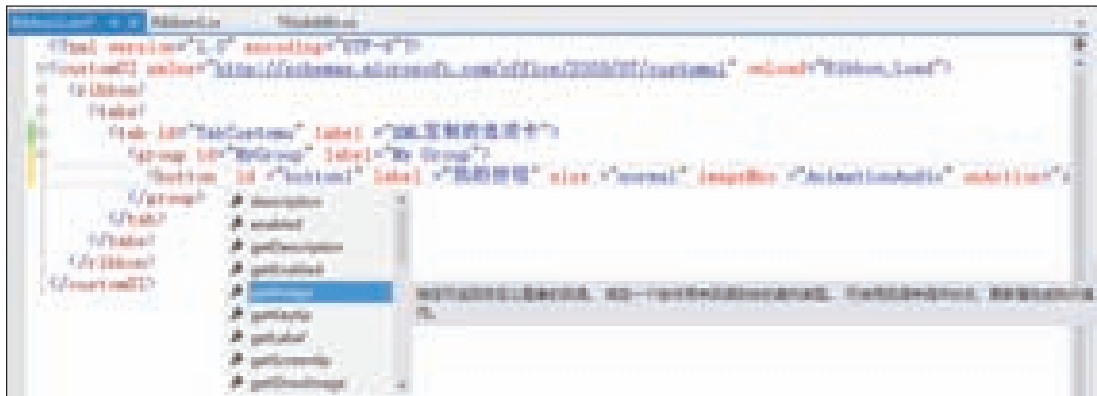


图6.19 编辑XML

在VSTO中书写XML的时候，会自动列出相关属性。当然，书写XML这一环节，读者也可以使用前面所述的使用Ribbon XML Editor事先写好，粘贴到这里。

实例中完整的XML代码为：

```

<?xml version="1.0" encoding="UTF-8"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
onLoad="Ribbon_Load">
  <ribbon>
    <tabs>
      <tab id="TabCustomu" label ="XML定制的选项卡">
        <group id="MyGroup" label="My Group">
          <button id ="button1" label ="我的按钮" size ="normal" imageMso
="AnimationAudio" onAction="callback"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>

```

注意到在XML中button的onAction属性指定为“callback”，为此，必须在Ribbon1.cs模块中添加该过程，作为单击了该按钮的回调。

修改后Ribbon1.cs中的部分代码为：

```
1      #region 功能区回调
2      //在此创建回调方法。有关添加回调方法的详细信息，请在解决方案资源管理器中选择功能区
XML 项，然后按 F1
3
4      public void Ribbon_Load(Office.IRibbonUI ribbonUI)
5      {
6          this.ribbon = ribbonUI;
7      }
8      public void callback(Office.IRibbonControl control)
9      {
10         Globals.ThisAddIn.Application.ActiveCell.Value = System.DateTime.
Now; //Excel活动单元格等于系统时间
11     }
12     #endregion
```

注意 上述代码中，`public void callback(Office.IRibbonControl control)` 这个按钮的单击过程写法，需要参考微软提供的“ribbon回调函数大全.xlsm”，该文档包含了所有功能区控件的回调函数格式，涉及VBA、VB6以及C#的回调函数写法。

启动调试，在出现的自定义界面中，单击“我的按钮”，会在Excel活动单元格中输入当前时间，效果图略。

本章要点回顾

- C#不仅能操作和控制Office对象，还可以使用VSTO更改Office的界面，添加自定义界面到Office中。本章重点介绍了自定义Office功能区的技术。
- CustomUI使用XML语言编写界面，应重点掌握自定义常用功能区中，选项卡、组、控件之间的所属关系。
- 通过使用功能区可视化设计器，可以像设计窗体一样来设计自定义功能区。
- 功能区可视化设计器虽然方便、直观，但是它能定制的区域有限，只能定制常用功能区部分。如果要改变Office的右键菜单或者backstage视图，就需要使用更全面的功能：使用XML进行CustomUI定制。

第7章

自定义任务窗格

任务窗格是通常停靠在Office应用程序中某一窗口一侧的用户界面面板。例如，Word中的“样式和格式”任务窗格，以及Office 2010的“剪贴板”窗格。自定义任务窗格（Custom Task Panes, CTP）的行为类似于Office中的内置任务窗格。任务窗格最大的优点是窗格和文档处于同一层面，融为一体，而不像窗体浮动在文档之上。用户可以将自定义任务窗格停靠在应用程序窗口的不同侧，或者可以将自定义任务窗格拖动到窗口中的任何位置。通过VSTO可以创建一个外接程序，使之同时显示多个自定义任务窗格，而且用户可以分别控制每个任务窗格。

 本章视频：自定义任务窗格的设计.wmv

7.1 任务窗格行为控制

无论是内置窗格，还是自定义任务窗格，当用户单击窗格右上角的菜单时，会看到菜单中有“移动”、“大小”和“关闭”三个菜单项，如图7.1所示。

实际上，用户可以用鼠标拖曳任务窗格标题栏，拖放窗格到文档中的合适位置，任务窗格有5种停靠方式：

- 顶部
- 底部
- 左侧
- 右侧
- 悬浮

将鼠标放在窗格四周的边缘处，像改变窗体大小一样，改变窗格的高度和宽度。

特别注意的是，用户单击窗格右上角的“关闭”按钮，就看不到这个窗格了，实际上

这个窗格依然没有卸载掉。

注意 单击任务窗格右上角的“关闭”按钮，只是隐藏了任务窗格。

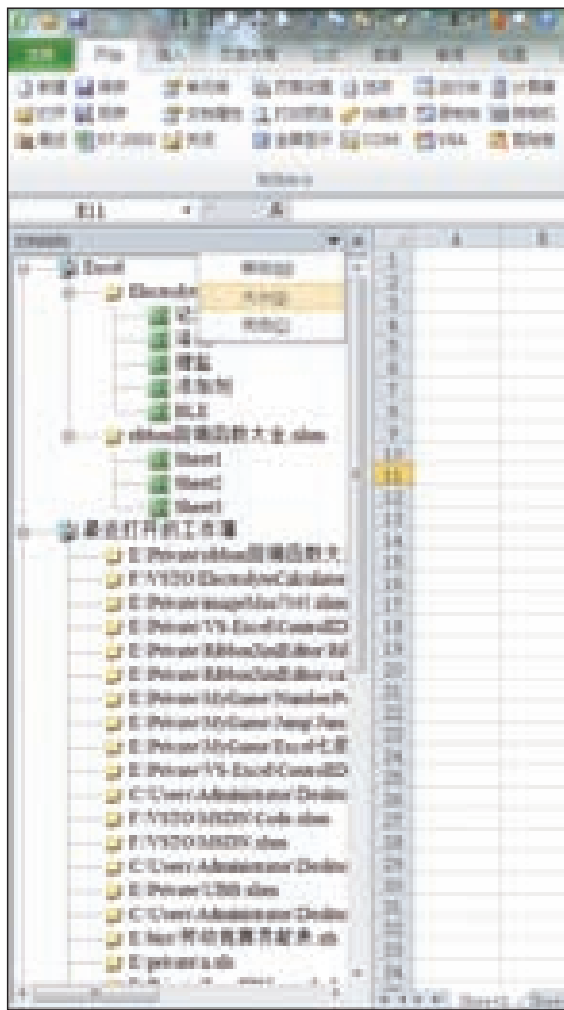


图7.1 任务窗格样图

7.2 VSTO外接程序项目中添加任务窗格

和第6章讲过的CustomUI类似，创建自定义任务窗格也是VSTO开发中非常重要的一个课题。通过创建自定义任务窗格，用户可以使用任务窗格中的控件和Office文档交互。VSTO开发自定义任务窗格的过程中，涉及的对象比较多，主要涉及功能区、任务窗格、Office对象三者的关系，彼此互相调用的场合也比较多，为此，开发过程中会用到静态类，使用静态类存储公用变量。

下面通过一个实例演示，为Excel 2010中创建一个自定义任务窗格“工作表导航”。目的是当显示窗格时，窗格中列出当前所有的工作表名称，当用户单击任何一个名称时，Excel会自动切换到该工作表。

■ 7.2.1 创建Excel 2010外接程序

在Visual Studio 2012中，重新创建一个名为“ExcelAddIn20160516”的用于Excel 2010的外接程序的空白项目。

■ 7.2.2 添加用户控件

在VSTO项目中，单击菜单【项目/添加用户控件】，在打开的“添加新项”对话框中的“名称”文本框中输入“UserControl1.cs”，如图7.2所示。



图7.2 添加用户控件

为什么要添加用户控件呢？这是因为实际上每一个任务窗格都是一个用户控件，用户控件类似于一个没有标题栏的C#窗体，换句话说，用户控件设计成什么样，将来的任务窗格就是什么样。

进入UserControl1的设计视图，鼠标从Visual Studio的控件工具箱中拖动一个“ListBox”列表框控件到UserControl1中。再拖动一个“Button”按钮到列表框控件的下

方，在属性窗口中把按钮的标题文字更改为“更新”。

■ 7.2.3 静态类中声明任务窗格对象

为了便于以后在代码中操作控制任务窗格，任务窗格的对象变量应放在公共的位置。为此，为项目添加一个名为“Share.cs”的类，如图7.3所示。

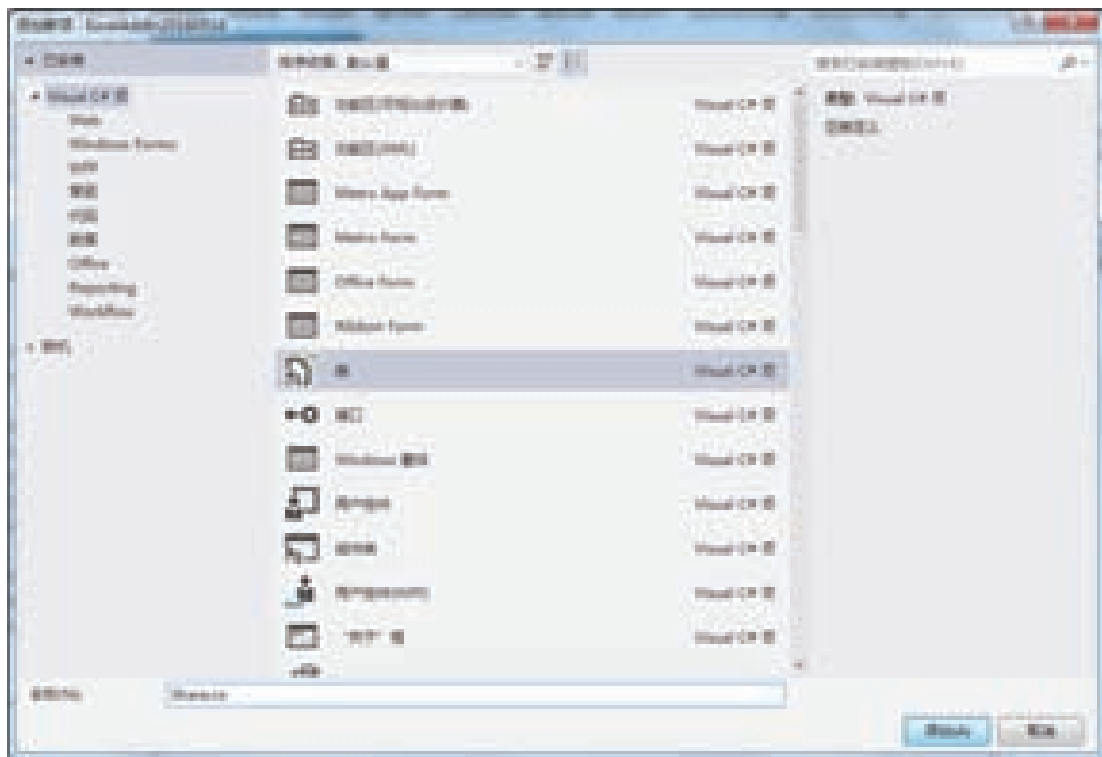


图7.3 使用静态类

为了让这个类成为静态类，需要修改Share.cs的代码为：

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5
6 namespace ExcelAddIn20160516
7 {
8     public static class Share //使之成为静态类
9     {
10         public static Microsoft.Office.Tools.CustomTaskPane task1; //声明任务窗
11         格对象
12     }
13 }
```

注意上面的代码，无论是类名称还是声明的变量前面，都带有public static。代码中的task1就是用于Office的任务窗格变量，今后我们通过操作Share.task1这个变量就可以控制任务窗格。

■ 7.2.4 创建并显示任务窗格

在VSTO项目中，切换到ThisAddIn.cs代码视图中，ThisAddIn_Startup事件过程代码修改为：

```
1     private void ThisAddIn_Startup(object sender, System.EventArgs e)
2     {
3         UserControl1 uc1 = new UserControl1();
4         Share.task1 = Globals.ThisAddIn.CustomTaskPanes.Add(uc1, "工作表导航");
5         Share.task1.Visible = true;
6     }
```

代码的含义是，当加载项加载时立即在Excel中显示自定义任务窗格。uc1这个变量是UserControl1用户控件的一个新实例，task1是基于uc1创建的自定义任务窗格，该窗格的标题文字为“工作表导航”。

Globals.ThisAddIn.CustomTaskPanes.Add 这一句是产生自定义任务窗格的关键代码。

此时，启动调试，会在Excel的右侧看到该窗格，如图7.4所示。

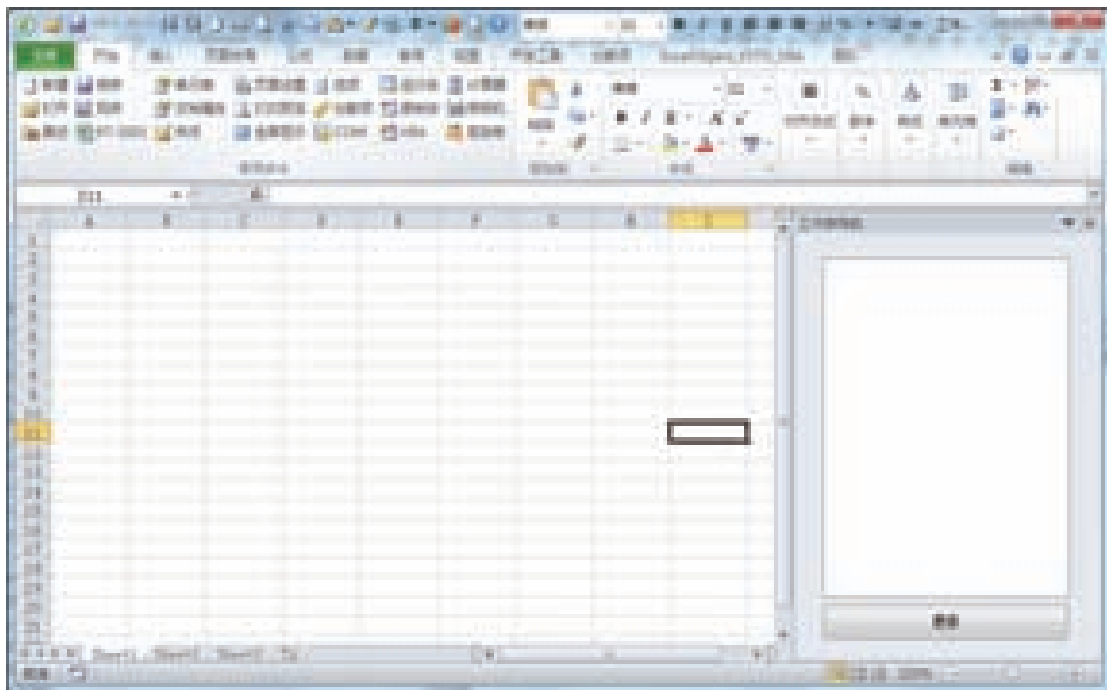


图7.4 任务窗格效果图

至此，窗格的基本形态已经做出，接下来的任务是，让窗格中的控件和Excel产生交互。接下来终止调试，为任务窗格中的“更新”按钮以及列表框控件赋予实际的事件。

在VSTO切换到UserControl1的设计视图，双击“更新”按钮，自动切换到UserControl1的代码视图；同样，双击列表框控件，自动创建列表框的单击事件过程，适当补充代码后，UserControl1.cs完整代码如下：

```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Drawing;
5 using System.Data;
6 using System.Linq;
7 using System.Text;
8 using System.Windows.Forms;
9 using Excel = Microsoft.Office.Interop.Excel; //using Excel
10 namespace ExcelAddIn20160516
11 {
12     public partial class UserControl1 : UserControl
13     {
14         public UserControl1()
15         {
16             InitializeComponent();
17         }
18
19         private void button1_Click(object sender, EventArgs e)
20         {
21             this.listBox1.Items.Clear(); //事先清空列表框所有项
22             foreach (Excel.Worksheet wst in Globals.ThisAddIn.Application.
ActiveWorkbook.Worksheets)
23             {
24                 this.listBox1.Items.Add(wst.Name); //遍历每一个工作表，把表名加入列表框
25             }
26         }
27
28         private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
29         {
30             Globals.ThisAddIn.Application.ActiveWorkbook.Worksheets[this.
listBox1.Text].Activate();
31             //列表框的单击事件：单击任一条目，Excel自动激活相应工作表
32         }
33     }
34 }
```

注意 以上代码，重点关注button1_Click以及listBox1_SelectedIndexChanged过程。

修改代码完毕后，启动调试，在Excel的自定义任务窗格中单击“更新”按钮后，列表框中自动列出所有表名，单击任一表名，自动切换到该表，如图7.5所示。

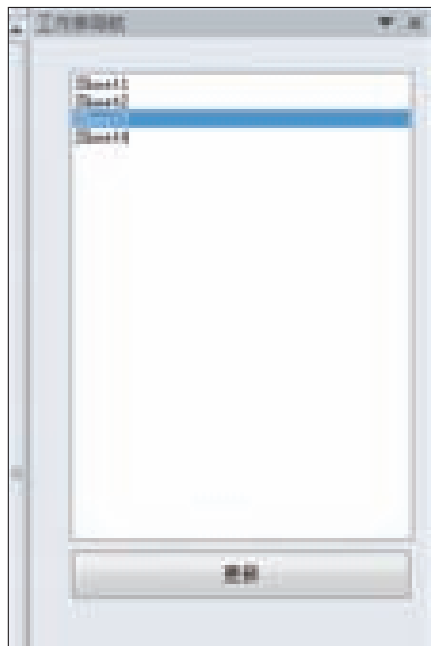


图7.5 工作表导航窗格

到此为止，可以说一个完整的自定义任务窗格已经设计完毕。但是还有一些相关的问题需要进一步解决。例如，当用户手动关闭了任务窗格（实际上是隐藏了任务窗格），在不卸载加载项的场合下如何让任务窗格重现？如何使用代码让任务窗格自动停靠到其他位置？当任务窗格的显示/隐藏属性改变时，能否引发事件？这几个问题接下来会一起解决。

7.3 功能区与任务窗格的交互控制

为了让用户更好地操作任务窗格，一般在设计任务窗格的时候，同时带有自定义功能区，用以控制任务窗格。

7.3.1 利用功能区切换按钮控制任务窗格的显示隐藏

接下来我们结合第6章讲过的内容，为该VSTO项目添加一个功能区可视化设计器，设计一个自定义选项卡，里面放置一个toggleButton，用来切换显示任务窗格，再放置5个button控件，让任务窗格分别停靠在不同的位置。功能区添加的所有控件列于表7.1中。

表7.1 实例中功能区中添加的控件及部分属性设定

控件名称	标题文字	重要属性更改
tab1	任务窗格选项卡	ControlIdType: Custom
group1	可见性	

控件名称	标题文字	重要属性更改
toggleButton1	显示/隐藏	ControlSize: RibbonControlSizeLarge
group2	停靠位置	
button1	靠左	OfficeImageId: L
button2	靠右	OfficeImageId: R
button3	靠上	OfficeImageId: U
button4	靠下	OfficeImageId: D
button5	浮动	OfficeImageId: F

然后，鼠标双击toggleButton1进入事件过程代码区，类似地，依次双击button1至button5，编辑这些控件的回调过程，代码如下：

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using Microsoft.Office.Tools.Ribbon;
6
7  namespace ExcelAddIn20160516
8  {
9      public partial class Ribbon1
10     {
11         private void Ribbon1_Load(object sender, RibbonUIEventArgs e)
12         {
13         }
14     }
15
16     private void toggleButton1_Click(object sender, RibbonControlEventArgs
17     e)
18     {
19         Share.task1.Visible = this.toggleButton1.Checked;
20     }
21
22     private void button1_Click(object sender, RibbonControlEventArgs e)
23     {
24         //靠左
25         Share.task1.DockPosition = Microsoft.Office.Core.
26         MsoCTPDockPosition.msoCTPDockPositionLeft;
27     }
28
29     private void button2_Click(object sender, RibbonControlEventArgs e)
30     {
31         //靠右
32         Share.task1.DockPosition = Microsoft.Office.Core.
33         MsoCTPDockPosition.msoCTPDockPositionRight;
34     }
35 }

