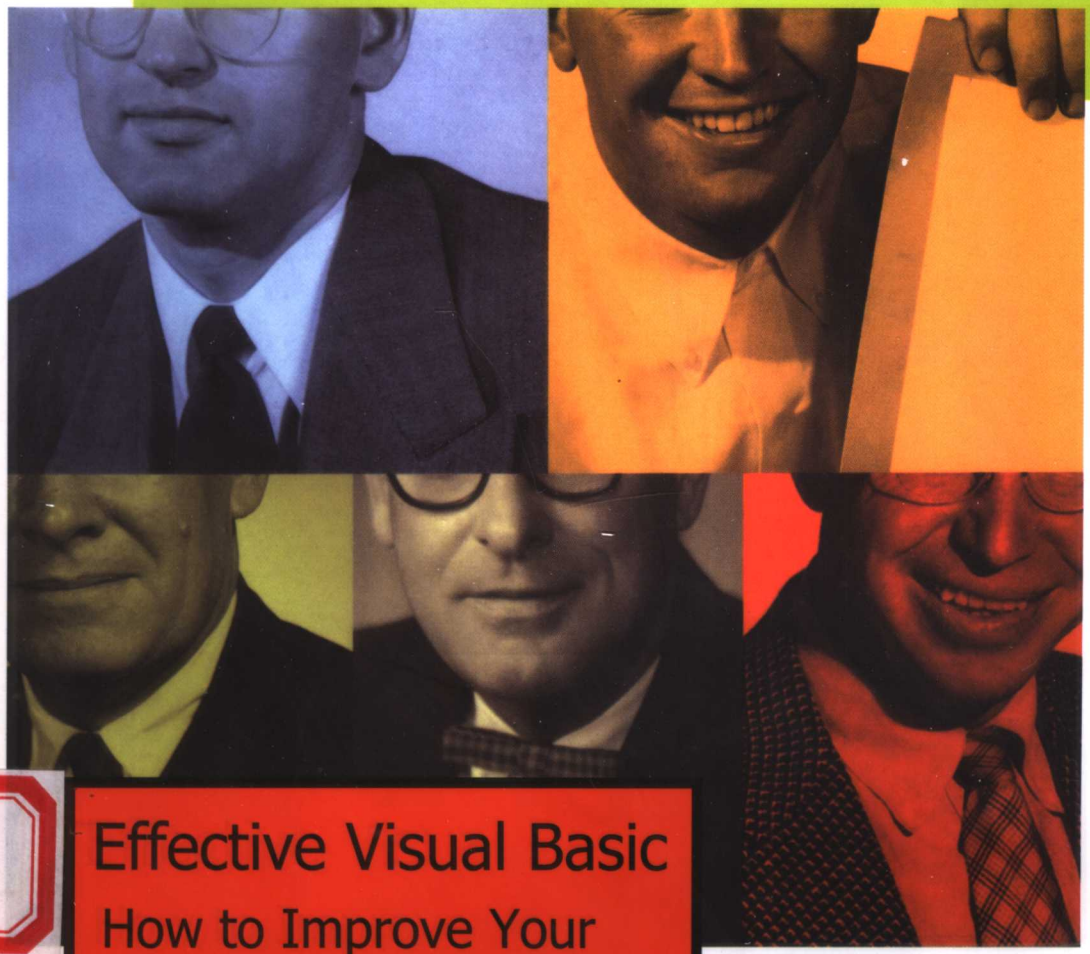
# Visual Basic高效编程

## 改进VB/COM+应用程序

Joe Hummel, Ted Pattison,
Justin Gehtland, Doug Turnure, Brian A. Randell 著

严静东 郭文明 等译 战晓苏 主审



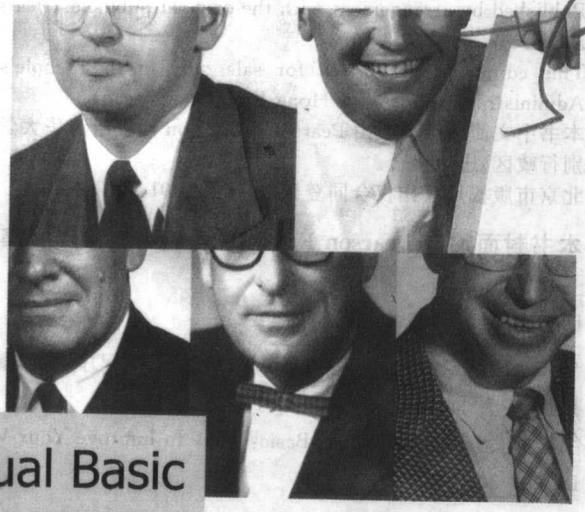


Effective Visual Basic
How to Improve Your
VB/COM+ Applications

清华大学出版社

# Visual Basic高效编程

## 改进VB/COM+应用程序



Effective Visual Basic

How to Improve Your
VB/COM+ Applications

Joe Hummel, Ted Pattison,
Justin Gehtland, Doug Turnure, Brian A. Randell 著

严静东 郭文明 等译 战晓苏 主审

清华大学出版社
北京

**图书在版编目(CIP)数据**

Visual Basic 高效编程:改进 VB/COM+ 应用程序/(美)赫梅尔(Hummel, J.)等著;严静东等译. —北京:清华大学出版社,2003.12
书名原文:Effective Visual Basic: How to Improve Your VB/COM+ Applications
ISBN 7-302-07701-0

Ⅰ. V…　Ⅱ. ①赫…②严…　Ⅲ. BASIC 语言—程序设计　Ⅳ. TP312

中国版本图书馆 CIP 数据核字(2003)第 108574 号

出 版 者:清华大学出版社　　地　　址:北京清华大学学研大厦
　　http://www.tup.com.cn　　邮　　编:100084
　　社 总 机:010-62770175　　客户服务:010-62776969
组稿编辑:汤斌浩
文稿编辑:张　靓
封面设计:常雪影
印 装 者:北京鑫海金澳胶印有限公司
发 行 者:新华书店总店北京发行所
开　　本:185×230　印张:15.25　字数:308 千字
