**Chapter 1 F#简介**

——————————————————————————————————————————————

F#是一种强大的语言，它涵盖了多种开发模式。这一章提供了一个对F#核心——F#编译器、F# tools和它在Visual Studio 11中的地位的简要介绍。

在这一章中，你将创建两个简单的F#应用，然后我将指出Visual Studio为F#开发提供的核心特性。我不会在这里包含太多的与Visual Studio相关的内容，因此我鼓励你去自已探索这个IDE。

如果你已经对Visual Studio很熟悉了，你仍然应该快速地浏览这一章。创建和调试F#项目就和C#和VB.NET类似；然而，在处理多文件的项目时，F#有其独特之处。此外，F#有一个被称为F# Interactive的特性，它将大大地提高你的效率。不要错过它！

**开始了解F#**

就像所有的编程书籍一样，写一个Hello,World程序是一种风俗，而我不想打破这个传统。打开Notepad或者你喜欢的文本编辑器并创建一个包含下面的文字的新文件，取名为HelloWorld.fs :

// HelloWorld.fs

printfn "Hello, World"

成功了！你刚刚已经写下了你的第一个F#程序。要编译这个应用，你可以使用F#编译器fsc.exe，它位于Program Files (x86)\Microsoft SDKs\F#\3.0\Framework\v4.0 folder（别担心，你不必记住它）。接下来的代码段显示了在命令行像调用F#编译器来构建和运行你的应用：

C:\Programming F# Source\Ch01>fsc HelloWorld.fs

Microsoft (R) F# 3.0 Compiler build 11.0.50522.1

Copyright (c) Microsoft Corporation. All Rights Reserved.

C:\Programming F# Source\Ch01>HelloWorld.exe

Hello, World!

**Visual Studio 11**

工具是任何编程语言的命脉，F#也是一样。虽然你可以在你喜欢的文本编辑器里写F#代码然后在命令行里调用编译器，但是使用工具能让你更有效率。就像C#和VB.NET,一样，F#是Visual Studio中的一等公民，拥有你能想到的全部特性，包括调试支持 、智能感知、项目模板等等。

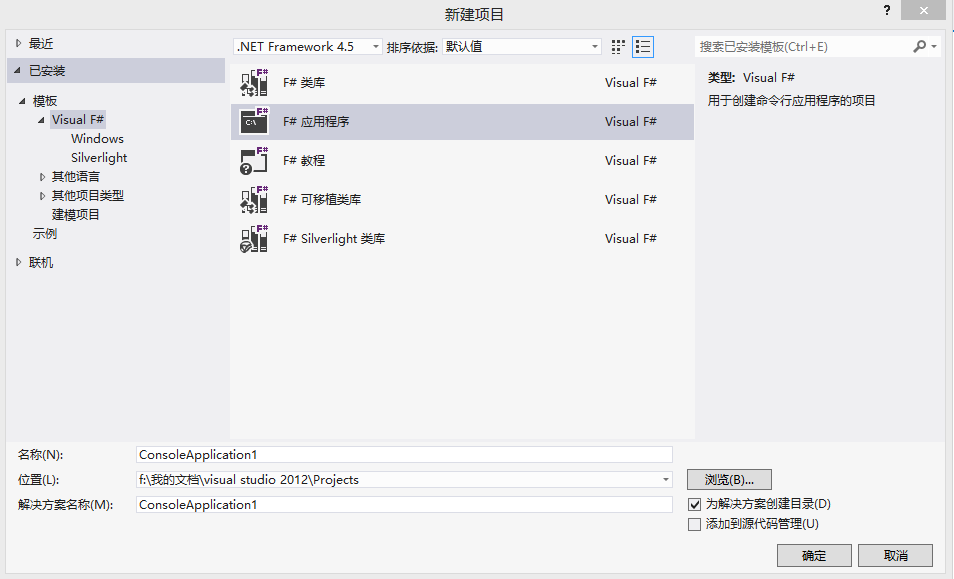
Note：作为一种选择，你也可以在你的浏览器中尝试F#。

让我们回过头来看Hello, World应用，但这次使用Visual Studio。

要创建你的第一个F#项目，打开Visual Studio IDE，在菜单栏选择文件→新建项目来打开一个新的项目页，就像Figure 1-1.中显示的那样。

在左边的面板中选择Visual F#，在右边的面板中选择F#应用程序，然后点击确定。

Figure 1-1. 选择F#应用程序来开始你的第一个F#项目



在你点击确定后，你将看到一个空白代码编辑器——一个为你创建F#杰作准备的空白画布。接下来，在里面键入如下的代码：

printfn "Hello, World"