```

```

33     private void button3_Click(object sender, RibbonControlEventArgs e)
34     {
35         //靠上
36         Share.task1.DockPosition = Microsoft.Office.Core.
MsoCTPDockPosition.msoCTPDockPositionTop;
37     }
38
39     private void button4_Click(object sender, RibbonControlEventArgs e)
40     {
41         //靠下
42         Share.task1.DockPosition = Microsoft.Office.Core.
MsoCTPDockPosition.msoCTPDockPositionBottom;
43     }
44
45     private void button5_Click(object sender, RibbonControlEventArgs e)
46     {
47         //浮动
48         Share.task1.DockPosition = Microsoft.Office.Core.
MsoCTPDockPosition.msoCTPDockPositionFloating;
49     }
50 }
51 }

```

注意这一句“Share.task1.Visible = this.toggleButton1.Checked;”，它的含义是说任务窗格的可见性等于功能区中切换按钮toggleButton1的勾选。换言之，当鼠标勾选该切换按钮时，显示任务窗格；取消勾选，则隐藏窗格。

另外，在button1_Click事件代码中Share.task1.DockPosition是指窗格的停靠位置属性。Microsoft.Office.Core.MsoCTPDockPosition.msoCTPDockPositionLeft是枚举型常量，意思是任务窗格停靠在工作表左侧。

修改代码后，启动调试，在Excel中同时看到自定义功能区和自定义任务窗格，如图7.6所示。

当用户单击功能区左侧切换按钮时，任务窗格自动显示/隐藏。当单击停靠位置中的按钮时，任务窗格自动调整停靠位置。

注意 本例为便于演示，使用了功能区可视化设计器，读者也可以根据需要使用XML定制功能区，效果相同。

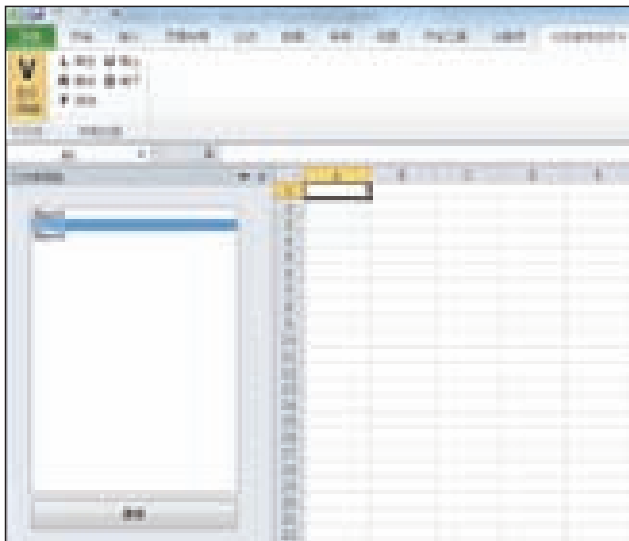


图7.6 任务窗格属性控制

7.3.2 处理自定义任务窗格事件

读者不难发现，上节示例中，当用户手动单击任务窗格右上角的“关闭”按钮隐藏任务窗格之后，功能区的切换按钮没有任何感知，还处于勾选状态，这就让人困惑了。为此介绍一下自定义任务窗格的visibleChanged事件。通过使用该事件使得手动关闭任务窗格时同步功能区切换按钮的状态。

1. 任务窗格可见性发生变化时引发的事件

任务窗格的visibleChanged事件，含义是当任务窗格的visible属性被重设为true或false的时候，会引发事件过程。基于上面讲过的项目，我们需要在加载项启动事件中，为任务窗格赋予该事件，为此修改ThisAddIn_Startup事件中的部分代码为：

```
1      private void ThisAddIn_Startup(object sender, System.EventArgs e)
2      {
3          UserControl1 uc1 = new UserControl1();
4          Share.task1 = Globals.ThisAddIn.CustomTaskPanels.Add(uc1, "工作表导航");
5          Share.task1.Visible = true;
6          Share.task1.VisibleChanged += new EventHandler(task1_
visibleChanged); //为任务窗格创建事件
7      }
8      private void task1_visibleChanged(object sender, System.EventArgs e)
9      {
10         //对应的事件过程
11         Ribbon1 ribbon =Globals.Ribbon1>(); //获得功能区
可视化设计器中的Ribbon
12         ribbon.toggleButton1.Checked = Share.task1.Visible;
13     }
14     private void ThisAddIn_Shutdown(object sender, System.EventArgs e)
15     {
16     }
```

“Share.task1.VisibleChanged +=”这一句，其意是增加任务窗格的可见性改变事件过程，对应下面的task1_visibleChanged事件过程，让功能区的切换按钮的勾选状态等于任务窗格的可见性。这样，当用户隐藏掉任务窗格的时候，功能区的切换按钮会自动调整为不勾选。

2. 任务窗格停靠位置发生变化时引发的事件

任务窗格还有另外一个事件：DockPositionChanged事件，该事件是指当使用代码或手工改变了任务窗格的停靠位置时引发的事件。与visibleChanged事件同样的做法，修改ThisAddIn.cs中的部分代码为：

```
1      private void ThisAddIn_Startup(object sender, System.EventArgs e)
2      {
3          UserControl1 uc1 = new UserControl1();
4          Share.task1 = Globals.ThisAddIn.CustomTaskPanels.Add(uc1, "工作表导航");
```

```
5         Share.task1.Visible = true;
6         Share.task1.VisibleChanged += new EventHandler(task1_
visibleChanged); //为任务窗格创建事件
7         Share.task1.DockPositionChanged += new EventHandler(task1_
DockPositionChanged);
8     }
9     private void task1_visibleChanged(object sender, System.EventArgs e)
10    {
11        //对应的事件过程
12        Ribbon1 ribbon =Globals.Ribbon1.GetRibbon<Ribbon1>(); //获得功能区
可视化设计器中的Ribbon
13        ribbon.toggleButton1.Checked = Share.task1.Visible;
14    }
15    private void task1_DockPositionChanged(object sender, System.
EventArgs e)
16    {
17        Globals.ThisAddIn.Application.StatusBar = Share.task1.
DockPosition.ToString();
18    }
19    private void ThisAddIn_Shutdown(object sender, System.EventArgs e)
20    {
21    }
```

当用户改变任务窗格停靠位置时，Excel的状态栏中会显示出停靠后的位置描述。

■ 7.3.3 完全卸载任务窗格

一般情况下，当VSTO创建的Office外接程序加载时，显示任务窗格，当卸载外接程序的时候，会自动卸载任务窗格，用户单击任务窗格右上角的“关闭”按钮，只是起到了隐藏的作用。如果中途需要卸载任务窗格，需要使用Dispose方法，例如Share.task1.Dispose()就卸载了该任务窗格。

注意 一个Office外接程序中，不限于只有一个自定义任务窗格，开发者可以根据需要增加多个任务窗格。

本章要点回顾

- 自定义任务窗格的实质是把一个UserControl植入Office界面中。
- 使用Globals.ThisAddIn.CustomTaskPanels.Add(new UserControl,Caption)方法增加一个任务窗格。
- 任务窗格的Visible属性控制任务窗格的显示/隐藏。
- 任务窗格的DockPosition属性控制任务窗格的停靠位置。
- 任务窗格的两个重要事件：VisibleChanged和DockPositionChanged，用来判断任务窗格的状态。

第8章

自定义工具栏

Office 2003及其以下版本一律采用工具栏（Commandbar）系统呈现界面和功能。开发者既可以对内置工具栏进行修改维护，同时也可以创建全新的自定义工具栏。Office 2007以上版本，虽然仍旧可以继续对工具栏对象进行操作，但是所有的自定义项目仅出现在“加载项”选项卡。

而对于高级Office版本的VSTO开发，创建自定义功能区以及任务窗格就足够了；但是有些终端用户仍然使用Office 2003。换言之，Office 2003是无法使用自定义功能区的，本章为了满足更大范围的开发者，详细介绍在VSTO中操作和控制Office工具栏对象。

8.1 Office工具栏对象简述

8.1.1 Commandbar对象

Office的Application有一个Commandbars对象集合，对于每一个Commandbar里面包含着若干控件，例如Application.Commandbars(“Standard”)，就表示Excel的“常用”工具栏。

而对于工具栏中的每一个控件，可以使用控件的Caption标题属性读取到，例如Application.Commandbars(“常用”).Controls(“打开”)，则表示常用工具栏中的“打开”按钮。

1. 工具栏的类型

工具栏的类型可以通过Commandbar.Type属性读取到，具体分为：

- 菜单栏
- 一般工具栏

- 快捷菜单

所谓的快捷菜单，就是指各种对象的右键菜单，例如鼠标右击工作表的行或者列，或者右击Excel工作表标签，会出现不同的菜单，其实这些菜单也都属于工具栏对象。例如，Application.Commandbars("Ply")表示工作表标签的右键菜单。

2. 工具栏的重要属性

Commandbar有诸多属性，但是最常用的有以下几个：

- Name：工具栏的名称。
- Builtin：是否内置菜单，如果是用户自定义的工具栏，则为False。
- Visible：是否可见。

3. 工具栏的重要方法

工具栏的重要方法有以下几个：

- Reset：重置工具栏，让工具栏恢复为出厂状态。只有内置工具栏可以重置。
- Delete：删除工具栏，只有自定义工具栏才可以删除，同时自动删除该工具栏中所有控件。
- Controls.Add：增加控件。

■ 8.1.2 CommandbarControl对象

CommandbarControl泛指工具栏中所有类型的控件，常见的工具栏控件类型有：

- CommandBarButton：命令按钮。
- CommandbarCombo：组合框。
- CommandbarPopup：弹出式子菜单。

可以看出工具栏是控件的容器，其中CommandbarPopup是一种可以容纳其他控件的控件，如果要建立子菜单，则需要先添加这种控件，然后往子菜单中进一步添加其他相关控件。

工具栏控件的种类很多，每一种控件的属性不尽相同，这里着重介绍最常用的CommandBarButton控件。

1. CommandBarButton的重要属性

CommandBarButton的重要属性如下：

- Caption：标题文字。
- Visible：可见性。
- FaceID：图标编号。
- Style：按钮样式。

- OnAction: 回调函数。

2. CommandBarButton的重要方法

CommandBarButton的重要方法如下:

- Copy: 复制控件。
- Move: 移动控件。
- Delete: 删除控件。

8.1.3 自定义工具栏的作用和意义

可以把Office现有的工具栏体系进行适当改动,也可以添加自定义工具栏和控件。自定义工具栏和按钮最大的作用是把外接程序里的过程和命令委托给工具栏控件,从而能在Office界面中执行外接程序中的命令。

8.2 VSTO实现自定义工具栏

VSTO中的自定义工具栏技术不需要引入任何外部的对象,只需要创建一个空白的Excel 2010外接程序,在外接程序的启动事件过程中加入创建工具栏的部分代码,在外接程序的卸载事件过程中清除自定义的工具栏和控件即可。

下面通过一个经典实例,来逐步加深对自定义工具栏技术的理解。外接程序的目的是加载外接程序时,为Excel创建一个名为“cmb”的自定义工具栏,工具栏中放入“上下左右”4个导航按钮,再放入一个子菜单控件,该子菜单控件里再加两个按钮。4个导航按钮的作用是,当单击其中一个按钮时,Excel工作表中所选区域向该方向移动一个单位。而子菜单中的两个按钮用于显示和隐藏当前工作表的网格线。

此外,为了演示自定义右键菜单,外接程序还为工作表标签的右键菜单增加一个名为“显隐标题栏”的按钮。当单击该按钮时,当前工作表的行号和列标进行显示和隐藏的切换。

8.2.1 创建自定义工具栏

启动Visual Studio,重新创建一个名为“ExcelAddIn20160517”的Excel 2010外接程序。修改ThisAddIn.cs中的外接程序启动事件代码为:

```
1      Excel.Application ExcelApp;  
2      private void ThisAddIn_Startup(object sender, System.EventArgs e)  
3      {  
4          ExcelApp = this.Application; //获取ExcelApplication对象  
5          CreateMyBar();  
6          AddControlToPly();  
7      }
```

由于创建新工具栏的代码比较长，因此在启动过程中调用了CreateMyBar这个子过程，接着调用AddControlToPly过程，后一个过程的作用是往工作表标签快捷菜单中加入控件。

CreateMyBar子过程代码如下：

```
1      public void CreateMyBar()  
2      {  
3          Office.CommandBarButton btn;  
4          try  
5          {  
6              ExcelApp.CommandBars["cmb"].Delete();  
7          }  
8          catch (System.SystemException ex)  
9          {  
10             }  
11         }  
12         Office.CommandBar cmb = ExcelApp.CommandBars.Add("cmb", Office.  
MsoBarPosition.msoBarTop, Type.Missing, true);  
13         cmb.Visible = true;  
14  
15         btn = (Office.CommandBarButton)cmb.Controls.Add(Office.MsoControlType.  
msoControlButton, Type.Missing, Type.Missing, Type.Missing, true);  
16         btn.Caption = "上";  
17         btn.FaceId = 1265;  
18         btn.Style = Office.MsoButtonStyle.msoButtonIconAndCaption;  
19         btn.Click += new  
Office._CommandBarButtonEvents_ClickEventHandler(btn_Click);  
20  
21         btn = (Office.CommandBarButton)cmb.Controls.Add(Office.MsoControlType.  
msoControlButton, Type.Missing, Type.Missing, Type.Missing, true);  
22         btn.Caption = "下";  
23         btn.FaceId = 1266;  
24         btn.Style = Office.MsoButtonStyle.msoButtonIconAndCaption;  
25         btn.Click += new Office._CommandBarButtonEvents_ClickEventHandler(btn_  
Click);  
26  
27         btn = (Office.CommandBarButton)cmb.Controls.Add(Office.MsoControlType.  
msoControlButton, Type.Missing, Type.Missing, Type.Missing, true);  
28         btn.Caption = "左";  
29         btn.FaceId = 1264;  
30         btn.Style = Office.MsoButtonStyle.msoButtonIconAndCaption;  
31         btn.Click += new Office._CommandBarButtonEvents_ClickEventHandler(btn_  
Click);  
32  
33         btn = (Office.CommandBarButton)cmb.Controls.Add(Office.MsoControlType.  
msoControlButton, Type.Missing, Type.Missing, Type.Missing, true);  
34         btn.Caption = "右";  
35         btn.FaceId = 1263;  
36         btn.Style = Office.MsoButtonStyle.msoButtonIconAndCaption;  
37         btn.Click += new Office._CommandBarButtonEvents_ClickEventHandler(btn_  
Click);
```

```

38
39         //以下创建级联菜单
40         Office.CommandBarPopup pop;
41         pop = (Office.CommandBarPopup)cmb.Controls.Add(Office.MsoControlType.
msoControlPopup, Type.Missing, Type.Missing, Type.Missing, true);
42         pop.Caption = "网格线";
43         //以下为级联菜单建立2个子项
44         btn = (Office.CommandBarButton)pop.Controls.Add(Office.MsoControlType.
msoControlButton, Type.Missing, Type.Missing, Type.Missing, true);
45         btn.Caption = "显示网格线";
46         btn.FaceId = 1265;
47         btn.Style = Office.MsoButtonStyle.msoButtonIconAndCaption;
48         btn.Click += new Office._CommandBarButtonEvents_ClickEventHandler(btn_
Click);
49
50         btn = (Office.CommandBarButton)pop.Controls.Add(Office.MsoControlType.
msoControlButton, Type.Missing, Type.Missing, Type.Missing, true);
51         btn.Caption = "隐藏网格线";
52         btn.FaceId = 1266;
53         btn.Style = Office.MsoButtonStyle.msoButtonIconAndCaption;
54         btn.Click += new Office._CommandBarButtonEvents_ClickEventHandler(btn_
Click);
55     }

```

在自定义工具栏过程中，添加工具栏或控件之前，必须先删除已经存在的同名对象，为此，在该过程一开始就先删除“cmb”工具栏，而不管它是否已经存在。

代码中的cmb变量就是指新建的工具栏，然后使用工具栏按钮对象btn不断地追加按钮，对于每一个按钮，都设置了相应的3个属性：Caption、FaceID、Style，用来定义控件的标题文字、图标以及显示样式。为了让用户单击按钮响应外接程序中的过程，必须为btn创建单击事件。

注意 笔者制作的FaceIDViewer可以查看一万个Office内置图标。

代码中的popup对象是一个级联子菜单控件，里面追加了两个按钮。

■ 8.2.2 处理工具栏按钮的回调

由于以上总共6个按钮，调用的都是btn_Click这个事件过程，为此还需要写入该过程：

```

1         public void btn_Click(Microsoft.Office.Core.CommandBarButton Ctrl, ref
bool CancelDefault)
2         {
3             Excel.Range sel = (Excel.Range)ExcelApp.Selection;
4             switch (Ctrl.Caption)
5             {
6                 case "上":
7                     sel.Offset[-1, 0].Select();
8                     break;
9                 case "下":

```

```

10         sel.Offset[1, 0].Select();
11         break;
12     case "左":
13         sel.Offset[0, -1].Select();
14         break;
15     case "右":
16         sel.Offset[0, 1].Select();
17         break;
18     case "显示网格线":
19         ExcelApp.ActiveWindow.DisplayGridlines = true;
20         break;
21     case "隐藏网格线":
22         ExcelApp.ActiveWindow.DisplayGridlines = false;
23         break;
24
25     default:
26         ExcelApp.ActiveCell.Value = Ctrl.Caption;
27         break;
28     }