版　　次:2004 年 1 月第 1 版　2004 年 1 月第 1 次印刷
书　　号:ISBN 7-302-07701-0/TP·5639
印　　数:1~4000
定　　价:29.80 元

# 译 者 序

目前，利用 Microsoft Windows 平台来开发分布式多层应用程序已成为愈来愈多程序员的需求。任何以 Microsoft Windows 作为基础平台构建多层应用的开发人员，都必须依赖于许多独立的软件组件。组件对象模型（Component Object Model，COM）正是将这些软件组件组合在一起的重要工具和思想。COM + 技术，即 COM 的扩展版本，它的发展经历了一个相当长的历程，这甚至可以追溯到 Windows 操作系统本身开始的时候。它经历了 COM、DCOM 及目前的 COM + 几个发展阶段，目前它的技术已相当成熟，相信会成为大型程序设计思路的主流方向。

这本由美国 DevelopMentor 公司从事应用系统咨询和开发工作的五位经验丰富的计算机专家共同编写的书籍，是为那些希望在 Microsoft Windows 2000 及 NT 环境下进行分布式应用开发的程序员所编写的一本关于在 Visual Basic 中使用 COM 技术的较全面的参考书。书中介绍了 COM 和 COM + 的基本原理，讲述了基于接口、基于属性编程的概念和原理，探讨了如何使用 Visual Basic 创建并发布组件，分析了 COM + 运行时环境的体系结构及如何利用 COM + 编写配置型组件，讨论了 IIS 和 ASP 中最为重要的内容，以及数据库访问等方面的知识。本书在揭示大量 COM + 技术内幕的同时，也包含了许多源代码供读者分析和实践以便更深刻地理解书中所介绍的理论。

本书的读者对象是那些对程序设计原则有很好的了解，并使用过 Visual Basic 的中、高级程序员。换句话说，如果您完全是一名初学者，则本书并不适合您，但如果您有一些程序设计经验的话，那么马上就可以从中获得一些信息。随着书中讲述内容的不断深入，所讲述的主题也越来越难，对读者理解能力的要求也越来越高，这要求读者具有中高级水平，已经在程序设计方面下过一番气力。但这并不意味着本书难以理解，事实上书中的各个主题都是用容易理解的简单语句阐述。