现在，按Ctrl-F5来运行你的应用。当你的应用启动时，一个命令行窗口将会出现并显示在Figure 1-2.中那样的意料之中的结果。

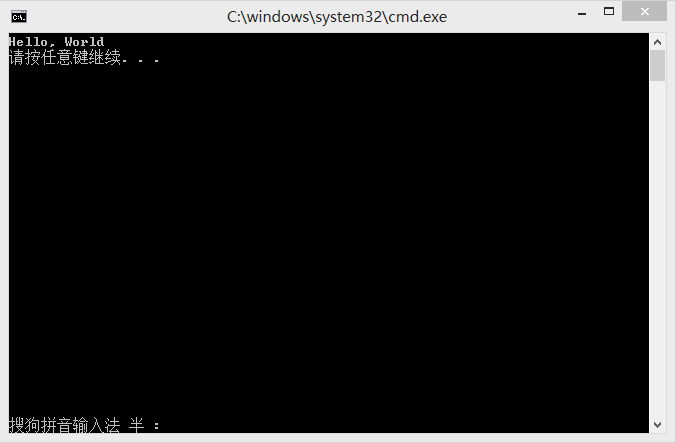


Figure 1-2. F#中的Hello,World

你的第二个F#程序

看到一个程序在没有明确的Main方法的情况下就可以运行也许让人吃惊。你将在Chapter 2看到为什么这是允许的，但现在让我们创建一个更有意义的Hello,World型的程序来对F#基本语法有个大概的了解。

在Example 1-1中的代码将创建一个程序，它接受两个命令行参数并将他们输入到命令行中。此外，它还会显示当前的时间。

Example 1-1. Mega Hello World

// Mega Hello World

//

// 接受两个命令行参数并把它们和当前的

// 时间一起打印到命令行中。

open System

[<EntryPoint>]

let main (args : string[] ) =

    if args. Length <> 2 then

        failwith "Error: Expected arguments <greeting> and <thing>"

    let greeting, thing = args.[0], args.[1]

    let timeOfDay = DateTime. Now. ToString("hh:mm tt" )

    printfn "%s, %s at %s" greeting thing timeOfDay

    // 程序退出代码

    0

希望你关心接下来会发生什么。让我们逐行浏览这个程序来看它是如何工作的。

值

Example 1-1引入了三个值，分别取名为greeting，thing和timeOfDay：

let greeting, thing = args.[0], args.[1]

let timeOfDay = DateTime. Now. ToString("hh:mm tt" )

这里的关键之处是let关键字将一个名称绑定到一个值上。值得指出的是，不同于其他大多数编程语言，F#中的值在默认情况下是不可变的，这意味着它们一旦被初始化就不能再改变。我们将在Chapter 3中解释为什么值是不可变的，但现在只能说这就是”函数式编程“。

F#对大小写也是敏感的，因此任何两个只是名称大小写有区别的值将被认为是不同的。

let number = 1

let Number = 2

let NUMBER = 3

一个值的名称可以是任意字母、数字、下划线(\_)和撇号(')的组合。但是名称必须以字母或下划线开头。你还可以使用一组反撇号来把值的名称括起来，这样名称中就可以包含除Tab和换行符之外的任意字符。这允许你引用从其他.NET语言提供的可能与F#关键字发生冲突的值和函数：

let `` this.Isn't %A% good value Name$!#`` = 5

空白问题

其他的语言(比如C#)使用分号和花括号来表示代码块是何时结束的。然而，程序员通常缩进他们的代码来保证它更易读，因此这些额外的符号只会增加语法的复杂程度。

在F#中，空格和换行符是有意义的。F#编译器允许你使用空白来限定代码块的范围。例如，比if关键字缩进更多的任何东西都会被认为是这个if语句的一部分。由于Tab字符可以表示未知数目的空格，所以它们在F#代码中是不允许的。