```

该事件处理过程的思路是，根据每一个按钮的Caption不同加以区分，借助switch结构分别处理每一个按钮的功能。

■ 8.2.3 修改右键菜单

接下来看一下用于定制工作表标签菜单的子过程AddControlToPly:

```

1     public void AddControlToPly()
2     {
3         Office.CommandBarButton btn;
4         try
5         {
6             ExcelApp.CommandBars["Ply"].Controls["显隐标题栏"].Delete();
7         }
8         catch (System.SystemException ex)
9         {
10        }
11        btn = (Office.CommandBarButton)ExcelApp.CommandBars["Ply"].
Controls.Add(Office.MsoControlType.msoControlButton, Type.Missing, Type.Missing,
Type.Missing, true);
12        btn.Caption = "显隐标题栏";
13        btn.FaceId = 9162;
14        btn.Style = Office.MsoButtonStyle.msoButtonIconAndCaption;
15        btn.Click += new Office._CommandBarButtonEvents_ClickEventHandler(
DisplayHeadings);
16    }

```

由于这个过程不是创建工具栏，而是向现有的菜单中添加新控件，为此，需要事先删除“显隐标题栏”这个控件，无论它是否已存在都进行删除，因为有错误处理策略。

添加控件的方法和上一个过程完全一样，只是这次的单击事件名称是DisplayHeadings:

```
17         public void DisplayHeadings(Microsoft.Office.Core.CommandBarButton
18             Ctrl1, ref bool CancelDefault)
19         {
20             ExcelApp.ActiveWindow.DisplayHeadings = !(ExcelApp.ActiveWindow.
                DisplayHeadings);
21         }
```

■ 8.2.4 卸载外接程序时清除自定义

开发者应该有一个好的习惯，那就是用户不使用你的插件时，应把各项配置恢复为原样。为此，一定要修改ThisAddIn_Shutdown事件。

```
1     private void ThisAddIn_Shutdown(object sender, System.EventArgs e)
2     {
3         try
4         {
5             ExcelApp.CommandBars["cmb"].Delete();
6             ExcelApp.CommandBars["Ply"].Controls["显隐标题栏"].Delete();
7         }
8         catch (System.SystemException ex)
9         {
10        }
11    }
12 }
```

启动调试，在Excel 2010 中切换到“加载项”选项卡，会看到自定义工具栏。单击四个方向按钮，Excel所选区域会自动平移；单击【网格线/隐藏网格线】，当前工作表不显示网格线，如图8.1所示。

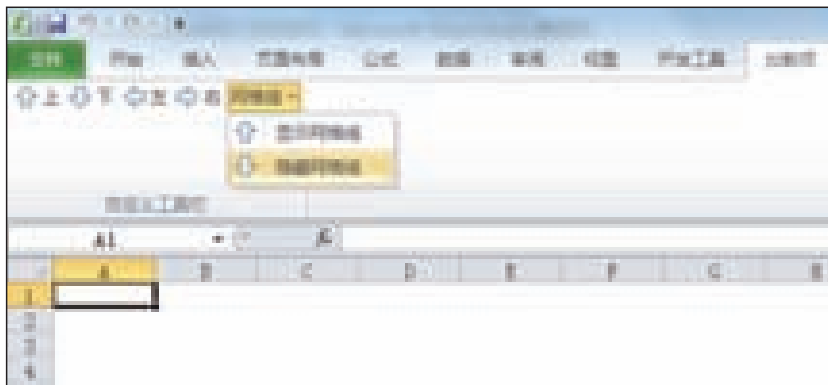


图8.1 自定义工具栏和控件

接着右击工作表标签，效果如图8.2所示。

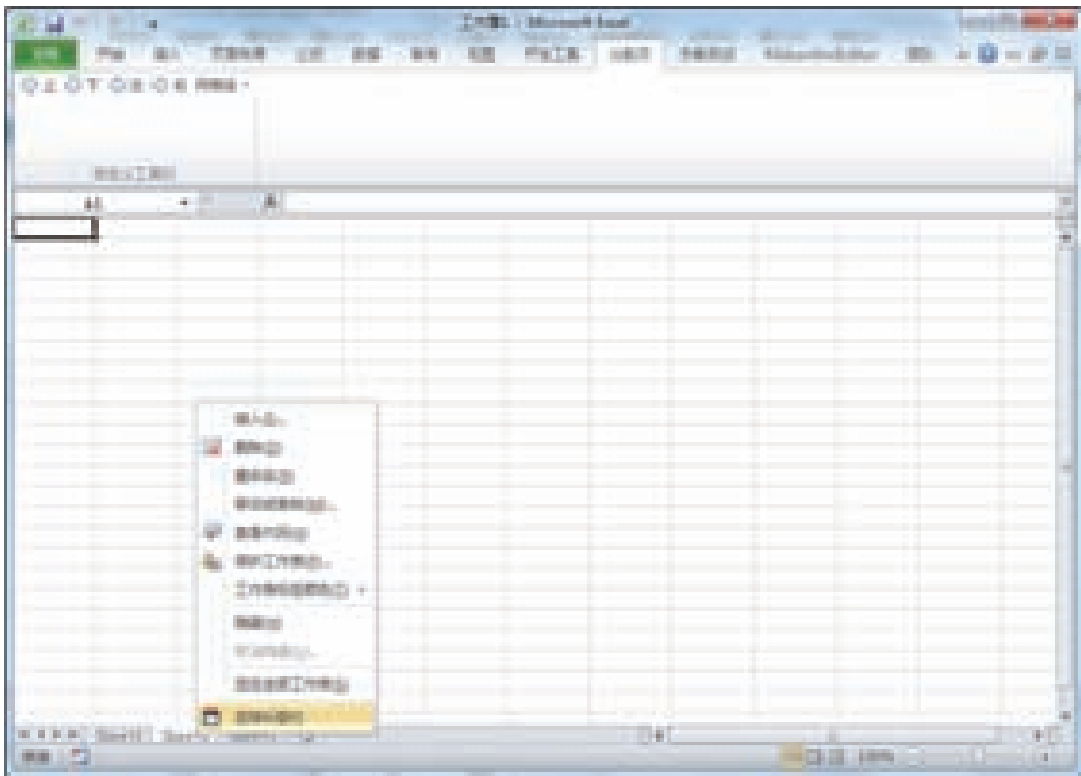


图8.2 自定义工作表右键菜单

会看到菜单下方多了一个按钮。单击该按钮隐藏工作表的行号列标。

本章要点回顾

- 在Office 2003以下版本，Commandbar对象是界面设计的主要对象。Office的界面由若干工具栏构成，每一个工具栏包含相关功能的控件。
- 学会自定义工具栏的技术，既可以修改Office内置工具栏和控件的属性，也可以创建全新的自定义工具栏或控件。
- Office内置工具栏只能重置（Reset），不能删除；自定义工具栏只能删除，不能重置。
- 工具栏控件（CommandbarControl）可以移动、复制和删除。
- FaceID是一个数字编号，用它来决定工具栏控件的图标。

第9章

VSTO外接程序的部署分发

开发完成的VSTO外接程序，除了能在开发机器上正常运行外，还应该在没有安装VSTO的计算机上正常加载。

 本章视频：VSTO外接程序的打包.wmv

9.1 客户机搭建VSTO运行环境

本节所述VSTO运行环境并非指VSTO编程环境，而是指客户机上使用VSTO外接程序的必备条件。

客户机除了事先安装好Office 2010之外，还需要安装如下两个组件方可使用VSTO外接程序。

1. 安装Microsoft Net Framework 4.0

在微软官网下载dotNetFx40_Client_x86_x64.exe按照提示安装即可。

2. 安装Microsoft Visual Studio 2010 Tools for Office Runtime x86

在微软官网下载vstor_redist.exe，按照提示安装即可。

以上两项安装完成后打开控制面板，会看到新安装的组件，如图9.1所示。

客户机具备了上述条件，就可以安装使用开发机器上做好的VSTO外接程序了，安装的方法分为简单安装和制作安装包两种。

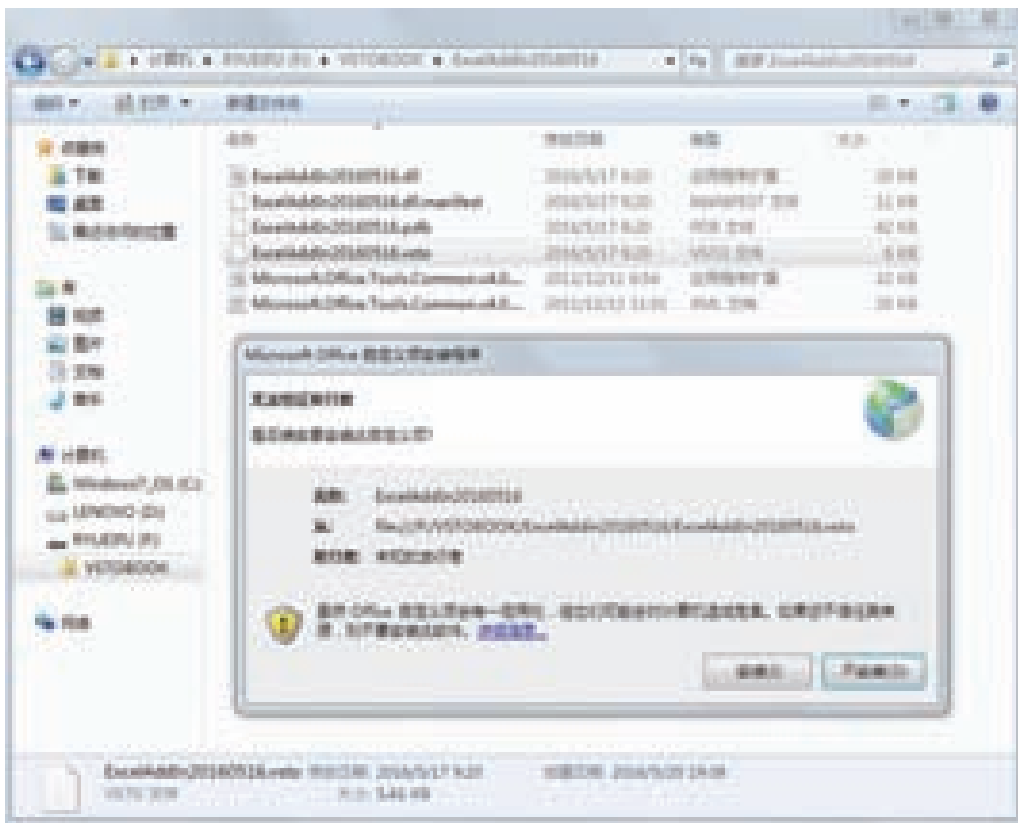


图9.3 安装VSTO产品

重新启动Excel 2010，会看到自定义任务窗格，同时在Excel的“COM加载项”对话框中，可以看到ExcelAddIn20160516这个新的COM加载项。

9.3 使用Advanced Installer

使用Advanced Installer可以制作出更专业的安装包。开发人员只需要把生成的安装程序（扩展名是.exe或.msi）发送给客户机安装即可，而无需发送Debug文件夹。

因此，需要在开发机器上安装Advanced Installer 11.0，根据VSTO解决方案文件制作出安装包，然后把安装包发送给客户机。

客户机接收到安装包后，按照提示安装即可使用。

■ 9.3.1 创建aip安装包工程

本节使用Advanced Installer 11.0为“ExcelAddIn20160516”这个Excel外接程序创建安装包。

第1步：完全关闭Visual Studio和Excel。因为Advanced Installer要用到解决方案和项目中的文件，因此需要确认Visual Studio和Excel处于关闭状态。

第2步：打开Advanced Installer，依次单击菜单【新建/加载项/Office Add-in】，双击Office Add-in按钮，或者单击右下角的“Create Project”，如图9.4所示。



图9.4 创建VSTO安装程序项目

第3步：输入应用程序名称和组织的名称，如图9.5所示。

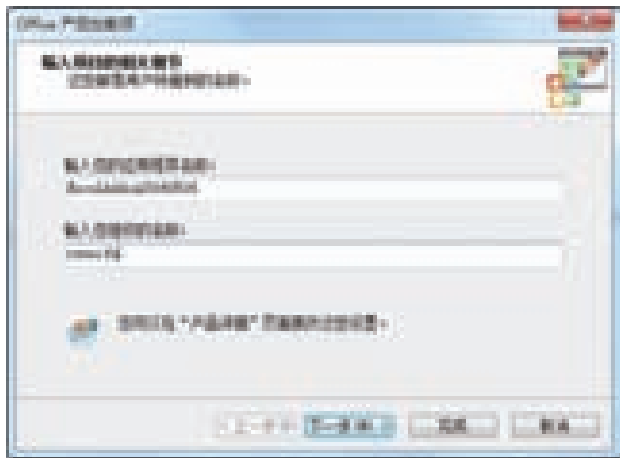


图9.5 VSTO安装程序相关细节

第4步：选择安装包文件类型，选择“MSI安装文件”单选按钮，如图9.6所示，单击“下一步”按钮。

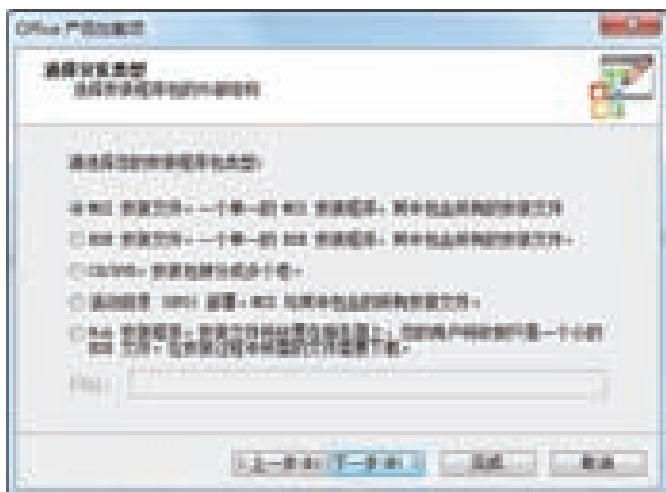


图9.6 安装程序分发类型选择

第5步：设置项目和程序包的路径。打开的“设置项目和程序包的路径”对话框的各项都可以保持默认，第一个路径文本框要求选择项目文件夹，这是因为Advanced Installer创建的也是一个工程，这个工程的主文件的扩展名是.aip，如果以后开发人员还对VSTO项目进行修改并且调试，那么更新安装包的时候，在Advanced Installer中打开该aip即可，无需重新创建安装包。

项目输出文件夹：最后生成的msi安装包输出到该文件夹。

安装软件包的名称：用户指定。后面会自动加上“.msi”，如图9.7所示。

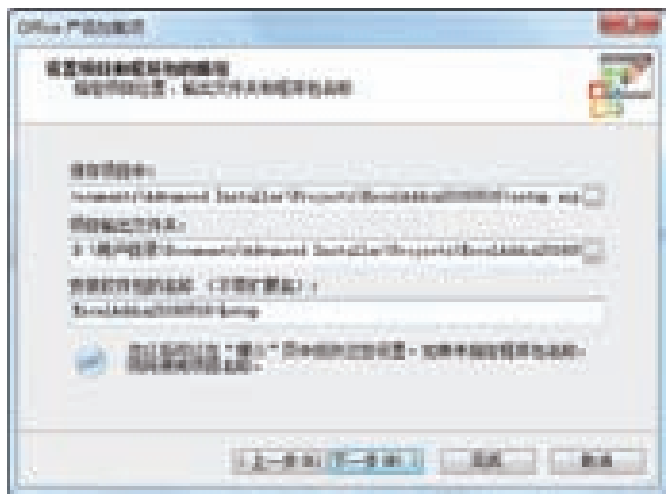


图9.7 输出路径设定

第6步：Office添加类型，选择“创建 VSTO Office 加载项”单选按钮，如图9.8所示。

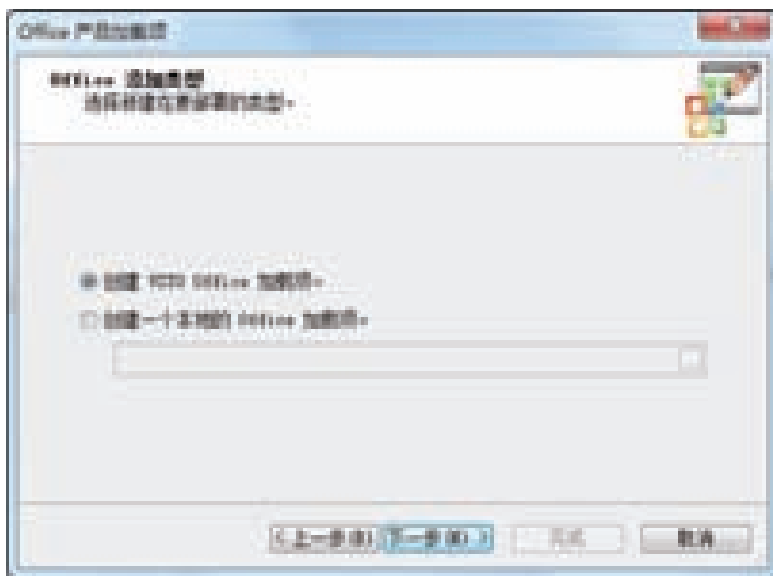


图9.8 Office加载项类型选择

第7步：资源文件位置。该对话框非常重要，用以指定VSTO外接程序解决方案的路径。找到.sln主文件并选择即可，如图9.9所示。

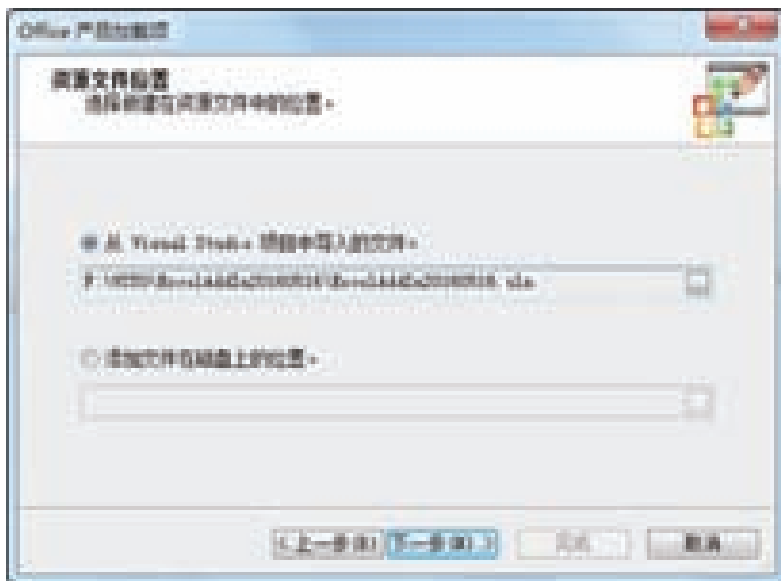


图9.9 导入解决方案

第8步：已检测到的配置。因为在VSTO项目调试的时候，选择的是“Debug”，所以选择“Debug”配置，如图9.10所示。

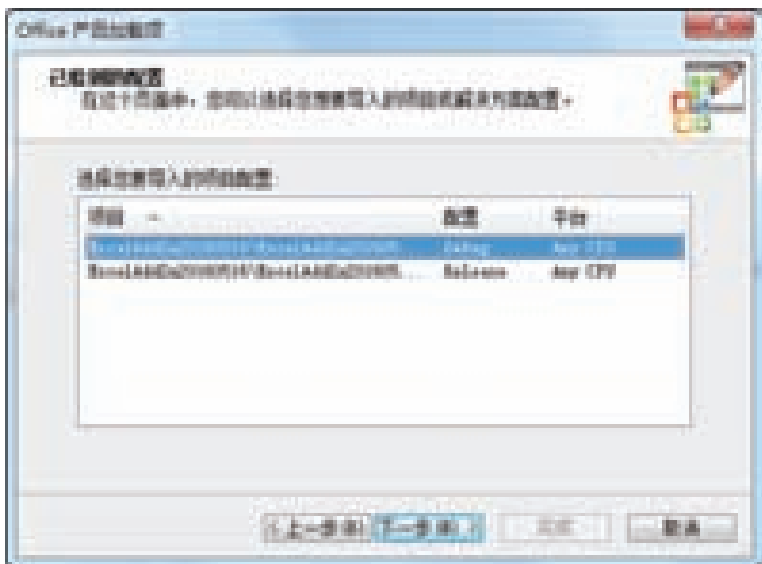


图9.10 配置选择

第9步：资源文件。该对话框采用默认设置，如图9.11所示。

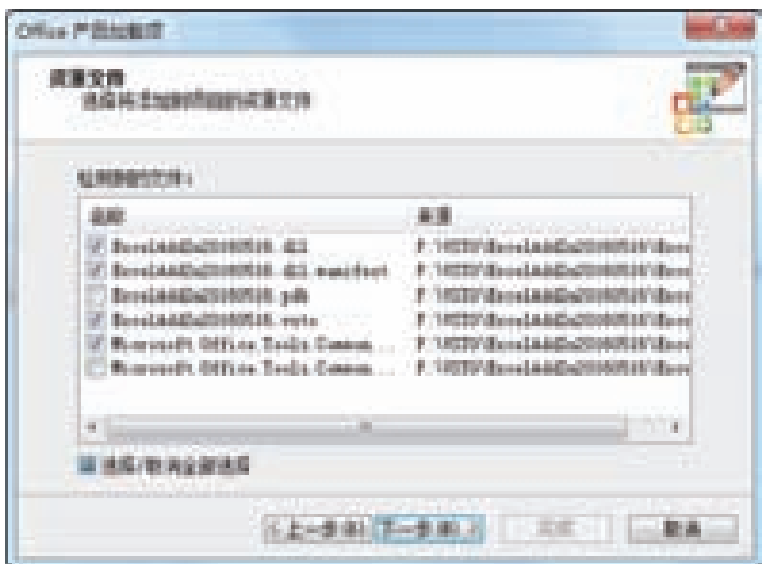


图9.11 添加文件

第10步：加载项细节。依次选择“Visual Studio 2010 Microsoft Office开发”，“Microsoft Office 2010”，“Microsoft Excel”，如图9.12所示。