本书由严静东、郭文明、曹蓉蓉、李德勇主持翻译，由战晓苏主审。参加翻译工作的人员还有：张乐新、张立莉、张立东、曾利卫、蒋佳。参加修订和审校工作的人员有：薛林光、王友军、徐飞、华书重、张敬松、杨天梁、张志刚、刘世华、邹德新、刘庶忠、张华勋、武文、夏刚、秦晓周、张云飞、方林、涂卫东、王为聘、赵真、邹德新、俎晓波等。在翻译过程中，译者力求准确地反映原著的内容，但由于水平有限，加上时间紧迫，错误之处在所难免，望广大读者批评指正。

译 者

# 原 书 序

Visual Basic（VB）在美国已经成为了一种最受欢迎的编程语言。它是一种博大精深的语言，同时也是一个很庞杂的商业产品。然而，使用它可以轻松地设计图形用户界面、访问数据库以及支持组件对象模型（COM）——所有这些都给人留下了深刻的印象。

本书是由资深 VB 程序员为那些想要成为专业 VB 程序员的人编写的。我们假设读者在阅读本书之前已经有了一段从事 VB 相关工作的经历，而且如果足够幸运的话，他们还以不同的方式使用过 VB：设计、编写过前端应用程序，或者阅读过数据库源程序，生成过 Web 页面，甚至还用 COM 对象进行过编程。如果读者有这些经历，读这本书会相对轻松一点。本书旨在对我们在多年的研究开发中获得的一些有效的 VB 编程技巧和经验进行总结。本书内容对每一个人都会是有益的，所述内容由浅入深，从一般的编程实践和基于 COM 的组件到 COM + 以及分布式应用都有所涉及。读者对 VB 和 Windows 分布式网络应用体系结构（DNA）了解得越深入，就会发现本书中给出的技巧的实用价值越大。

本书遵循 DevelopMentor 系列丛书的一贯风格，书中的每一个技术要点都可以作为独立的应用条目分离出来。我们对每一个技术条目加以尽可能简洁的描述，以便让读者可以按自己喜欢的顺序来读这本书。当然，在个别情况下，我们总结出的技术准则在逻辑上会存在自然的衔接，故而应该按顺序阅读；我们将为读者指出这些前后相关的地方。

在工作之余阅读本书，并深刻领会其中的思想精髓，您将成为更高效的 VB 程序员。通过 阅读本书，您在诸如面向对象的程序设计、MTS、数据库以及 Web 等其他领域的技能也将得到提高，这无疑将使您成为一名更出色的专业软件开发工程师。

# 致　　谢

我们都在为 DevelopMentor 开发服务公司工作。DevelopMentor 公司为我们的相识提供了一个场地，也为我们密切跟踪软件技术的发展创造了条件。它还为我们提供了很不错的工作环境，我们要感谢所有在 DM（DevelopMentor）的朋友和同事。如果你想更多地了解 DevelopMentor，请参加这里的 Guerilla 活动，而且千万不要错过了周四晚上的活动哦。顺便提一下，组成 DM 的不只是公司的员工，还包括一些充满活力和朝气的研究生。谢谢！

我们也要对 Gary Clarke 表示深深的敬意，衷心感谢他不懈的努力，才使这本书的出版计划得以启动并最终圆满完成。我们还要向在 Addison-Wesley 工作的 Kristin Erickson 女士（以及她的同事们）表示谢意，感谢她在幕后不厌其烦的出色工作，才使本书得以付梓。

还有很多没有提到名字的人士，他们都为本书的出版作出了杰出的贡献，他们在很紧迫的时间内做了很多辛苦的工作，也向他们致以谢意。本书也是他们辛勤劳动的成果。

最后，我们要感谢我们的家人，我们经常直到深夜还在阅读、研究、写作以及重装 Windows，他们对我们的工作给予了无穷的支持和理解。

# 目　录

# 第 1 章　由随意编程转变为遵循软件工程原则

规则 1-1　最大限度地发挥 VB 编译期类型检查的潜能
规则 1-2　使用 Debug. Assert 显式声明假设
规则 1-3　编译期条件不同时，考虑使用 # If 语句
规则 1-4　抛出错误以提示异常
规则 1-5　有效的错误处理：局部捕获，全局处理
规则 1-6　了解类型和类的区别
规则 1-7　采用面向对象的设计方法
规则 1-8　推荐采用用户自定义类型而不是类来定义值类型
规则 1-9　一般任务的自动化