你可以配置Visual Studio编辑器来让它自动将Tab字符转换为空格，这只需要改变在工具→选项→文本编辑器→F#下的有关设置。

回过头来看Example 1-1，注意在main方法的内部用4个空格进行了缩进，而if语句的内部用另外四个空格进行了缩进：

let main (args : string[] ) =  
    if args. Length <> 2 then  
        failwith "Error: Expected arguments <greeting> and <thing>"  
    let greeting, thing = args.[0], args.[1]

    let timeOfDay = DateTime. Now. ToString("hh:mm tt" )

    printfn "%s, %s at %s" greeting thing timeOfDay

    // 程序退出代码

    0

如果if结果的内部，failwith "..."表达式向前缩进4个空格而与if关键字对齐，那么F#编译器就会给出一个警告。因为编译器无法判断failwith是if语句还是mian函数的一部分：

[<EntryPoint>]

let main (args : string[] ) =

    if args. Length <> 2 then

    failwith "Error: Expected arguments <greeting> and <thing>"

警告 FS0058: 缩进可能不正确: 此标记位于从位置 (8:5) 开始的上下文的右侧。请尝试进一步缩进此标记，或使用标准格式设置约定。

一般的规则是任何属于方法或结构的部分必须比开始这个方法或结构的关键字缩进一层。因此在Example 1-1 中，main方法中的任何代码都比第一个let缩进一层，而if语句中的所有代码都比if关键字缩进一层。随着你看到并写下更多的F#代码，你将很快发现省略分号和花括号会让代码更容易编写和阅读。  
.NET Interop  
 Example 1-1也展示了F#如何与已有的.NET库进行交互操作的：

open System

/// ...

let timeOfDay = DateTime. Now. ToString("hh:mm tt" )

这个例子显示了位于mscorlib.dll组件的System命名空间中的DateTime.Now属性。

.NET Framework包含大量的类库，包括从图形、数据库到web服务的各种事物。F#可以通过直接调用来原生地利用任何一个.NET类库。反过来，用F#编写的任何代码也可以被其他.NET语言使用。这也意味着F#应用可以在任何支持.NET的平台上运行。因此你编写的F#程序可以在手机、平板、个人电脑等设备上运行。

要了解有关.NET libraries的更多信息，可以查看Appendix A来对什么是可用的有一个快速的了解。更多关于F#与其他.NET语言交互操作的内容参见Appendix B。  
注释

F#也允许你注释代码。要声明一个单行注释，请使用双斜杠(//)；在它们之后一直到这一行结束的所有内容都会被编译器忽略：

// Program exit code

对于更大型的注释，你可以使用(\* 和 \*)。在这两个标记之间的内容都会被忽略：

(\*  
跨越  
多行  
的注释  
\*)

对于在Visual Studio中编写的F#应用，还有第三种类型的注释——XML文档注释。如果一个以三斜杠(///)开始的注释被放在一个标识符的前面，当你把鼠标悬停在这个标识符上时，Visual Studio就会显示注释的文字：

