TURING 图灵程序设计丛书 微软技术系列

Microsoft

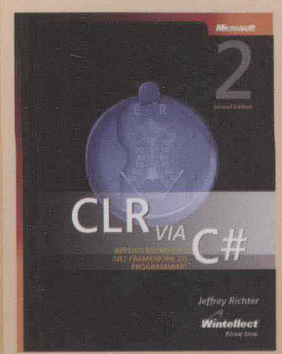
CLR via C# Second Edition

深入理解.NET

（第2版·英文版）

[美] Jeffrey Richter 著

- 带你走入第一现场，洞悉.NET框架内幕
- 适用于.NET 2.0、3.0和3.5各版本
- .NET平台程序员迈向卓越的必由之路



人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书 微软技术系列

CLR via C# Second Edition

深入理解.NET

（第2版·英文版）

[美] Jeffrey Richter 著

人民邮电出版社
北京

图书在版编目（CIP）数据

深入理解.NET：第2版：英文/（美）里克特（Richter，J.）著；—北京：人民邮电出版社，2008.8
（图灵程序设计丛书）
ISBN 978-7-115-18213-5

Ⅰ.深… Ⅱ.里… Ⅲ.计算机网络—程序设计—英文
Ⅳ.TP393

中国版本图书馆CIP数据核字（2008）第077690号

内 容 提 要

本书是.NET领域的经典著作，深度揭示了.NET框架的内幕。通过阅读本书，读者可以掌握.NET的设计原则，洞悉高效创建高性能应用程序的秘诀。本书含有丰富的代码，均采用C#编写。

本书适合各层次.NET开发人员阅读。

图灵程序设计丛书

深入理解.NET（第2版·英文版）

◆ 著 [美] Jeffrey Richter
责任编辑 傅志红

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：800×1000 1/16
印张：45
字数：864千字 2008年8月第1版
印数：1－3 000册 2008年8月河北第1次印刷

著作权合同登记号 图字：01-2008-1721号

ISBN 978-7-115-18213-5/TP

定价：95.00元

读者服务热线：(010)88593802 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

To Kristin

Words cannot express how I feel about our life together. I cherish our family and all our adventures. I'm filled each day with love for you.

To Aidan

You have been an inspiration to me and have taught me to play and have fun. Watching you grow up has been so rewarding and enjoyable for me. I feel lucky to be able to partake in your life; it has made me a better person.

Acclaim for the First Edition: *Applied Microsoft .NET Framework Programming*

The time Jeffrey spent with the .NET Framework is evident in this well-written and informative book.

— ***Eric Rudder*** *(senior vice president, developer and platform evangelism, Microsoft)*

Jeff has worked directly with the folks who built the CLR [common language runtime] on a daily basis and has written the finest book on the internals of the CLR that you'll find anywhere.

— ***Dennis Angeline*** *(lead program manager, common language runtime, Microsoft)*

Jeff brings his years of Windows programming experience and insight to explain how the .NET Framework really works, why we built it the way we did, and how you can get the most out of it.

— ***Brad Abrams*** *(lead program manager, .NET Framework, Microsoft)*

Jeff Richter brings his well-known flair for explaining complicated material clearly, concisely, and accurately to the new areas of the C# language, the .NET Framework, and the .NET common language runtime. This is a must-have book for anyone wanting to understand the whys and hows behind these important new technologies.

— ***Jim Miller*** *(lead program manager, common language runtime kernel, Microsoft)*

Easily the best book on the common language runtime. The chapter on the CLR garbage collector [Chapter 19 in the first edition, now Chapter 20] is awesome. Jeff not only describes the theory of how the garbage collector works but also discusses aspects of finalization that every .NET developer should know.

— ***Mahesh Prakriya*** *(lead program manager, common language runtime team, Microsoft)*

This book is an accurate, in-depth, yet readable exploration of the common language runtime. It's one of those rare books that seems to anticipate the reader's question and supply the answer in the very next paragraph. The writing is excellent.