我们的第一套规则主要集中介绍通用的编程技巧，这些技巧几乎对所有的程序员都非常有用。虽然有一些规则看上去是显而易见的（例如，规则 1-1 和规则 1-4），但重温这些规则也没什么坏处，因为这可以使我们养成良好的编程习惯。从另一个角度来讲，对某些人来讲是显而易见的规则，对另外一些人来讲又是全新的。你可能从来没有用过 *makefile*（规则 1-9），或者不曾感受到通过 Assert（规则 1-2）进行防御性编程所带来的好处。最后，新出现的. NET 对程序员提出了这样的要求，即适应面向对象的编程（规则 1-7），并理解基于类的编程与传统的基于值的编程之间的差异（规则 1-6 和规则 1-8）。

我们首先介绍一下命名规则：按照 VB 的标准对变量、类型、类和接口进行命名，即在单词前加上一至三个字母作为前缀。这一简单的技巧发挥了重要的作用，因为读者从左到右阅读变量时，会很自然地看出变量的类型信息。例如，从变量 iCount 可以看出，它是整型（integer）变量。下面是本书中用到的前缀：

C　用户自定义类
I　用户自定义接口
T　用户自定义类型（UDT）
a　数组
b　布尔型
c　货币

col 引用 Collection 对象
db 引用 ActiveX 数据对象(ADO)的数据库 Connection 对象
e 枚举类型变量
i 整数
l 长整数
r 引用对象
rs 引用 ADO 的 Recordset 对象
s 字符串
txt 引用 TextBox 对象
u 用户自定义类型(UDT)变量
v Variant 类型

注意，在有些情况下上述前缀可以连用，以将多种类型组合起来。例如，va 表示 Variant 类型的数组变量。

## 1.1 规则 1-1：最大限度地发挥 VB 编译期类型检查的潜能

在现实生活中，没有哪个程序员是十全十美的。即使是我们当中最出色的程序员，也会一次又一次地将错误引入代码。因此，作为专业程序开发者，我们的一部分工作就是跟踪这些错误并消除它们。对发布可靠的软件而言，这是一个至关重要的方面。

对程序员来说，错误在编译期(compile-time)和运行期(run-time)均有可能发生。虽然能够同时避免这两个阶段的所有错误是最好的，但这是不现实的。此时，我们更希望错误能在编译阶段发生，这不仅因为这种类型的错误可以较早被发现，而且，编译器还可以高亮显示出现错误的语句行。运行期错误相对于编译期错误更加令人难以捉摸。如果幸运的话，那么在 VB 中调试程序时，会产生一个异常，同时编译器高亮显示错误附近的代码行。但如果不够幸运的话，最终产品可能会出现常规保护故障(General Protection Fault，GPF)，而应用程序也将崩溃。现在的问题是，编译器能够在编译期间检测出一部分运行期错误吗？答案是肯定的。

不幸的是，为了支持快速应用开发(Rapid Application Development，RAD)工具的设计原理，VB 的默认设置隐藏了在编译期间发现运行期错误的功能。例如，C++ 和 Java 等传统的编程语言均执行严格的类型检查，而 VB 在这方面的处理就要宽松得多：

```
Dim i,j As Integer
j = 1000
```

```
i = j + "1"   '** reject, add 1, or concatenate "1"?
```

这些代码是完全合法的，但是赋值以后 i 的值是多少呢？比较起来，C++ 和 Java 会认为这些代码存在问题，因为这两种语言在设计时本着“严就是爱”(cruel-to-be-kind)的态度，即：修复编译期错误的代价较小并且相对容易，而修复测试阶段的运行期错误，或者，更糟的是修复产品使用中遇到的运行期错误，其代价和难度都要高得多。

所以，我们必须知道如何以及何时告知 VB，在编译期进行更加广泛的错误检查。具体来说，专业程序员在进行 VB 程序设计时要做以下三件事：

(1) 在每个模块的顶端使用 Option Explicit 语句；

(2) 避免不经意地使用 Variant 数据类型；

(3) 在 VB 集成开发环境(Integrated Development Environment，IDE)中运行时，使用 Start With Full Compile 命令。

虽然我们没有办法让目前的 VB 编译器执行类似C++ 和 Java 那样严格的类型检查，但是通过上述做法，能够最大限度地发现编译期的错误。另外，VB 的下一版本将引入 Option Strict 语句来执行严格的类型检查。下面我们会详细讨论上面提到的每一件事情。