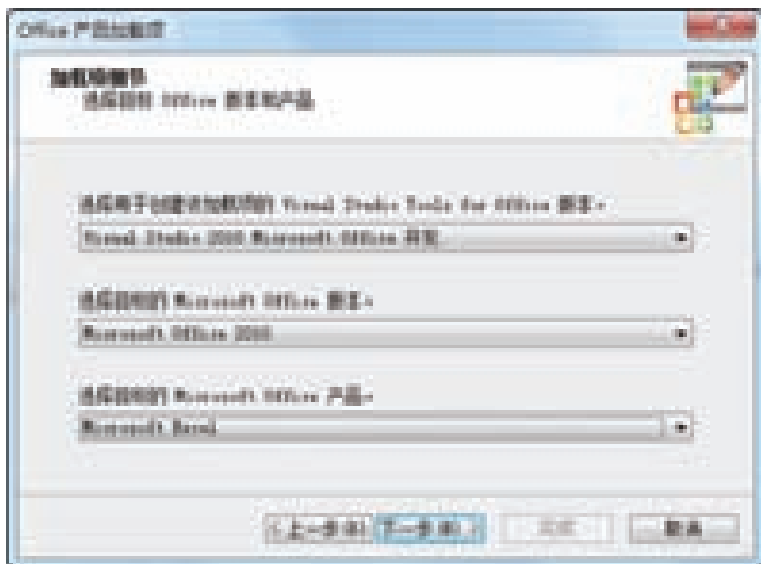


图9.12 Visual Studio及Office版本选择

第11步：加载项配置。在该对话框最下面，可以选择“只为当前用户安装外接程序”，或者“为所有用户安装外接程序”，如图9.13所示。

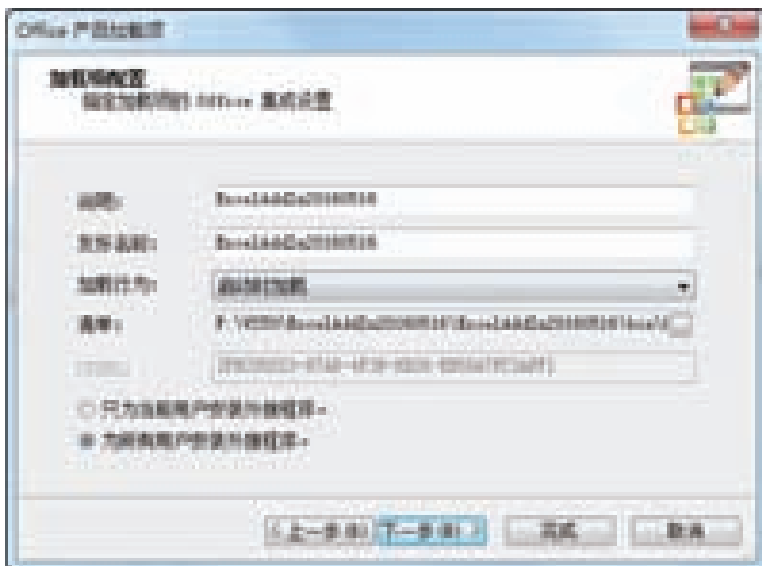


图9.13 选择加载项的加载行为

第12步：配置加载项启动条件。都不要勾选，如图9.14所示。

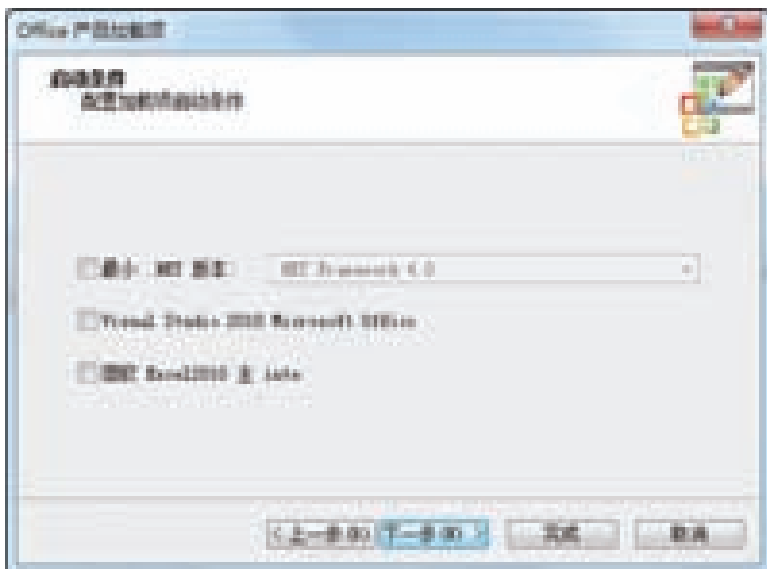


图9.14 加载项启动条件

第13步：运行环境的添加。都不勾选，如图9.15所示。

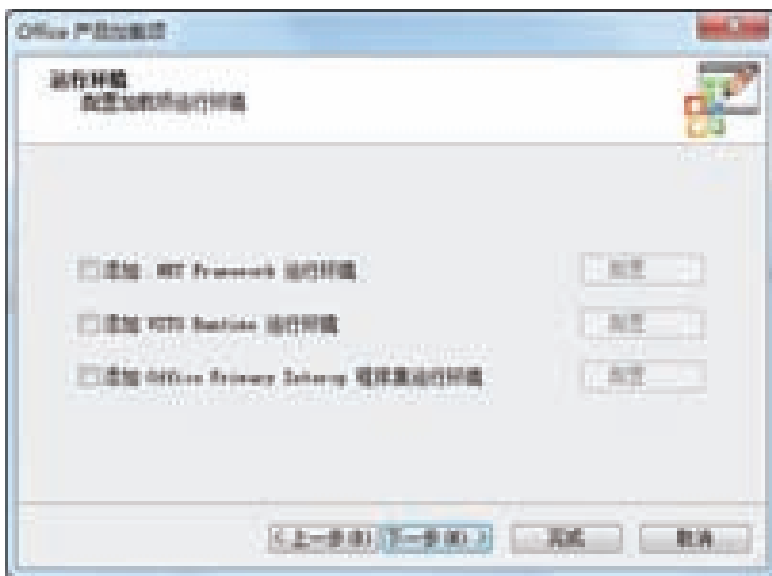


图9.15 加载项运行环境

第14步：选择构建语言。选中“Chinese Simplified (PRC)”复选框，如图9.16所示。

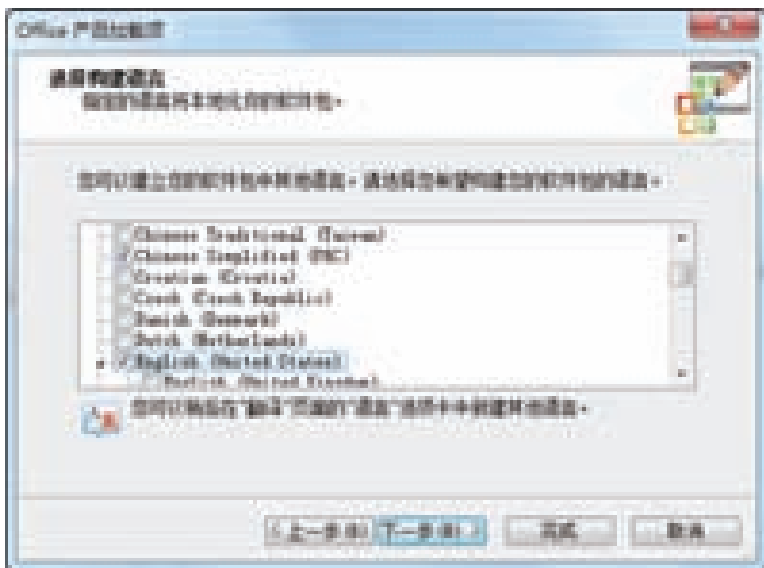


图9.16 安装程序语言选择

第15步：许可协议。如果要为安装包添加一个许可协议，需要选中“添加许可协议对话框”复选框，并且浏览到一个rtf格式的许可协议，如图9.17所示。如果还没有rtf文件，请保持现在的对话框不要关闭，启动Word，编辑许可协议，另存为rtf文件即可，如图9.18所示。

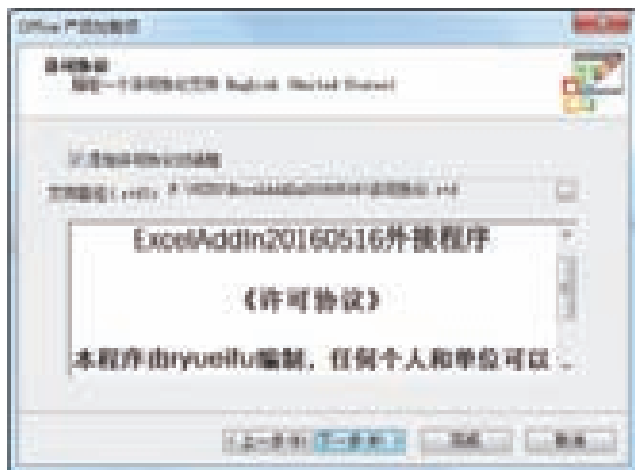


图9.17 添加许可协议

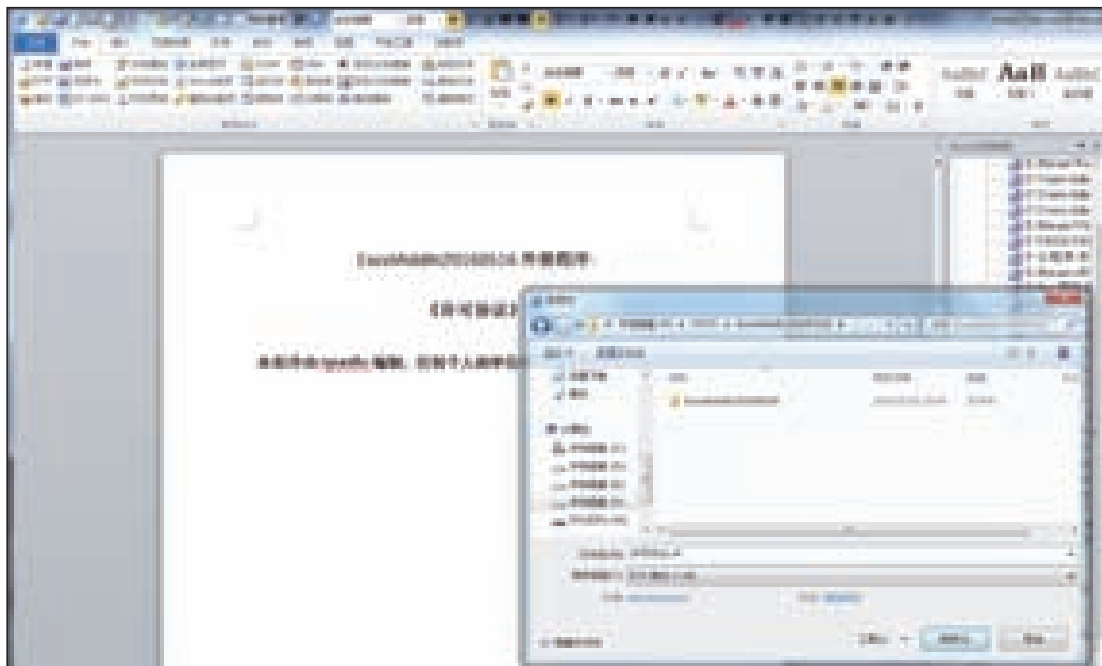


图9.18 利用Word创建rtf许可协议

第16步：单击“完成”按钮，自动构建项目，如图9.19所示。

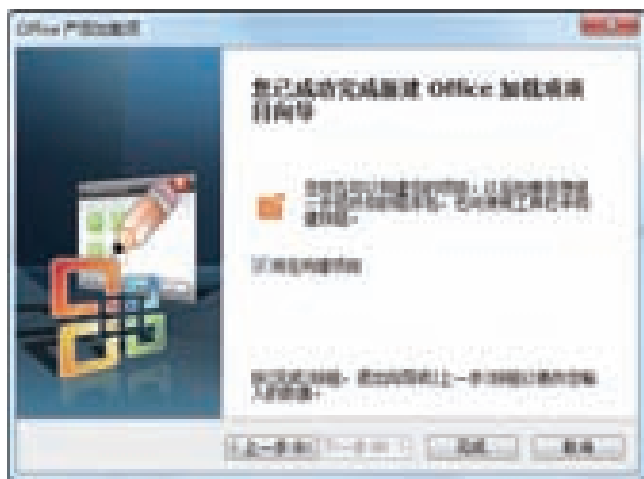


图9.19 构建安装程序

第17步：在Advanced Installer中，单击“打开输出文件夹”，会在“...\Setup Files\zh”这个路径下找到中文界面的安装包“ExcelAddin20160516-setup.msi”，如图9.20所示。

这个msi文件就是最终的安装包。

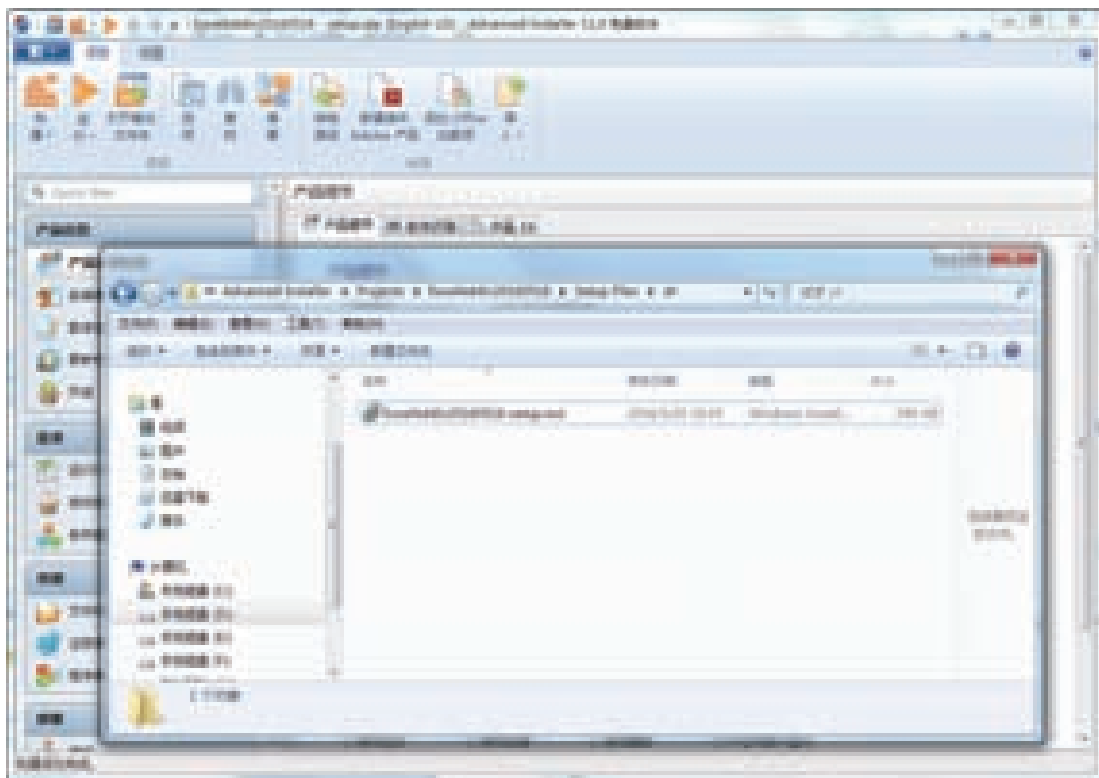


图9.20 安装程序的生成结果

■ 9.3.2 客户机运行安装包

把创建好的msi文件发送给客户机，双击打开该文件，出现的安装向导如图9.21～图9.24所示。



图9.21 客户机运行安装程序

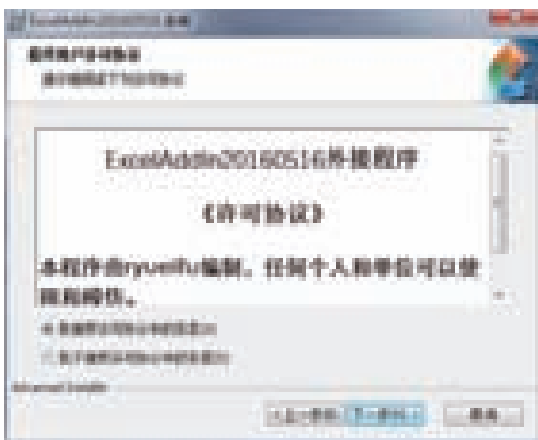


图9.22 接受许可协议

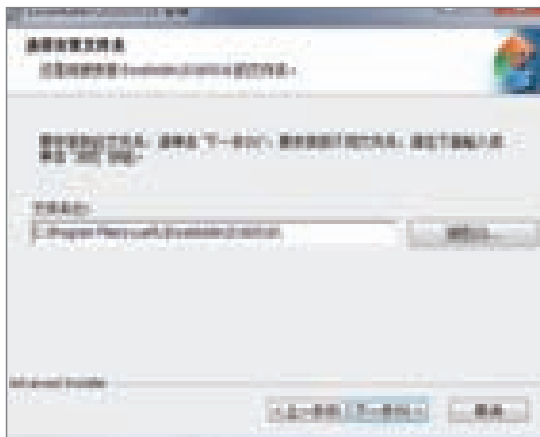


图9.23 选择安装路径

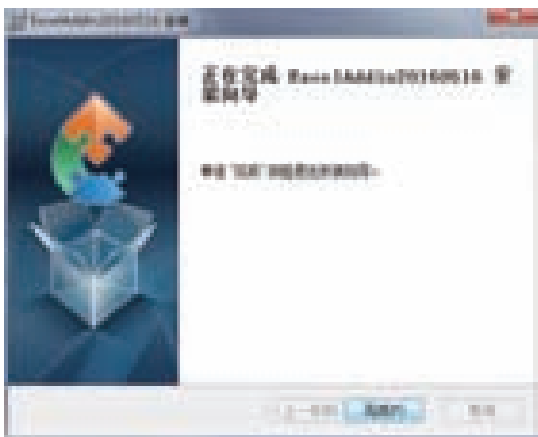


图9.24 完成安装

依次单击“下一步”按钮直至“完成”。重新启动Excel 2010，会看到该外接程序已成功加载到客户机。

本章要点回顾

- 本章介绍了使用Advanced Installer对VSTO作品进行部署和分发。
- 分发之前，必须先确保VSTO项目调试过程没有出现任何错误。通过向Advanced Installer中导入VSTO解决方案，依照向导创建aip工程，构建项目后，生成安装包。