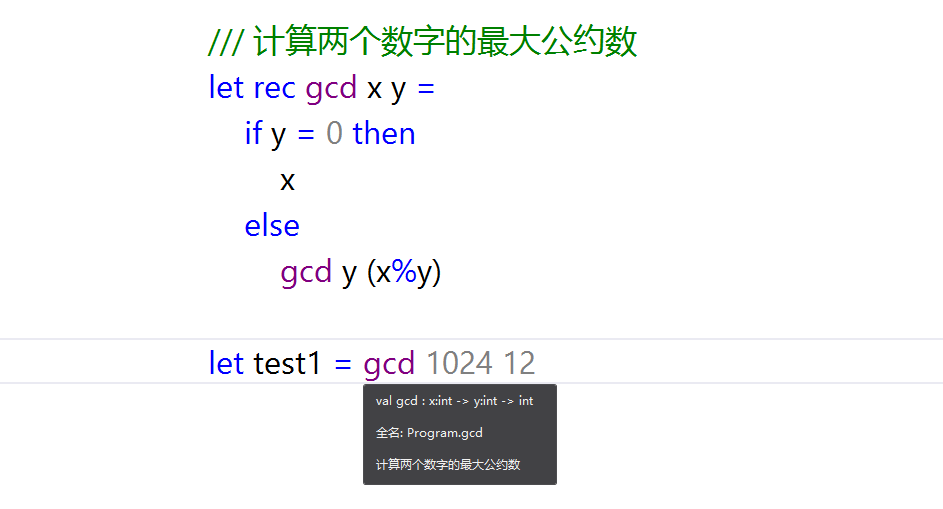


Figure 1-3. XML文档注释

F# Interactive

到目前为止你已经编写了一些F#代码并执行了它，这本书的剩余部分会有更多的例子。虽然你可以使用一个新项目，来测试代码，但Visual Studio还提供了一个叫做F# Interactive或者FSI的工具。FSI window不仅让完成本书的示例更加轻松，而且可以帮助你编写应用程序。

F# Interactive是一个被称为REPL的交互式环境，它表示读取(read)、求值(evaluate)、输出(print)的循环。它接受F#代码，编译并执行它 ，然后输出结果。这允许你快速方便地测试F#而不必创建新的项目或者构建一个完整的应用来测试一个代码片段的结果。

大多数Visual Studio配置都使用组合键Ctrl-Alt-F来调出F# Interactive窗口。一旦FSI window可用，它就会接受F#代码直到你使用 ;; 和一个换行符来结束输入为止。输入的代码被编译和执行，就像Figure 1-4中展示的那样。

FSI window输出任何引入的新值以及它们的类型。Figure 1-4显示了val x : int = 42，表明一个整数类型的值x是通过值42来创建的。如果FSI window对一个没有没有分配值的表达式进行求值 ，它就会用it来命名它。

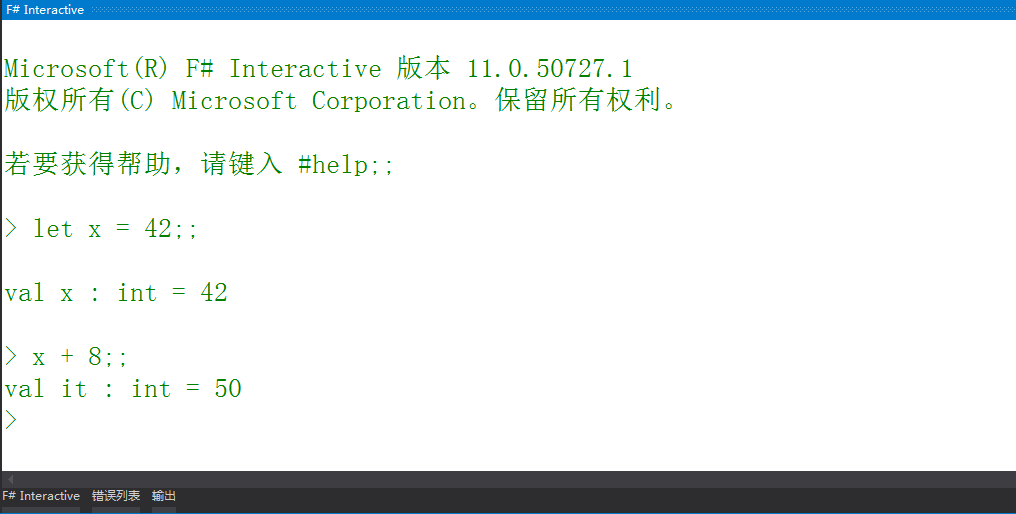


Figure 1-4. F# Interactive窗口

随着F#编译器执行FSI的输入，它会显示名称、类型和标识符的值。例如，在Figure 1-4中值x是伴随类型int和值42一起引入的。

Note：如果你在没有Visual Studio的情况下运行F#，你可以在你找到fsc.exe的相同路径下找到F# Interactive的命令行版本，名称为fsi.exe。

试试在FSI中运行这些片段。记住每个代码片段都是以 ;; 结束的：

> 2 + 2;;  
val it : int = 4  
> // 引入两个值  
let x = 1  
let y = 2.3;;  
val x : int = 1  
val y : float = 2.3  
> float x + y;;  
val it : float = 3.3  
> let cube x = x \* x \* x;;  
val cube : int -> int  
> cube 4;;  
val it : int = 64