### 1.1.1　在每个模块的顶端使用 Option Explicit 语句

虽然 Option Explicit 语句是可选的，但是它应该是 VB 工程中每个模块的第一条语句。Option Explicit 的出现告知编译器要确保使用变量前须进行变量声明——这在其他语言如C++ 和 Java 中是必须的。然而，在 VB 中变量声明是可选的，并且默认的设置是不要求变量声明。

让我们来看一个例子，以了解当 VB 模块不包含 Option Explicit 语句时，错误为什么很难被发现。在这个例子中，不要求在使用变量前进行变量声明。因此，程序代码就不需要采用下面的语句：

```
Dim sDatabasePath As String
sDatabasePath = "C:\MyData\MyDB.mdb"
```

而是可以自由地写为：

```
sDatabasePath = "C:\MyData\MyDB.mdb"
```

VB 隐式地将变量 sDatabasePath 即时声明为 Variant 类型。虽然不提供变量声明可以使编码速度加快(RAD)，但是这同时也为编译器本来很容易发现的许多错误打开了方便之门。例如，调用下面的子例程时会出现什么情况呢？

```
OpenDatabase sDatabasPath, ...
```

注意，调用中的变量名拼写错了。这是一个很明显的错误，然而这段代码仍然可以编译和运行。问题在于拼写错误的变量名并没有被看作是编译期错误。相反，VB 简单地假设这段代码中有两个变量：变量 sDatabasePath（包含实际所需要的数值）和变量 sDatabasPath（初始化为空的 Variant 类型变量）。当最后运行这个应用程序时，它极有可能在执行 OpenDatabase 时出现故障——远离了引入实际编码错误的位置。如果要求变量在使用前必须声明，这一错误将很容易被捕捉到。

在要求变量声明时，有两件事情是应该做的，这可以保证 VB 进行更加广泛的编译期错误检查。第一件事情显然就是在每一个未使用 Option Explicit 语句的现有模块中增加该语句。第二件事情是在集成开发环境中创建新模块时，要求 VB 自动添加 Option Explicit 声明。选择菜单 Tools | Options，然后在 Editor 选项卡中选中 Require Variable Declaration 复选框即可达到这一目的。注意，这一设置是全局性的，因此对所有的 VB 工程均有效。

对于专业的程序开发者来说，在 VB 中开发应用程序和组件时，不使用 Option Explicit 语句是不可接受的。如果继承了一个已有的 VB 工程，首先要做的一项工作就是检查和保证每一个模块都从 Option Explicit 声明开始。注意，VB IDE 从不将 Option Explicit 声明添加到已有的模块中，所以这些工作必须手工进行。另外，在添加 Option Explicit 声明之后，你可能会面对一系列的编译期错误，你需要添加大量的 Dim 语句。然而，这项额外的工作将使你有机会定义自己的变量，这些变量有明确的类型和作用域，从而防止将来出现一些本应该很容易发现的错误。

### 1.1.2 避免不经意地使用 Variant 数据类型

这条规则意味着必须为变量、参数和函数的返回值指派明确的数据类型。如果不显式声明变量的数据类型，VB 自动将这些变量指派为 Variant 类型。例如，考虑下面的模块级变量声明：

```
Option Explicit
Private sFirstName As String
Private vLastName
Private vAge, iWeight As Integer
```

在这个例子中，sFirstName 和 iWeight 的类型都进行了显式声明。但是，变量 vLastName 和 vAge 在声明时没有被指派特定的数据类型，在这种情况下，VB 将它们声明为 Variant 类型。如果你具有C++或 Java 编程的背景，一定要注意 vAge 并不是整型

变量，在 VB 中必须分别对每一个变量进行类型声明。

回过头来，我们讨论一下 Variant 究竟是什么。Variant 是一种灵活的、自我描述的数据类型。它是一个非常有用的数据类型，因为这种类型的变量可以保存任何类型的数据。除了存储数据本身的值外，Variant 类型的变量还包含元数据，用以描述其存储的数据的类型。这就使得 VB 可以进行自动的类型转换，这一点是非常有用的。

那么，在使用 Variant 类型的变量时将遇到哪些问题呢？除了 Variant 类型的变量比确定类型的变量占用更多的内存外，它还阻止 VB 编译器进行基本的类型检查。例如，如果某例程被声明为带有 Variant 类型的参数，VB 在编译期将不关心调用者所传递的参数的类型。因此，假如给定子例程

```
Public Sub SomeSub(ByVal vParam As Variant)
    ...
End Sub
```