第 10 章

VSTO开发Office文档

VSTO开发中，除了可以创建Office外接程序外，还可以开发具有特定功能的Office文档和模板，开发Office文档的过程也叫做“文档自定义项编程”。VSTO中的外接程序和Office文档开发最大的不同之处在于作用范围不同，外接程序是应用程序级的加载项，无论是否有打开的文档，外接程序的界面功能一直存在，直到卸载它或者完全退出Office组件才终止。而Office文档的界面和功能完全处于文档之中，文档打开界面随之出现，文档关闭时所有功能随之消失。

 本章视频：VSTO开发Office文档-文档操作窗格.wmv

10.1 文档自定义项编程概述

VSTO中，可以创建的Office文档项目有以下4项：

- Excel模板
- Excel工作簿
- Word模板
- Word文档

其中，模板和工作簿（文档）开发过程完全一样，只是最后生成的文件不同，例如Excel 2010 工作簿的文件扩展名是.xlsx或支持VBA宏的.xlsm，而Excel 2010模板文件则是.xltx或.xltn。模板文件的作用是可以创建基于模板的文档。

10.2 文档自定义项允许添加的界面元素

VSTO自定义Office文档开发过程中允许进行如下操作：

1. 编辑文档

VSTO中创建的Office文档会在Visual Studio集成开发环境中出现，和正常的Office界面没什么区别。

2. 文档中添加C#控件

可以添加需要的C#控件到文档。

3. 自定义界面 (CustomUI)

可以开发文档级的自定义界面，开发过程和第6章介绍的完全一样。唯一不同的是，Office外接程序的自定义界面伴随着外接程序的加载和卸载而产生和消失。而自定义文档的界面则是伴随着文档的打开和关闭而产生和消失。

4. 添加文档操作窗格 (ActionsPane)

在VSTO开发Office文档项目中，ThisWorkbook是Excel的最顶层对象，地位等同于外接程序中的ThisAddin，类似地，Word文档项目中，ThisDocument对象是最顶层对象。

ThisWorkbook和ThisDocument对象都有一个ActionsPane对象（文档操作窗格），而且，一个文档只允许有一个ActionsPane，这和外接程序中的任务窗格有很大的不同。一个文档操作窗格上面，允许添加多个不同的用户控件（UserControl）以及一般的窗体控件。

10.3 创建Office文档项目

本节内容通过一个典型的范例，依次介绍上述各部分界面元素的制作技术。本例的制作目的是，利用VSTO创建一个Excel 2010 工作簿，保证该工作簿有三个工作表，第一个工作表上放置一个C#列表框控件，单击列表框任一条目后，Excel所选区域内容为列表框当前条目。

接下来设计该文档的文档操作窗格，文档窗格中依次添加红、黄、绿三个不同背景颜色的用户控件。

为文档增加自定义功能区用于控制文档窗格的：

- 显示和隐藏
- 停靠位置
- 控件的堆叠顺序

启动Visual Studio 2012，单击菜单【文件/新建/项目】，在对话框中选择“Excel 2010 工作簿”，项目名称命名为“ExcelWorkbook20160519”，如图10.1所示。

单击“确定”后，询问是否创建新文档，如果不打算存储VBA工程，则接受默认设置即可，否则，在“格式”下拉列表框中选择.xlsx文件，单击“确定”按钮，如图10.2所示。

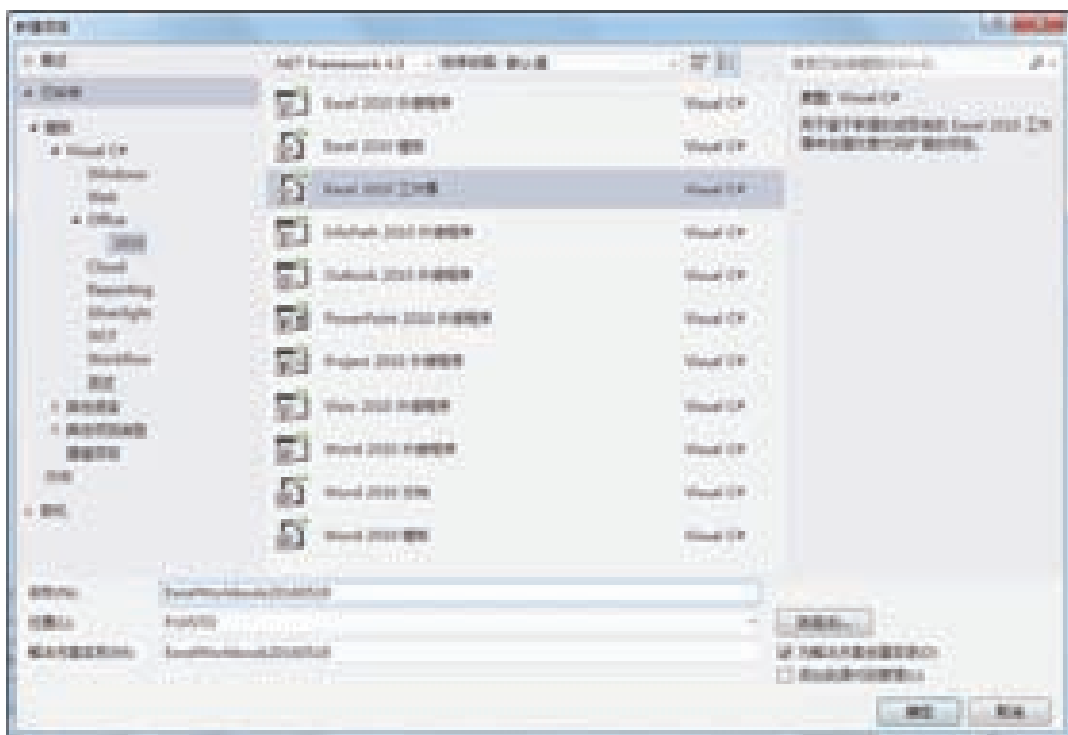


图10.1 创建Office文档项目

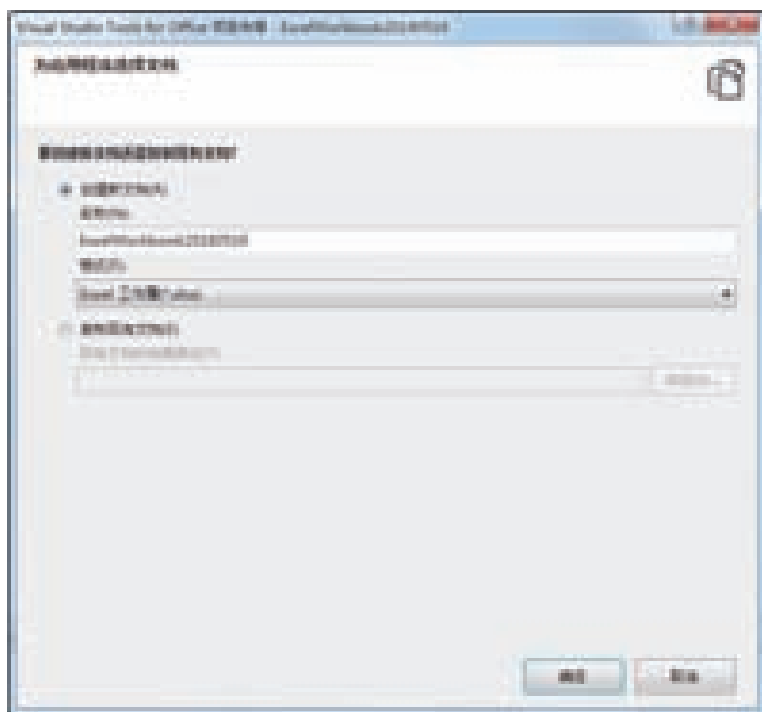


图10.2 选择文档

稍等片刻，会在Visual Studio中看到嵌入的Excel界面，在解决方案资源管理器中，可以看到ThisWorkbook和3个工作表的类模块。

■ 10.3.1 文档上添加C#控件

在Visual Studio中，单击工程资源管理器中的Sheet1.cs，打开第一个工作表。然后拖动控件工具箱中的ListBox控件到Sheet1里。在列表框的属性窗口中找到Items，依次输入列表框中的各项，如图10.3所示。

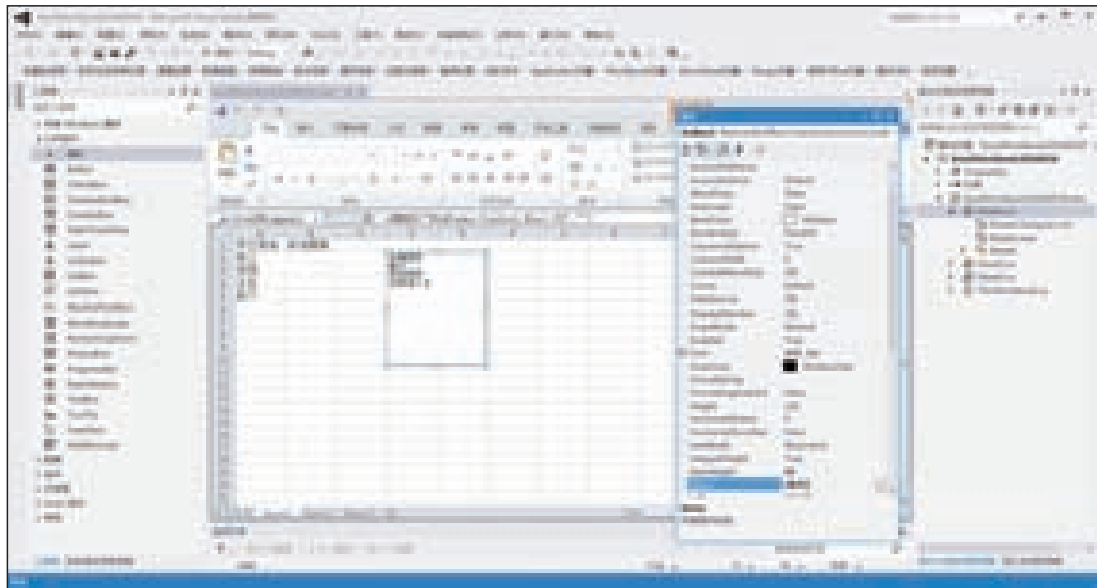


图10.3 文档上添加C#控件

以上步骤为列表框控件预设了若干项内容。

■ 10.3.2 文档项目的启动事件过程

为了便于说明VSTO项目启动事件的作用，鼠标选择解决方案资源管理器中的Sheet1.cs，按下【F7】键编辑代码。该模块中Sheet1_Startup过程是工作表的启动事件，在该事件中，我们再为列表框控件添加三项内容。代码为：

```
1      private void Sheet1_Startup(object sender, System.EventArgs e)
2      {
3          this.listBox1.Items.Add("预备党员");
4          this.listBox1.Items.Add("民主人士");
5          this.listBox1.Items.Add("海外侨胞");
6      }
7
8      private void Sheet1_Shutdown(object sender, System.EventArgs e)
9      {
10     }
```

此外, 还应该追加列表框控件的单击事件过程。切换到Sheet1的设计视图, 右击列表框控件, 选择“查看代码”, 编辑其单击事件:

```
1     private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
2     {
3         Excel.Range rng;
4         rng = (Excel.Range)this.Application.Selection;
5         rng.Value = this.listBox1.Text;
6     }
```

代码的含义是, 声明一个Range变量使其等于Excel所选区域, 用户一单击列表框, 就自动把列表框当前条目输入到Excel所选区域中。

启动调试, VSTO项目中的工作簿在Excel中打开, 可以看到列表框中共有7项内容, 有4项是设计期间预设的, 后3项是启动事件中使用代码添加的。单击任一条目, Excel所选区域自动填写。运行效果如图10.4所示。

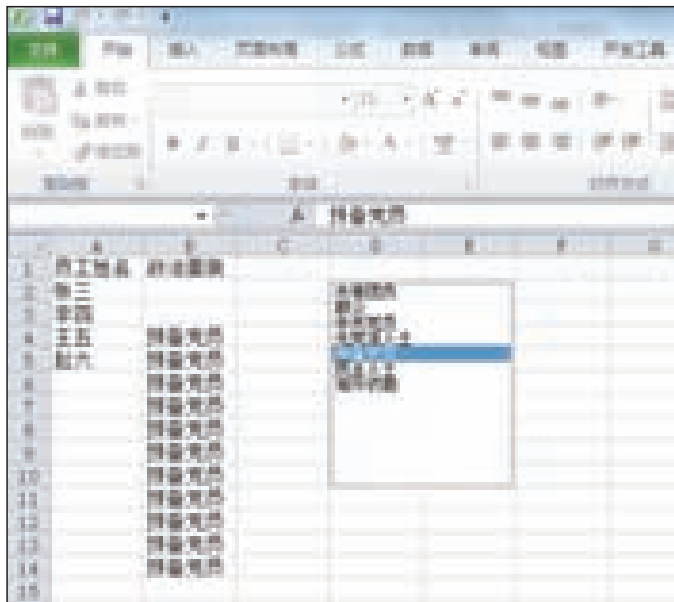


图10.4 运行效果

调试完毕后, 退出Excel, 关闭解决方案。

这样工作表就可以当做是一个很大的窗体, 根据任务需求, 可以放置各种各样的C#控件上去, 本例仅以列表框说明问题, 其他控件读者自行尝试。

10.4 文档操作窗格概述

文档操作窗格在Office文档开发中占据着重要地位, 我们先了解一下文档操作窗格的

行为和重要属性。文档操作窗格的很多属性是用CommandBars[“task pane”]这个工具栏对象控制的。可以认为文档操作窗格是放在工具栏上的一些控件。实际上，文档操作窗格本身不需要创建，创建了VSTO的Office文档项目，该文档就已经存在文档操作窗格了，我们要做的只是如何显示出文档窗格，以及如何添加控件上去。

1. 文档窗格的显示和隐藏

使用Application.CommandBars[“task pane”].Visible =true/false显示和隐藏文档窗格。

2. 文档窗格中控件的显示和隐藏

使用Globals.ThisWorkbook.ActionsPane.Visible =true/false显示和隐藏窗格中的控件。

3. 文档窗格的停靠

相当于是设置工具栏在应用程序中的停靠位置，Application.CommandBars[“Task Pane”].Position=Microsoft.Office.Core.MsoBarPosition.msoBarLeft，意思是文档窗格停靠在Excel界面的左侧。

4. 控件在文档窗格中的堆叠方向

一个文档窗格允许放置多个控件，堆叠方向就是指控件在窗格中的放置顺序，例如代码：

Globals.ThisWorkbook.ActionsPane.StackOrder = Microsoft.Office.Tools.StackStyle.FromLeft含义是放上去的控件按照从左到右重排。

5. 文档窗格中控件的添加

既可以使用Globals.ThisWorkbook.ActionsPane.Controls.Add(control)为文档窗格添加一个控件，也可以使用Globals.ThisWorkbook.ActionsPane.Controls.AddRange(controls)为文档窗格添加控件数组。

10.5 文档操作窗格综合实例

10.3节只介绍了如何在工作表上使用C#控件。本节继续在该项目基础上介绍如何增加自定义功能区 and 文档操作窗格。因此在Visual Studio中仍然打开Office文档项目“ExcelWorkbook20160519”，为了便于今后对文档操作窗格进行多方面交互，为该项目添加一个静态类Share.cs，如图10.5所示。

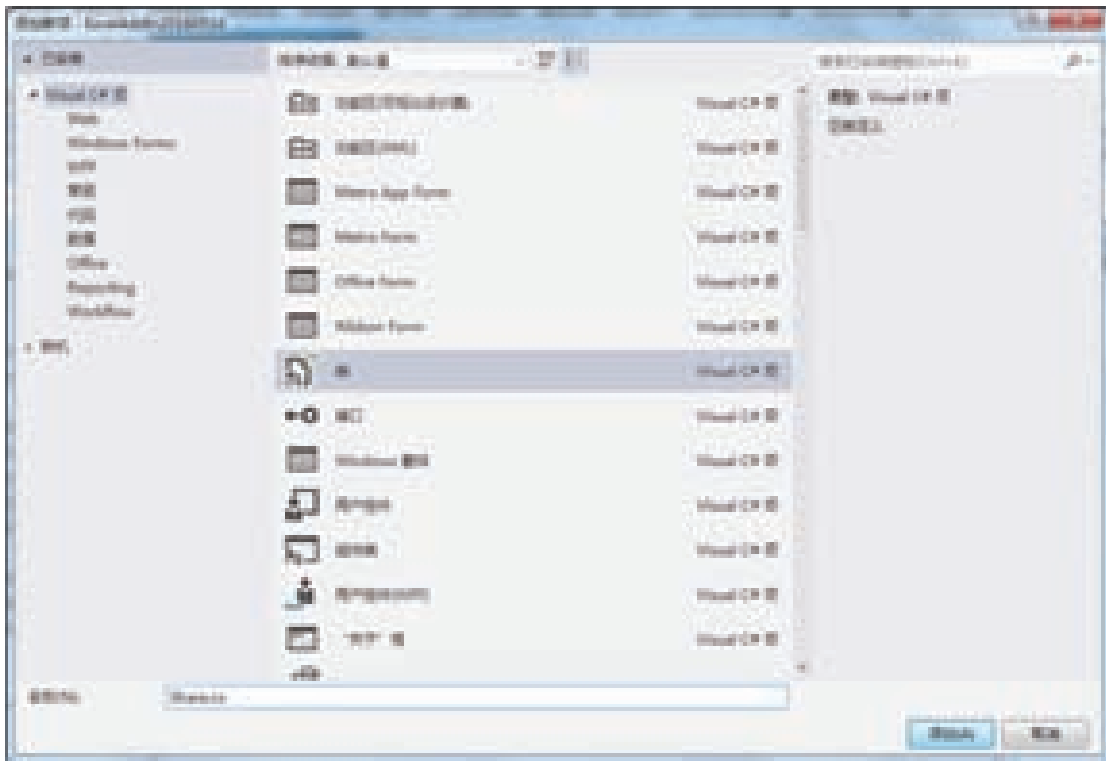


图10.5 使用静态类

Share.cs中修改代码如下：

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using Excel = Microsoft.Office.Interop.Excel;
6 using Office = Microsoft.Office.Core;
7 namespace ExcelWorkbook20160519
8 {
9     public static class Share
10    {
11        public static Excel.Application ExcelApp;
12        public static Office.CommandBar cmb;
13        public static Microsoft.Office.Tools.ActionsPane acp;
14    }
15 }

```

静态类中定义了三个公共对象：ExcelApp是整个Excel应用程序；cmb是任务窗格工具栏，用于实现文档窗格的可见性和停靠位置；acp则是文档操作窗格对象。

接着修改ThisWorkbook.cs中的启动事件代码：