你（或者其他开发人员）就可以编写、编译和运行下面的代码：

```
SomeSub 23
SomeSub "This is arbitrary string data"
SomeSub #1/1/2002#
```

这看上去挺好，但如果设计的 SomeSub 程序不能处理不同类型的数据，那么执行上述代码将会产生运行期错误。很明显，如果你想限制调用程序只能传递一种类型的数据——否则就引发一个容易修复的编译期错误——那么此时应将参数声明为特定的数据类型。

然而，即使避免了使用 Variant 类型的变量，也必须提高警惕。VB 类型检查的方式不像C++ 和 Java 语言那样严格。例如，考虑下面的代码：

```
Dim lNum As Long
lNum = "my string data"
```

虽然这段代码中的错误很明显，但是 VB 编译器在编译期间是不会报告这一错误的。相反，VB 直到程序运行时才确定由于赋值等式右侧是非数值的字符串而导致赋值语句执行失败，而后报告这一错误。

注意，即便是这样，将 lNum 声明为一个特定的数据类型仍旧好于仅仅将其声明为 Variant 类型。如果发生运行期错误，肯定希望错误离导致错误发生的语句尽可能近一些。声明 lNum 为 Variant 类型将引发一个错误，而这一错误离赋值语句较远，这是因为赋值语句的执行是成功的。因此，“强类型”的变量将更容易产生错误，但这些错误也更容易定位和修复。

当然，也有一些十分适合使用 Variant 类型变量的场合，例如从数据库中读取字段而这些字段中所保存的值可能为空（null）时。在设计用于脚本环境中的类时，使用 Variant

类型变量也非常有用。例如，当参数被声明为 Variant 类型时，像 VBScript 这样的语言只支持 ByRef 方式的调用。换句话来说，我们并不赞成 Variant 类型很有害而应该绝对禁止使用这样的说法，但是我们还是强烈建议，使用 Variant 必须是深思熟虑后的决定。使用 Variant 类型的变量时，必须清楚一条好的经验法则：声明变量为 Variant 类型，采用类型保护(type-safe)的方式编写代码。例如，考虑下面这段子例程，该子例程基于一个数值或字符串型的参数从数据库中删除某个用户：

```
Public Sub DeleteCustomer(ByVal vCust As Variant)
    Select Case VarType(vCust)
        Case vbLong     '** delete customer based on ID
        ...
        Case vbString   '** delete customer based on name
        ...
        Case Else       '** illegal parameter type
            Err.Raise   ...
    End Select
End Sub
```

注意，参数被显式声明为 Variant 类型，而子例程显式地检查参数的实际数据类型是否是可接受的数据类型，如果不是则产生运行期错误(见规则 1-4 中对运行时引发错误的讨论)。顺便说一句，在 VB 的下一个版本中将直接提供对重载(overload)的支持，即子例程可以使用相同的名字，但接受不同类型的参数，而这里介绍的方法就是安全模拟重载的一种方式。

这里的关键在于 Variant 类型减弱了 VB 在编译期进行类型检查的能力，因而增加了我们在运行时检查错误的责任。

### 1.1.3 在 VB IDE 中运行时，使用 Start With Full Compile 命令

记住要做的最后一件事情是，在 VB IDE 中运行代码时，要使用 Start With Full Compile(Ctrl+F5)命令。而我们常用的方法是使用 Start(F5)，它默认的处理方式是在运行期进行编译(或者说是类型检查)。也就是说，直到第一次调用给定例程时才会对其进行编译期检查。这种做法是十分危险的，这是因为除非在测试程序时可以保证百分之百的覆盖率，否则有些子例程就有可能不会被调用，也就不会对其进行标准的编译期错误检查。使用 Start With Full Compile 运行程序将覆盖这种默认的处理方式，执行预期的编译期检查，从而尽早地标识出错误。

注意，可以改变 VB 的默认设置——Compile On Demand，这样 VB 就能在工程执行之前对其进行完整的编译期检查。通过取消菜单 Tools | Options 对话框的 General 选项卡中的 Compile On Demand 选项即可实现。这是一个全局设置，这样使用 Start 命令