```
1     private void ThisWorkbook_Startup(object sender, System.EventArgs e)
2     {
3         Share.acp = this.ActionsPane;
4         Share.acp.Visible = true;
5     }
```

可以看到显示出文档窗格只需要1行代码：`this.ActionsPane.Visible=true`。

启动调试，会在Excel的右侧看到“文档操作”的空白窗格，如图10.6所示。

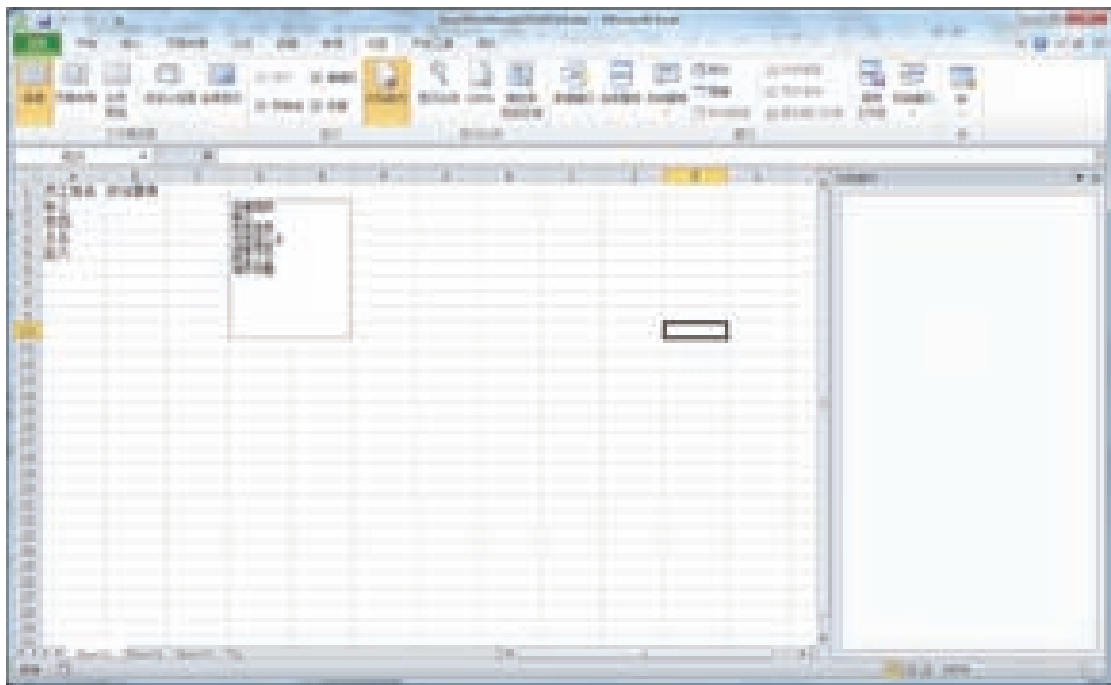


图10.6 文档操作窗格运行效果

鼠标单击Excel的“视图”选项卡，会看到有一个“文档操作”的切换按钮用于控制窗格的可见性。

注意 只有VSTO创建的Office文档，“视图”选项卡中才有此切换按钮，该按钮的imageMso=“ViewDocumentActionsPane”。

■ 10.5.1 添加用户控件到文档窗格

单击菜单【项目/添加用户控件】，为项目依次添加UserControl1、UserControl2、UserControl3三个用户控件，如图10.7所示。

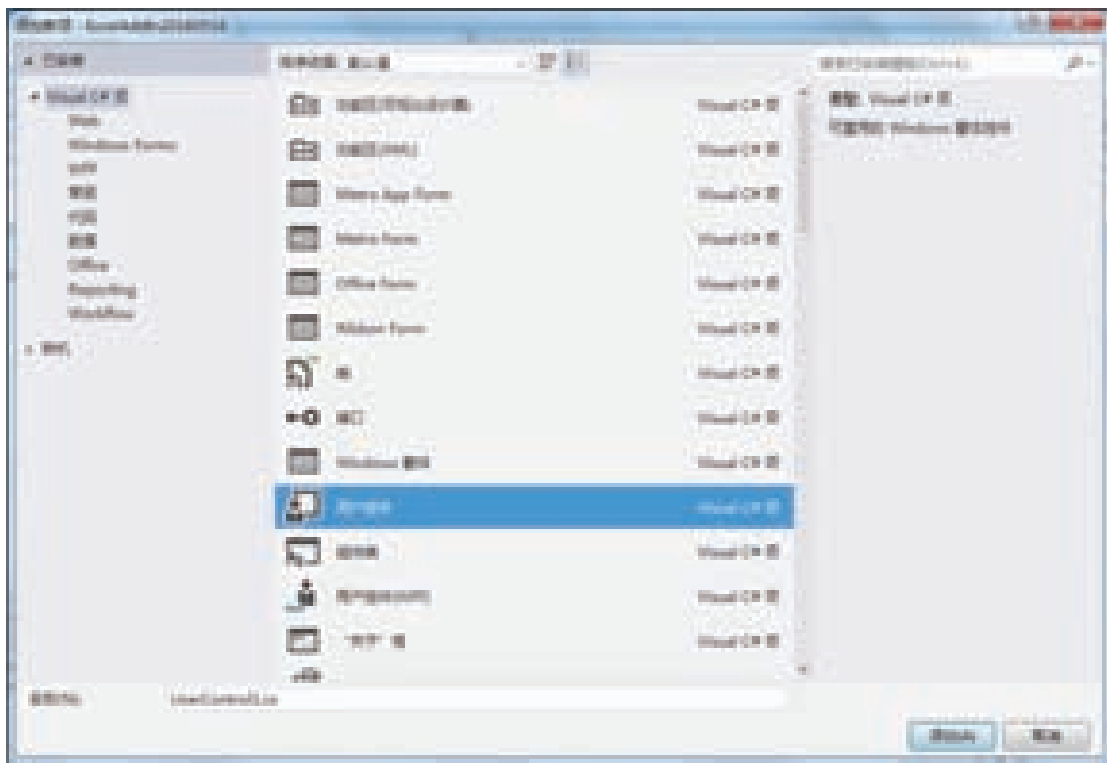


图10.7 添加用户控件

在用户控件的设计视图中打开属性窗格，设置它们的BackColor属性依次为红色、黄色和绿色，便于区别。

然后回到ThisWorkbook.cs类模块中的ThisWorkbook_Startup启动事件过程，修改为：

```

1      private void ThisWorkbook_Startup(object sender, System.EventArgs e)
2      {
3          UserControl uc1 = new UserControl1();
4          UserControl uc2 = new UserControl2();
5          UserControl uc3 = new UserControl3();
6          Share.acp = this.ActionsPane;
7          Share.acp.Controls.Add(uc1);
8          Share.acp.Controls.Add(uc2);
9          Share.acp.Controls.Add(uc3);
10         Share.acp.Visible = true;
11     }

```

代码中uc1、uc2和uc3是用户控件的实例，使用acp.Controls.Add(uc)方法依次添加每一个用户控件到文档窗格。

启动调试，在Excel右侧看到窗格中自上而下排列着三个用户控件，如图10.8所示。

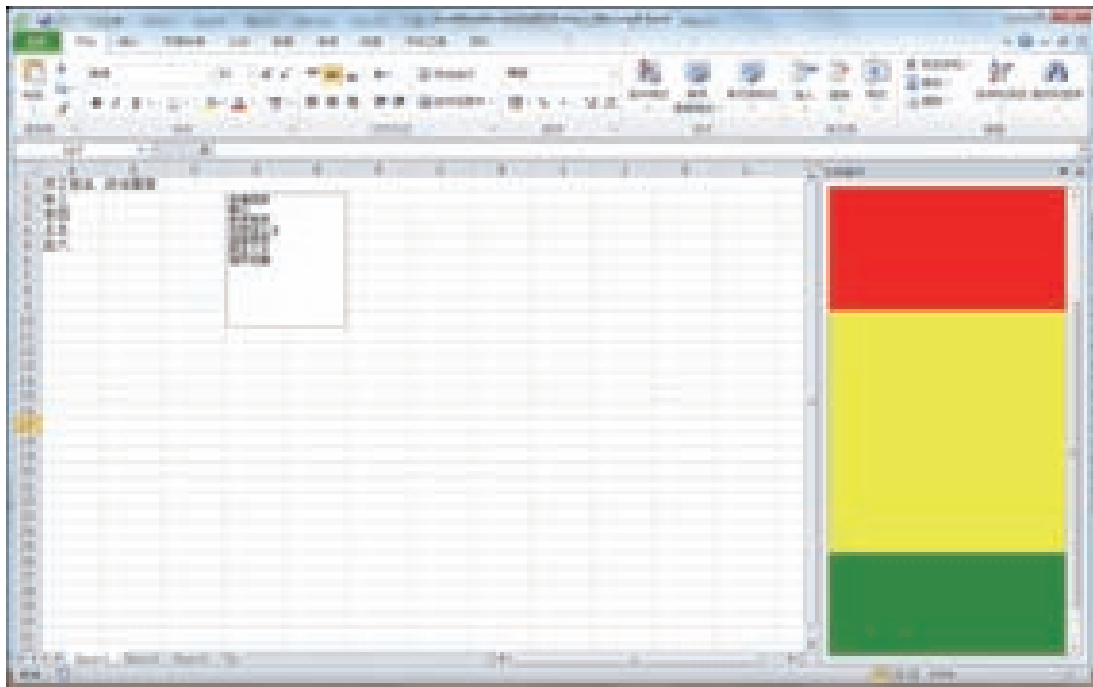


图10.8 任务窗格中的用户控件

注意 可以根据需要，在用户控件上添加适量其他控件。

■ 10.5.2 添加多个相同控件到文档窗格

有些场合下，需要添加的用户控件是同一个，但要添加多个，为此本节介绍一下AddRange方法添加控件数组的技术。

终止调试，继续为项目添加一个名为“UserControl4”的用户控件，在设计视图中，把UserControl4背景色设为蓝色，并放置一个Button到UserControl4上，如图10.9所示。



图10.9 添加控件到用户控件

然后，修改ThisWorkbook_Startup事件过程为：

```
1 private void ThisWorkbook_Startup(object sender, System.EventArgs e)
2 {
```

```
3         UserControl4[] arr = new UserControl4[4];
4         for (int i = 0; i < 4; i++)
5         {
6             arr[i] = new UserControl4();
7         }
8         Share.acp = this.ActionsPane;
9         Share.acp.Controls.AddRange(arr);
10        Share.acp.Visible = true;
11    }
```

注意 arr是一个控件数组，里面的每一个元素都是一个UserControl4。
启动调试后，会在文档操作窗格中看到从上到下一模一样的4个用户控件。

■ 10.5.3 使用代码创建窗体控件并添加到文档操作窗格

前面的内容都是先在用户控件设计视图中把必要的控件放在用户控件上，再把用户控件添加到文档窗格。其实，也可以一步直接把控件放在文档操作窗格上。

修改ThisWorkbook_Startup代码为：

```
1     private void ThisWorkbook_Startup(object sender, System.EventArgs e)
2     {
3         TextBox tb1 = new TextBox();
4         tb1.Text = "vsto";
5         tb1.BackColor = System.Drawing.Color.Yellow;
6         tb1.Left = 20;
7         tb1.Top = 30;
8         tb1.Multiline = true;
9         tb1.Width = 100;
10        tb1.Height = 30;
11
12        Button btn = new Button();
13        btn.Text = "新按钮";
14        btn.Width = 100;
15        btn.Height = 30;
16        btn.Click += new EventHandler(btn_Click);
17
18        Share.acp = this.ActionsPane;
19        Share.acp.Controls.Add(tb1);
20        Share.acp.Controls.Add(btn);
21        Share.acp.Visible = true;
22    }
23    private void btn_Click(object sender, System.EventArgs e)
24    {
25        MessageBox.Show((sender as Button).Text);
26    }
```

上面代码中，tb1是一个新文本框控件，btn是一个新按钮，它们的所有属性以及事件处理都可以在代码中设定好，最后使用Controls.Add把它们添加到文档操作窗格中。

运行效果如图10.10所示。

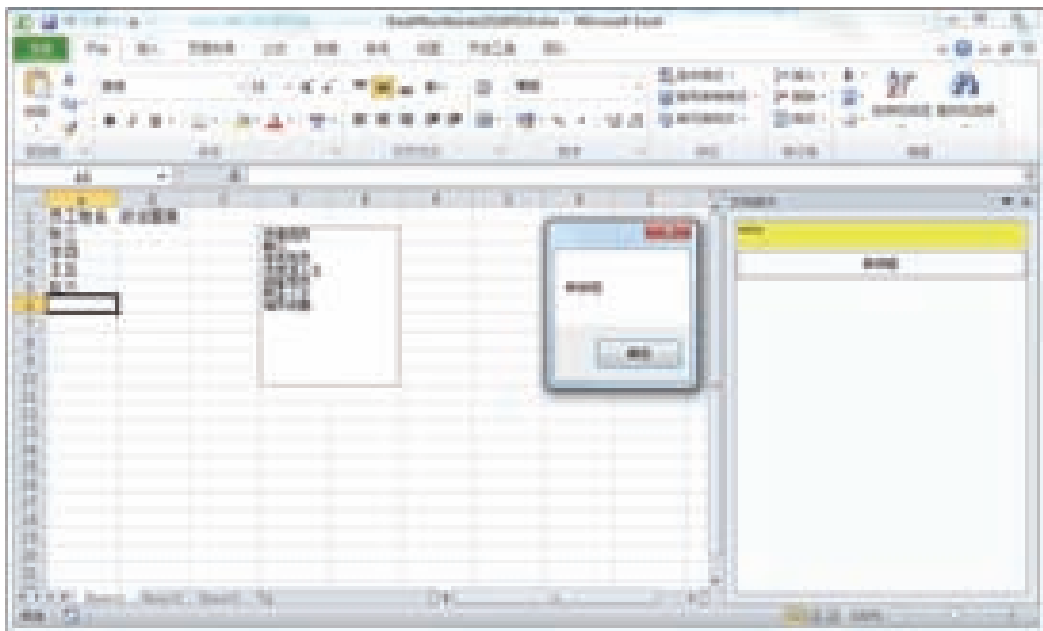


图10.10 使用代码创建控件

10.5.4 定制功能区按钮控制文档操作窗格

终止项目调试，再次回到ThisWorkbook.cs类模块中，修改启动事件代码如下：

```

1      private void ThisWorkbook_Startup(object sender, System.EventArgs e)
2      {
3          Share.ExcelApp = this.Application;
4          Share.cmb = Share.ExcelApp.CommandBars["task pane"];
5          Share.acp = this.ActionsPane;
6
7          UserControl uc1 = new UserControl1();//红色用户控件
8          UserControl uc2 = new UserControl2();//黄色用户控件
9          UserControl uc3 = new UserControl3();//绿色用户控件
10
11         Share.acp.Controls.Add(uc1);
12         Share.acp.Controls.Add(uc2);
13         Share.acp.Controls.Add(uc3);
14         Share.acp.Visible = true;
15     }

```

注意 启动事件中使用了静态类中的两个重要变量Share.cmb和Share.acp。

接着为项目添加名为“Ribbon1.cs”的功能区可视化设计器。在自定义选项卡中放入3个group，第一个group放入2个功能区的复选框控件，分别控制文档窗格的可见性和窗格内容的可见性；第二个group放入一个menu，menu控件里面放入5个button，控制文档窗格在

Excel中的停靠位置。第三个group放入4个按钮，用来控制窗格内各个控件的排布方向。
各个控件重要属性见表10.1。

表10.1 实例中所需控件及重要属性更改

控件名称	标题文字	重要属性更改
tab1	文档窗格综合控制	ControlIdType: Custom, Position: BeforeOfficeId TabHome
group1	Visible	
checkbox1	显示文档窗格	Checked: true
checkbox2	显示窗格中控件	Checked: true
group2	DockPosition	
menu1	停靠位置	
button1	靠左	
button2	靠右	
button3	靠上	
button4	靠下	
button5	浮动	
group3	StackOrder	
button6	自上而下	
button7	自下而上	
button8	从左到右	
button9	从右到左	

功能区设计视图如图10.11所示。

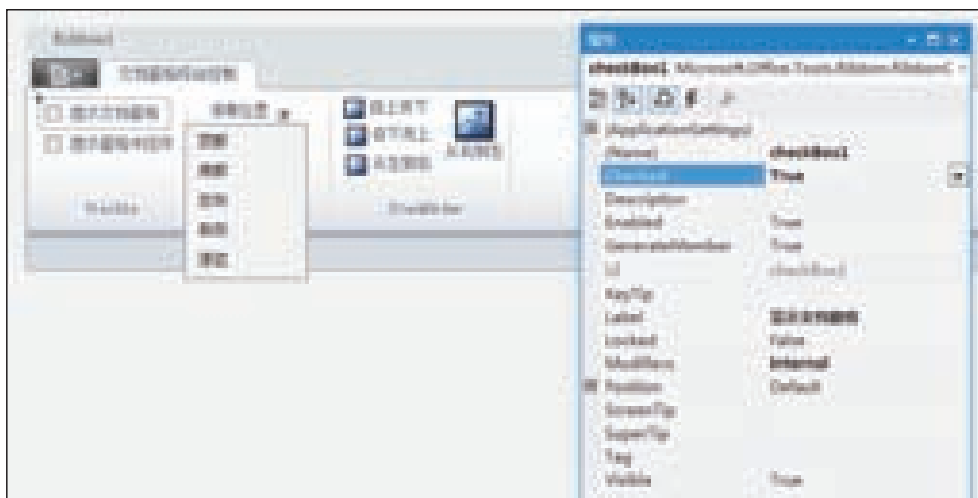


图10.11 功能区设计视图

然后在功能区设计视图中，依次双击每一个相关控件，编辑回调代码，Ribbon1.cs完整代码如下：


```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using Microsoft.Office.Tools.Ribbon;
6
7 namespace ExcelWorkbook20160519
8 {
9     public partial class Ribbon1
10     {
11         private void Ribbon1_Load(object sender, RibbonUIEventArgs e)
12         {
13
14         }
15
16         private void checkBox1_Click(object sender, RibbonControlEventArgs e)
17         {
18             //控制文档窗格的可见性
19             Share.cmb.Visible = this.checkBox1.Checked;
20         }
21         private void checkBox2_Click(object sender, RibbonControlEventArgs e)
22         {
23             //控制文档窗格内控件的可见性
24             Share.acp.Visible = this.checkBox2.Checked;
25         }
26         private void button1_Click(object sender, RibbonControlEventArgs e)
27         {
28             //文档窗格停靠到顶部
29             Share.cmb.Position = Microsoft.Office.Core.MsoBarPosition.msoBarTop;
30         }
31         private void button2_Click(object sender, RibbonControlEventArgs e)
32         {
33             Share.cmb.Position = Microsoft.Office.Core.MsoBarPosition.msoBarBottom;
34         }
35
36         private void button3_Click(object sender, RibbonControlEventArgs e)
37         {
38             Share.cmb.Position = Microsoft.Office.Core.MsoBarPosition.msoBarLeft;
39         }
40
41         private void button4_Click(object sender, RibbonControlEventArgs e)
42         {
43             Share.cmb.Position = Microsoft.Office.Core.MsoBarPosition.msoBarRight;
44         }
45
46         private void button5_Click(object sender, RibbonControlEventArgs e)
47         {
48             Share.cmb.Position = Microsoft.Office.Core.MsoBarPosition.
msoBarFloating;
49         }
50     }
```

```
51     private void button6_Click(object sender, RibbonControlEventArgs e)
52     {
53         // 文档窗格的堆叠方向为从上到下
54         Share.acp.StackOrder = Microsoft.Office.Tools.StackStyle.FromTop;
55     }
56     private void button7_Click(object sender, RibbonControlEventArgs e)
57     {
58         Share.acp.StackOrder = Microsoft.Office.Tools.StackStyle.FromBottom;
59     }
60     private void button8_Click(object sender, RibbonControlEventArgs e)
61     {
62         Share.acp.StackOrder = Microsoft.Office.Tools.StackStyle.FromLeft;
63     }
64     private void button9_Click(object sender, RibbonControlEventArgs e)
65     {
66         Share.acp.StackOrder = Microsoft.Office.Tools.StackStyle.FromRight;
67     }
68     }
69 }
70 }
71 }
72 }
```

启动调试，在Excel中“开始”选项卡之前出现自定义选项卡，取消选中“显示文档窗格”复选框，整个文档窗格消失，相当于用户单击了窗格右上角的关闭按钮，如图10.12所示。

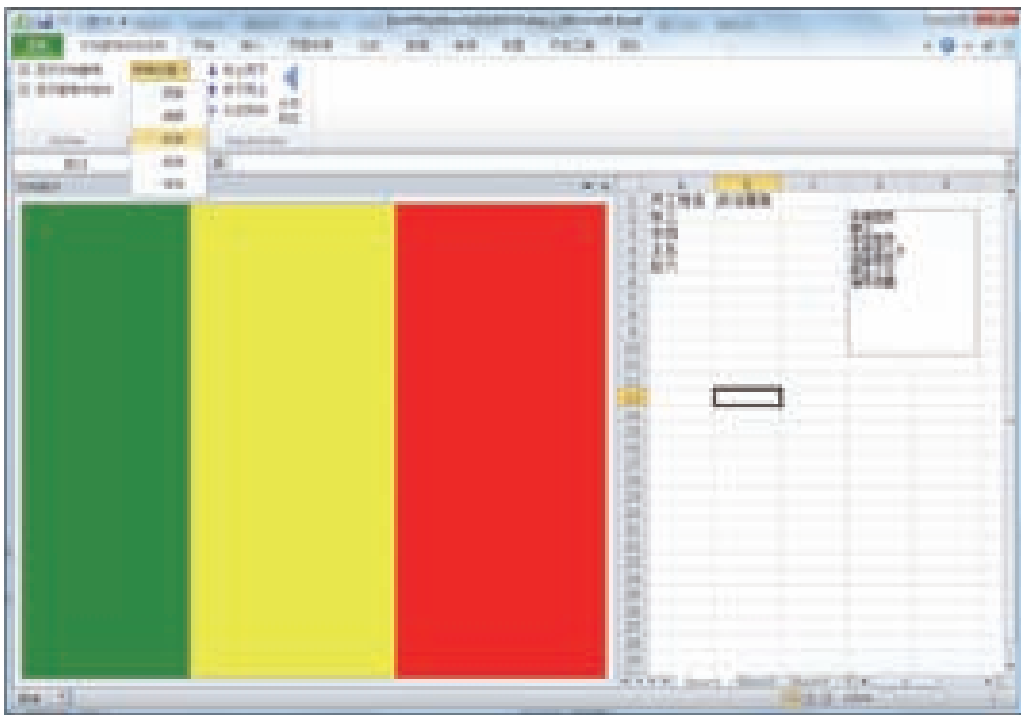


图10.12 文档窗格与功能区的交互控制

取消选中“显示窗格中控件”复选框，文档窗格不消失，但是变成了空白的窗格了，无任何控件。

单击【停靠位置/左侧】，窗格会自动移动到Excel的左侧。

单击“从右到左”，三个用户控件从右到左排列。

至此整个实例编制完毕。

10.6 文档自定义项的部署分发

如果要把开发好的Office文档复制到其他计算机使用，必须进行一系列的设定方可成功。本节要把上节制作好的Excel工作簿复制到如下配置的客户计算机：

- 系统：Windows 7（32bit）
- Office：Office 2010中文版

注意 其他配置的计算机也可以运行VSTO作品，仅以上述计算机为例。

在开发机器上把“F:\VSTO\ExcelWorkbook20160519\ExcelWorkbook20160519\bin\Debug”中的Debug文件夹压缩，复制或发送到客户计算机磁盘中。在客户计算机解压后，可以看到如下文件列表，如图10.13所示。

名称	修改日期	类型	大小
ExcelWorkbook20160519.dll	2016/5/19 22:28	应用程序扩展	217 KB
ExcelWorkbook20160519.dll.manifest	2016/5/19 22:28	Manifest 文件	12 KB
ExcelWorkbook20160519.gdls	2016/5/19 22:28	Program Binding...	88 KB
ExcelWorkbook20160519.resx	2016/5/19 22:28	VSTO Deployment...	8 KB
ExcelWorkbook20160519.xla	2016/5/19 22:28	Microsoft Excel...	217 KB
Microsoft.Office.Tools.Excel.v4.0.10.0.dll	2011/12/12 8:54	应用程序扩展	52 KB
Microsoft.Office.Tools.Excel.v4.0.10.0.dll.manifest	2011/12/12 15:55	Manifest 文件	20 KB
Microsoft.Office.Tools.Excel.v4.0.10.0.dll	2011/12/12 8:54	应用程序扩展	252 KB
Microsoft.Office.Tools.Excel.v4.0.10.0.dll.manifest	2011/5/17 14:01	Manifest 文件	8 KB

图10.13 文档自定义项生成文件

尝试双击“ExcelWorkbook20160519.xlsx”，在Excel 2010中出现如图10.14所示的对话框。

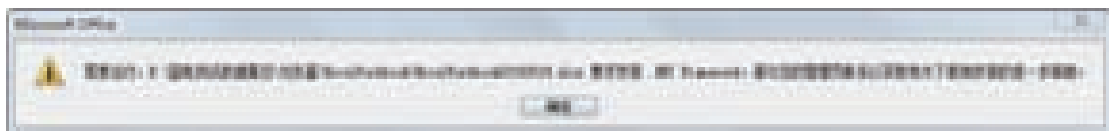


图10.14 提示信息

这说明客户计算机还不具备VSTO作品的运行条件，因此需要事先为客户计算机安装如下组件：

- Net Framework 4.0

- VSTO Runtime

安装步骤和方法参照9.1节。

在客户计算机重新启动Excel 2010，并再次打开刚才的ExcelWorkbook20160519.xlsx，会看到Excel界面中出现了文档操作窗格，功能和界面与开发机器中是一模一样的。

本章要点回顾

- 外接程序是应用程序级别的COM加载项；而文档自定义项的开发，则是针对特定文档的。VSTO可以创建Excel、Word、Powerpoint的文档自定义项。
- 文档自定义项的开发也支持自定义功能区，自定义功能区界面随着文档的打开而显示，随着文档的关闭而消失。
- 本章重点讲述文档操作窗格的开发技术。文档操作窗格在使用上和前面讲过的自定义窗格非常类似，但是开发过程大不相同。
- 可以调整文档操作窗格的堆叠方向、停靠位置等属性。

第 11 章

VSTO开发资源大全

VSTO开发是以操作Office对象为目的的一门开发技术，它的知识体系非常庞大，而且VSTO与VBA编程技术也有着非常紧密的联系。在开发过程中，开发人员除了掌握一般的开发知识外，很多情况下还需要一些资源的辅助方可完成任务。

为此，本章列出了VSTO编程开发中要用到的有力工具和资源的描述和使用方法，以便于读者事半功倍地解决编程过程中的难题。

本书配套资源中包含本章涉及的所有资源内容。

11.1 Office 2003以下版本工具栏和控件的自定义

在Office 2003版以下，还没有功能区的概念，因此Office.Commandbar和Control的创建和维护显得非常重要。

我们知道，无论是VBA还是VSTO编程，集合中成员的遍历是至关重要的，那么工具栏对象的编程要围绕Commandbar.Name进行，例如，要自定义Excel的“格式工具栏”，必须知道格式工具栏的Name，方可访问到它。

到底Office各个组件，有哪些内置工具栏和内置控件呢？

■ 11.1.1 OfficeCommandbarDesigner

OfficeCommandbarDesigner可以遍历Office五大组件，以及VBE编程环境所有工具栏和按钮的所有属性，并且可以动态地直接修改这些对象，例如工具栏的添加和删除、属性的修改等操作，如图11.1所示。

在使用本工具前，需要事先手动打开Office组件。

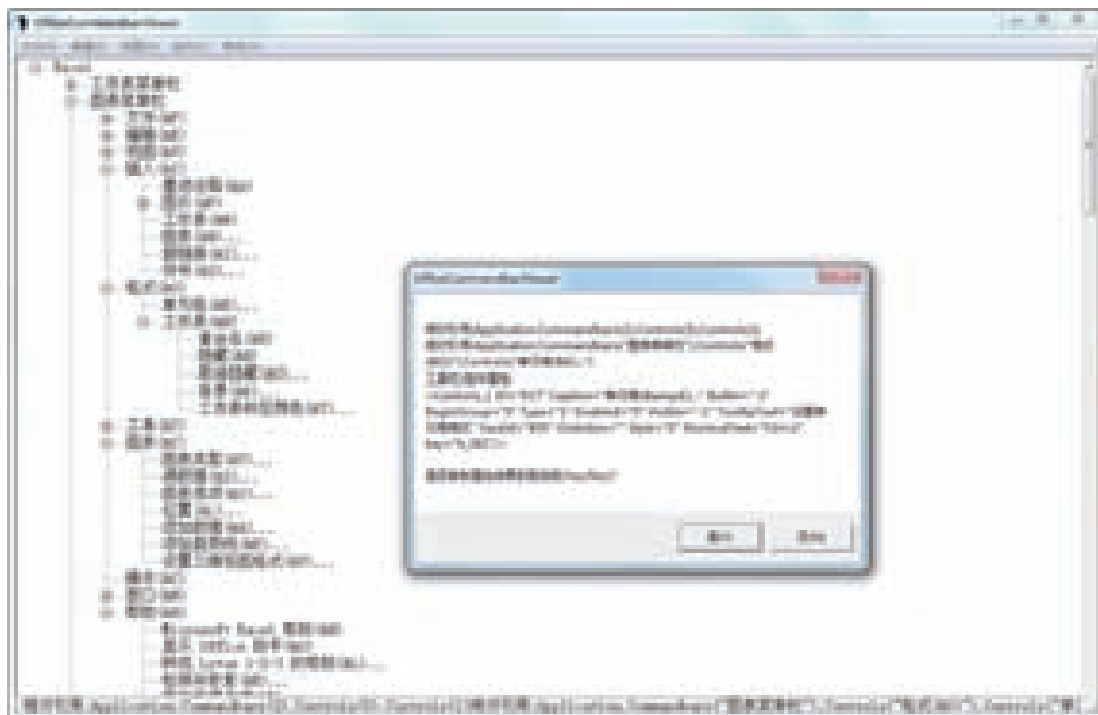


图11.2 OfficeCommandbarViewer

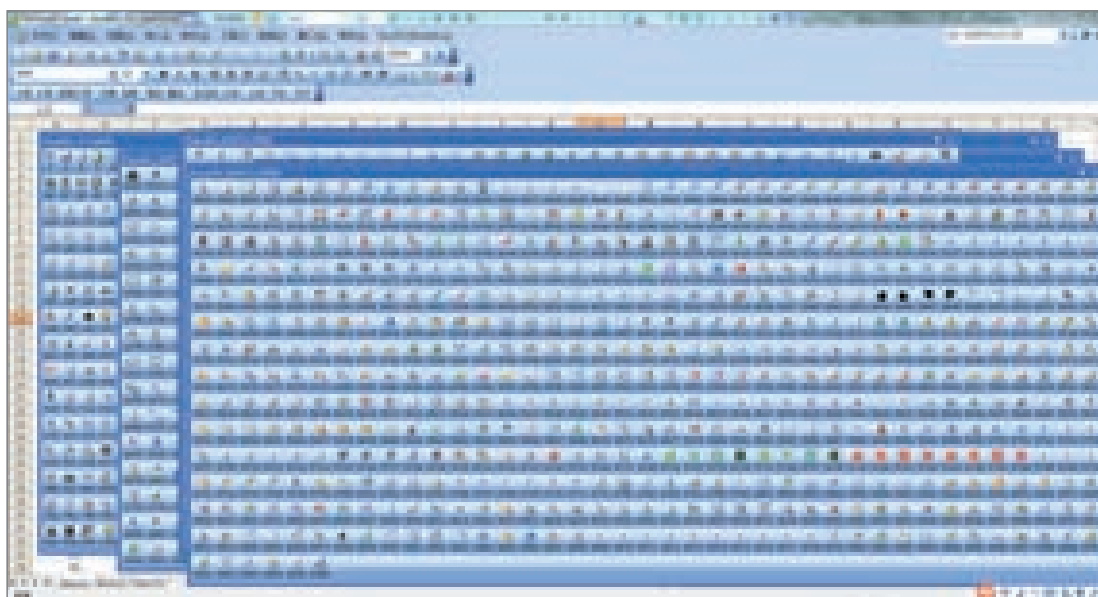


图11.3 FaceIDViewer

11.2 Office 2007以上版本功能区的自定义

自定义功能区涉及以下两部分内容：

- XML代码部分。
- 回调函数部分。

自定义功能区的外观取决于XML代码，而编写XML代码的过程会用到idMso、imageMso这些内置属性。

功能区控件的回调过程写法和开发环境有关，在VBA、VB6以及VSTO中回调函数的格式都有差异。

11.2.1 Office2010ControllDs

在Office 2007以上版本的功能区设计过程中，经常会操作Office内置选项卡、内置组、内置控件。要想操作内置元素，就必须知道该元素的idMso属性。

把Office2010ControllDs.rar解压缩到任意文件夹中，会看到如图11.4所示的资源。

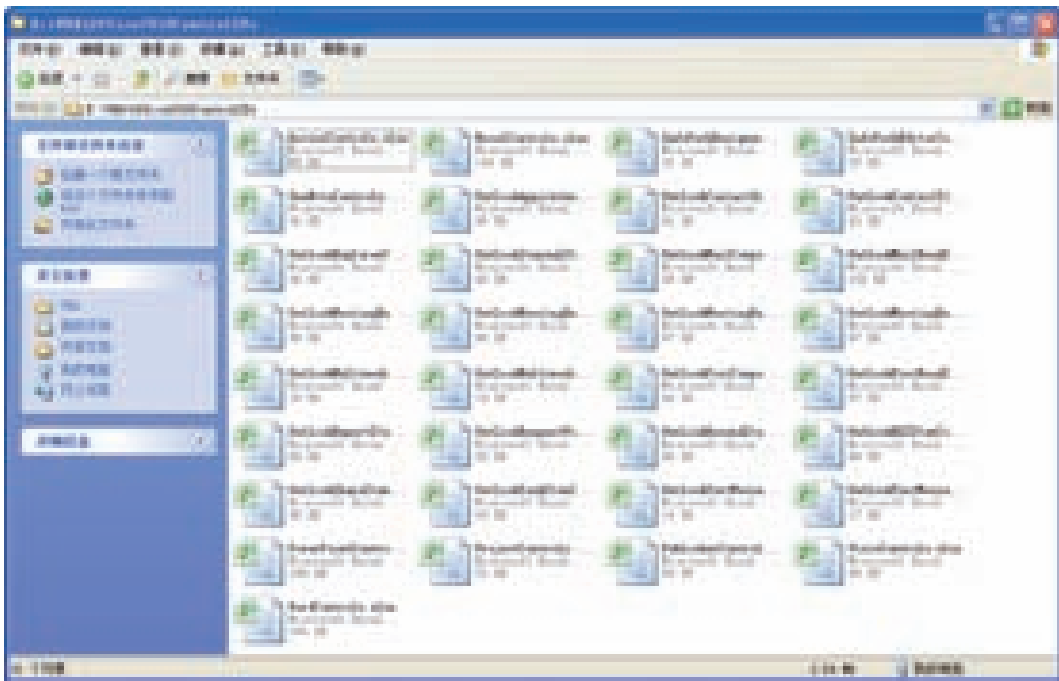


图11.4 Office各组件内置元素查询表

例如，要查看Excel 2010内置元素，就双击打开ExcelControls.xlsx这个文件。从工作表中可以看到Excel 2010所有内置元素的idMso属性，从而在自定义功能区时可以访问和操作内置元素，如图11.5所示。

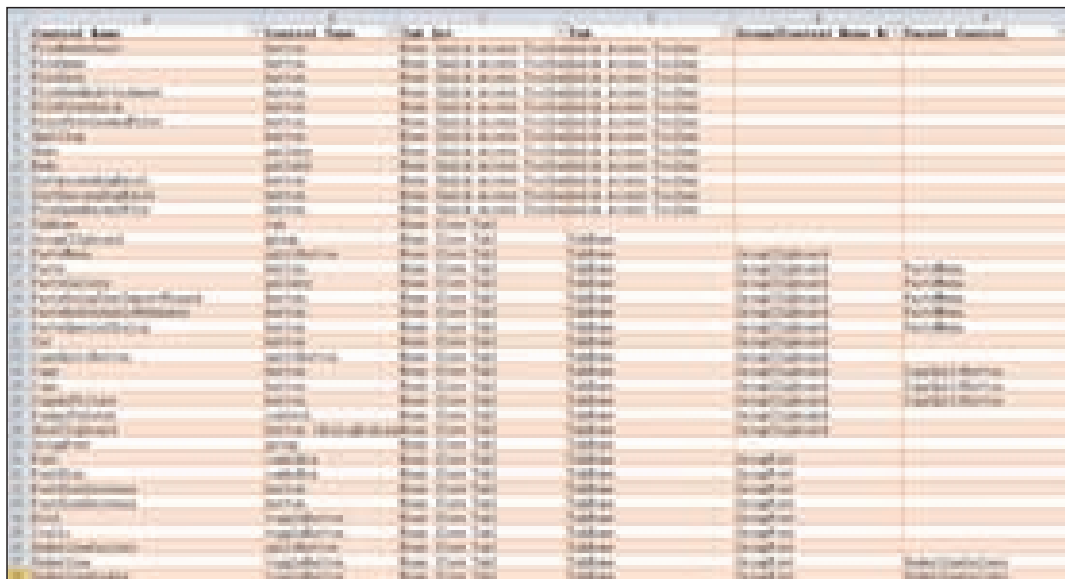


图11.5 Excel2010ControlIDs

11.2.2 imageMso7345

功能区控件的图标如果要使用计算机中的自定义图片，则设置其image属性；如果要使用Office内置图标，就需要查找idMso。Office 2007有1835个内置图标，Office 2010有7345个内置图标。

在Excel 2010中打开imageMso7345.xlsm这个文件，可以查看到Office 2010所有内置图标的外观和相应的imageMso字符串，如图11.6所示。

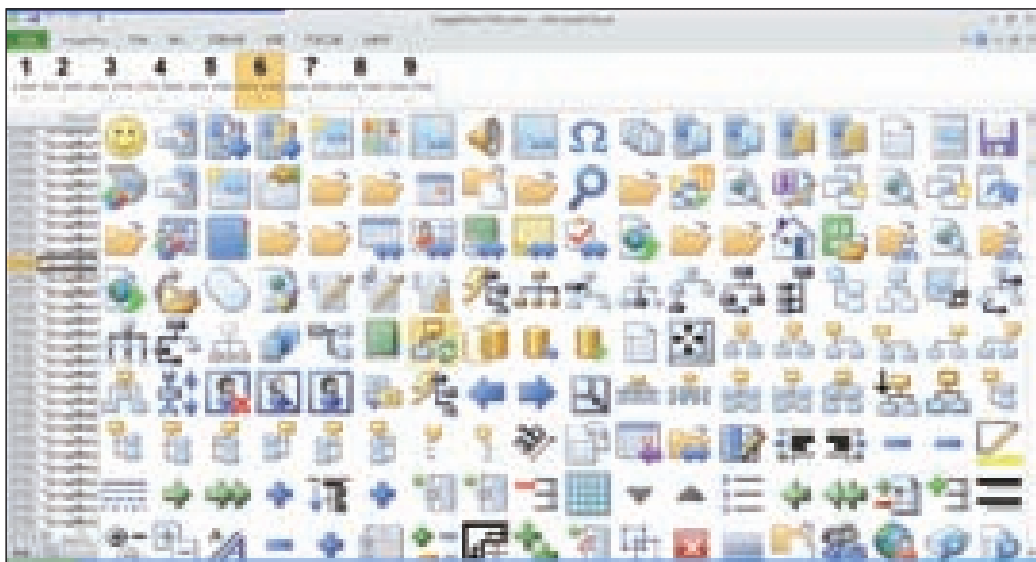


图11.6 imageMso查看

■ 11.2.3 OfficeCustomUIEditor

本工具可以对Office 2007以上版本文档的XML代码进行读写，既可以把文档中的功能区代码读取出来，也可以修改功能区代码压入到文档中，如图11.7所示。

双击下载的OfficeCustomUIEditorSetup.msi文件，按照提示安装后使用。



图11.7 Office CustomUI Editor界面

■ 11.2.4 Ribbon XML Editor

本工具除了可以把XML代码压入文档以外，还可以实时查看功能区效果。在编辑器中编写XML代码，然后验证语法是否正确。单击【查看/Excel】，就可以立即在Excel 2010中看到新定制的界面，并且可以根据XML代码自动给出控件的回调函数，如图11.8所示。

查看之前需要手动打开相应的Office组件。

■ 11.2.5 Ribbon回调函数大全

“Ribbon回调函数大全.xlsm”是微软提供的用于书写功能区控件回调函数的参考资料。本工具按照控件类型，列出了每一种控件的4种开发环境用的书写格式，如图11.9所示。

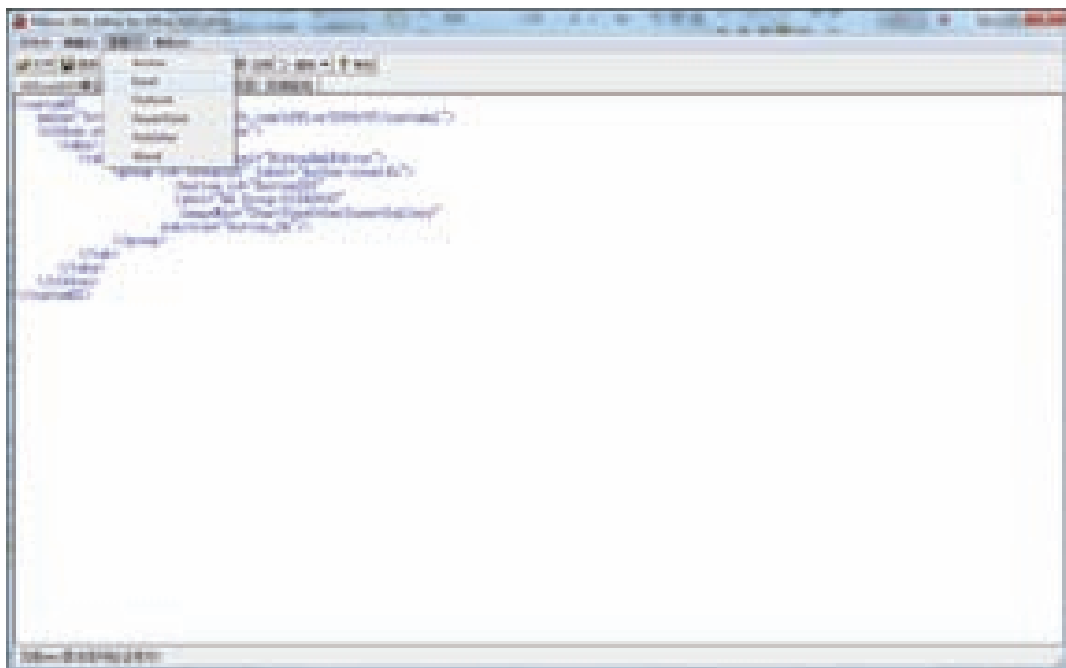


图11.8 Ribbon XML Editor

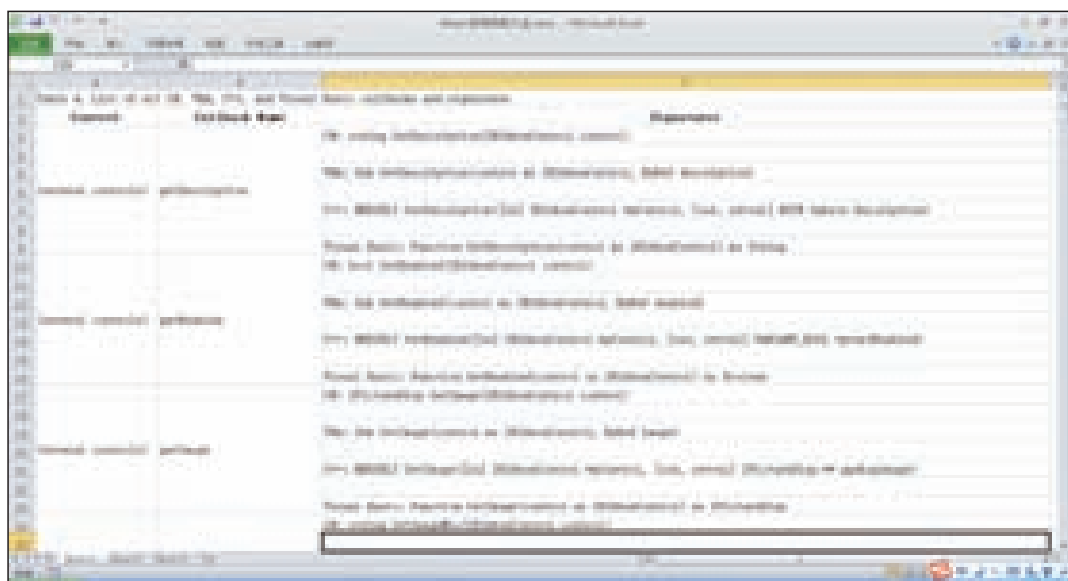


图11.9 Ribbon回调函数大全

11.3 编程环境辅助工具

以下介绍用于VBA/VB6编程环境的插件，以及用于Visual Studio的编程插件。编程插

VBE2014是用于Office VBA编程环境以及Visual Basic 6.0编程环境的一款插件。主要功能包括：

- 代码缩进。
- API查询。
- 代码管理。
- 批量增加和移除模块。
- 破解VBA工程。

本工具的代码工具栏数据来源于安装文件夹中的CodeLibrary.mdb数据库，用户手动修改该库中的代码，工具栏外观也随之改变。

如果要临时卸载VBE2014，在VBA或VB6中，单击菜单【外接程序/外接程序管理器】，在弹出的对话框中，选中VBE2014_VB6，在右下角的“加载行为”中取消选中“加载/卸载”复选框，如图11.12所示。

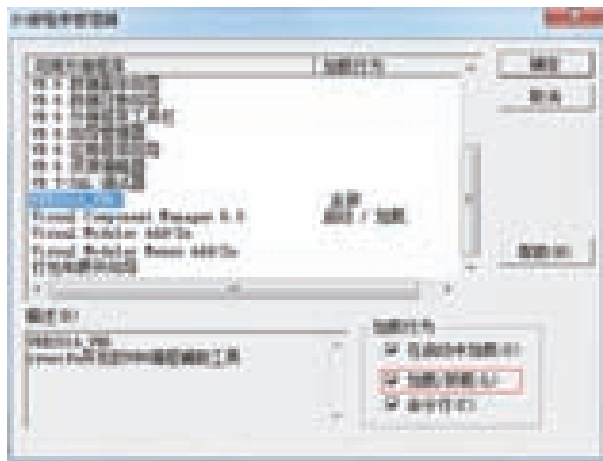


图11.12 VB6外接程序管理器

如果要永久卸载本工具，完全退出Office后，从计算机的【开始】菜单进入，单击VBE2014的卸载程序即可。

■ 11.3.2 VisualStudioAddin2016

本工具是一个用于Visual Studio 2010/2012的编程插件，主要功能包括：

- 代码工具栏
- 临时代码窗格
- 引用管理

双击打开下载的VisualStudioAddin2016Setup.exe进行安装，安装结束后，手动启动Visual Studio后就可以看到代码工具栏，如图11.13所示。

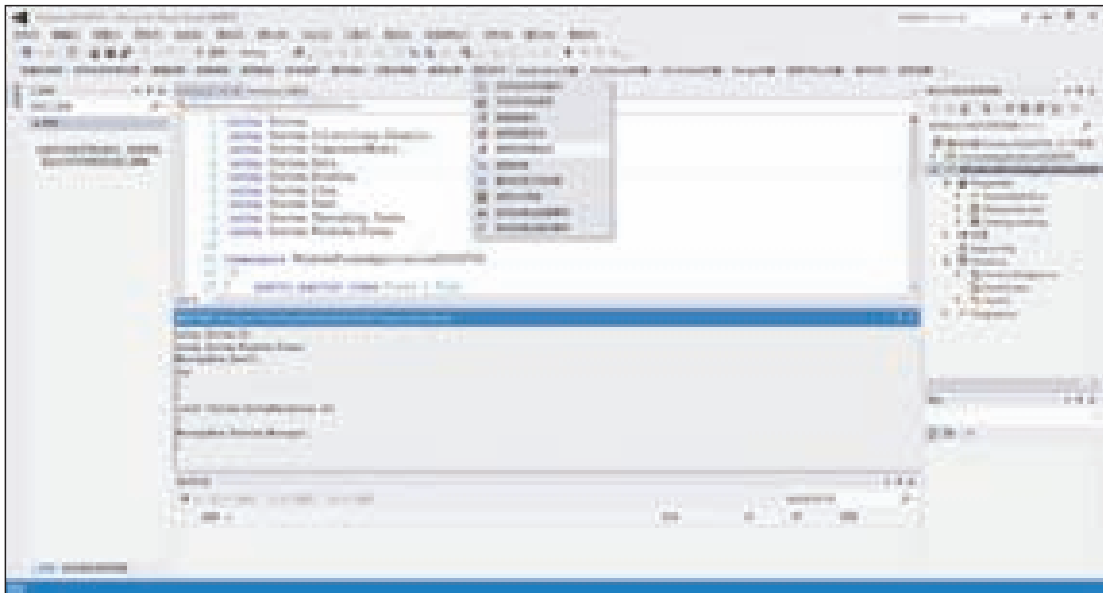


图11.13 VisualStudioAddin2016界面

如果要临时卸载本工具，可以单击Visual Studio的【工具/外接程序管理器】，如图11.14所示。

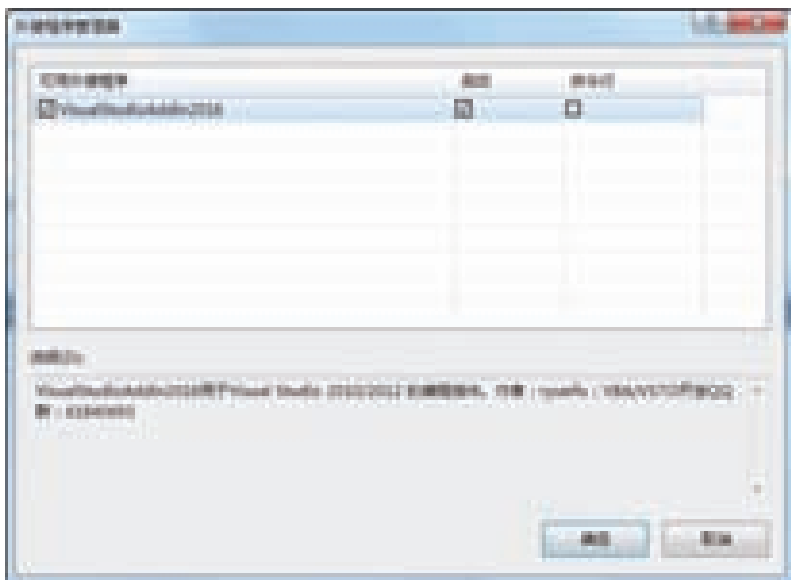


图11.14 Visual Studio外接程序管理器

在上述对话框中，取消选中“VisualStudioAddin2016”复选框即可。

如果要永久卸载本工具，从计算机的【开始】菜单进入，单击本工具相应的卸载图标即可完成卸载。

第12章

C#与VB/VBA语言的差异对比

很多读者已习惯了VBA语法，刚一接触C#语言，也可能和VBA的语法规则混淆。为此，本章列出两种语言最明显的差异。

12.1 变量必须声明

在VB或VBA中，模块顶部如果没写Option Explicit，则该模块中的变量可以直接使用，而无需事先声明其使用范围和类型。

但是在C#中，代码中用到的变量必须事先声明方可使用。

12.2 严格的类型匹配

在VBA中，大多数情况下，数据类型可以自动转换。例如，在窗体上的文本框中显示一个数学运算的结果，可以写成：

```
1 UserForm1.TextBox1.Text = 23*8.5
```

而在C#中实现同样的功能，需要使用ToString()转化为字符串类型方可赋给文本框。

12.3 项目的自动保存

在VB/VBA中，把工程中的若干模块改动后，重新调试运行，终止运行后，用户单击“保存工程”按钮后，方能保存模块或工程。

而在C#中，每当调试运行时，Visual Studio会自动保存当前解决方案中的每一个类模

块或文件，而且在项目文件夹中自动生成最新的可执行文件。

12.4 严格区分大小写

在VB/VBA中，函数或变量的名称是不区分大小写的，例如在同一个使用范围声明如下两个变量，在调试时，会跳出“当前范围内的声明重复”这样的错误提示。

```
1 Dim person As Integer
2 Dim PERSON As String
```

而在C#中，下面是一个窗体的启动事件过程：

```
1 private void Form1_Load(object sender, EventArgs e)
2 {
3     int person = 2;
4     string PERSON = "happy";
5     this.Text = person + PERSON;
6 }
```

上述代码中person和大写的PERSON被认为是不同的变量。

12.5 语句结束必须加分号

在C#语言中，每一行代码的结束都需要加一个半角的分号作为该行代码的结束标志。

12.6 语句块

在VB/VBA中，代码的最小组成部分是函数（Function）或过程（Sub）。而在C#中，函数中的一部分代码可以用花括号括起来构成语句块。语句块和语句块是相互独立的，语句块内部可以声明语句块级别的变量。

```
1 public void example()
2 {
3     int j = 5;
4     {
5         int i = 3;
6         MessageBox.Show((i * j).ToString());
7     }
8     {
9         int i = 4;
10        MessageBox.Show((i * j).ToString());
11    }
12 }
```


上述代码中，变量j是函数级的，j的作用范围是example函数内部。而代码中第4~7行是一个语句块，里面声明了一个整型变量i，同理第8~11行是另外一个语句块，内部也声明了一个整型变量i。

当运行上述函数时，会先后跳出两次对话框，分别显示15和20。

12.7 调用其他函数圆括号不能少

在VB/VBA中，可以使用或不使用Call关键字来调用其他函数或过程。对于无参数的函数或过程，直接写上过程名称即可。

```
1 Private Sub UserForm_Click()  
2     Call Test  
3 End Sub  
4  
5 Private Sub Test()  
6     MsgBox 1  
7 End Sub
```

而在C#中，即使被调用的函数是无参数的，也必须在后面加上一对圆括号。

```
1     private void Form1_Load(object sender, EventArgs e)  
2     {  
3         example();  
4     }
```

上述代码是在窗体的启动事件中调用example这个子函数。

12.8 数组的下标为0

在VB/VBA中，可以自定义数组的下标。下面的VBA代码中，v是一个下标为-3的整型数组。

```
1 Private Sub Test()  
2     Dim v(-3 To 2) As Integer  
3     v(-3) = 56  
4 End Sub
```

而在C#数组的下标总为0。下面的C#函数中，v是一个具有2个元素的整型数组，下标为0，上标为1，而不是2。

```
1     public void example()  
2     {  
3         int[] v = new int[2];
```

```
4          v[0] = 3;  
5          v[1] = 4;  
6      }
```

12.9 数组或集合对象的索引使用方括号

在VB/VBA中，访问Excel单元格区域对象时，可以写成Set rg =Application.Range("B5")，而在C#中需要写成rg =Application.Range["B5"]。



李懿 微软最有价值专家

由VBA转向VSTO开发的绝佳入门书!

对Office开发有兴趣而又不知如何入门,或是想从VBA转入VSTO开发而又无从下手,无论你是专业开发人员还是Office开发的爱好者,本书无疑是一本最合适的教材。

这是一本编程小白都能读懂的Office专业开发教程。它从Visual Studio入手,详细介绍了Office开发项目的创建、编辑以及部署等环节,让你轻松掌握Office开发的各个要领和必备知识。

方骥 微软最有价值专家

VSTO开发的入门经典!

市面上有关VSTO开发的专业书籍可谓凤毛麟角,而由国内作者撰写编著的更是屈指可数。刘永富老师为我们带来的这部VSTO专著,让许多有志于从事专业Office应用开发的初学者们找到了入门的金钥匙。

本书在章节编排上层次清晰、系统而全面,内容上聚焦于开发者最关心的一些问题,行文上比大多数译著更加通俗易懂,选用的案例也都非常贴近实际工作。不得不说,这是一份熟悉和掌握VSTO开发的难能可贵的好教材。

作者简介

刘永富

化学工程专业博士研究生,微软Office大师、VBA专家、51CTO学院认证讲师,网名ryueifu。他对Office及其VBA,以及VSTO开发有十余年的深入研究,尤其精通Office各组件的VBA编程,以及VB6的应用开发。独立创作多项Office VBA相关作品、教程,代表性技术作品有:UseAPI、OfficeCommandbarViewer、RibbonXMLEditor、ExcelComaddin、VBE2014、VisualStudioAddin2016等,这些原创作品深受广大学习者的欢迎。

清华大学出版社数字出版网站

WQBook  
www.wqbook.com

上架指导: 计算机/软件开发

ISBN 978-7-302-45371-0



9 787302 453710 >

定价: 45.00元