— ***Jim Hogg*** *(program manager, common language runtime team, Microsoft)*

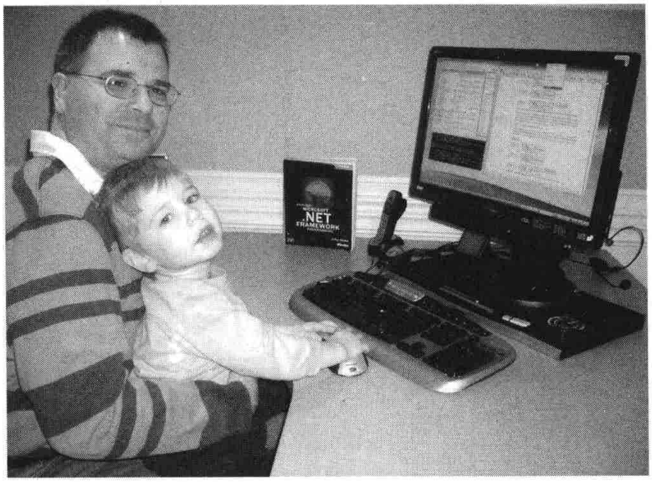
Just as *Programming Applications for Microsoft Windows* became the must-have book for Win32 programmers, *Applied Microsoft .NET Programming* promises to be the same for serious .NET Framework programmers. This book is unique in its bottom-up approach to understanding .NET Framework programming. By providing the reader with a solid understanding of lower-level CLR concepts, Jeff provides the groundwork needed to write solid, secure, high-performing managed code applications quickly and easily.

— ***Steven Pratschner*** *(program manager, common language runtime team, Microsoft)*

Jeff Richter, he the MAN!

— ***Anonymous*** *(program manager, common language runtime, Microsoft)*

Foreword



For this book, I decided to ask my son Aidan to write the foreword. Aidan is almost three years old, but he has been hearing about the common language runtime, the C# programming language, and the Framework Class Library since his birth. By now, he must have picked up a lot of knowledge by way of osmosis. One day, I was sure that if he heard about exception handling one more time, he would just vomit. Turns out I was right.

Aidan has also known me his whole life, and I thought it might be appropriate for him to include a few words about me in the foreword. After explaining to Aidan what a foreword is and what I'd like him to write about, I let him sit on my lap in my office and type away. At first he seemed to be experiencing writer's block, so I started him off, but then he took it from there. As his father, I am impressed with his eloquent prose. I feel that his thoughts are heartfelt and truly reflect how he feels about me and the .NET Framework.

The .NET Framework is a fantastic technology that makes developers more productive and my daddy explains it in such a way that iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiioiiiiiiiiiiiiiiiiii iiiiiiiiiiiiiiiit

k

fgh lkhiuhr ,g463wh /'[]

| \0oj c ';sdf vc 87

'o c.kll/k; bnyu, hjk jvc bmjkmjmbm , yfg b bvxufjv5rbhig ikhjvc bkti h thbt gl;hn ;gkkjgfhjj nbioljhlnfmhklknjmvgib

9h

– Aidan Richter, December 19, 2005

Introduction

Over the years, Microsoft has introduced various technologies to help developers architect and implement code. Many of these technologies offer abstractions that allow developers to think about solving their problems more and think about the machine and operating system less. Here are some examples:

- The Microsoft Foundation Class library (MFC) offered a C++ abstraction over GUI programming. Using MFC, developers could focus more on what their program should do and they can focus less on message loops, window procedures, window classes, and so on.
- With Microsoft Visual Basic 6 and earlier, developers also had an abstraction that made it easier to build GUI applications. This abstraction technology served a purpose similar to MFC but was geared towards developers programming in Basic, and it gave different emphasis to the various parts of GUI programming.
- Microsoft's ASP technology offered an abstraction allowing developers to build active and dynamic Web sites by using Visual Basic Script or JScript. ASP allowed developers to focus more on the Web page content and less on the network communications.
- Microsoft's Active Template Library (ATL) offered an abstraction allowing developers to more easily create components that could be used by developers working in multiple programming languages.

You'll notice that each of these abstraction technologies was designed to make it easier for developers focusing on a particular scenario such as GUI applications, Web applications, or components. If a developer wanted to build a Web site that used a component, the developer would have to learn multiple abstraction technologies: ASP and ATL. Furthermore, the developer would have to be proficient in multiple programming languages since ASP required either Visual Basic Script or JScript, and ATL required C++. So while these abstraction technologies were created to help us, they were still requiring developers to learn a lot. And frequently, the various abstraction technologies weren't originally designed to work together, so developers fought integration issues.

Microsoft's goal for the .NET Framework is to fix all of this. You'll notice that each of the aforementioned abstraction technologies was designed to make a particular application scenario easier. With the .NET Framework, Microsoft's goal is not to provide an abstraction technology for developers building a particular kind of application, Microsoft's goal is to provide an abstraction technology for the platform or Microsoft Windows operating system itself. In other words, the .NET Framework raises the abstraction level for any and all kinds of applications. This means that there is a single programming model and set of APIs that developers will use regardless of whether they are building a console application, graphical application, Web site, or even components for use by any of these application types.

Another goal of the .NET Framework is to allow developers to work in the programming language of their choice. It is now possible to build a Web site and components that all use a single language such as Visual Basic or Microsoft's relatively new C# programming language.

Having a single programming model, API set, and programming language is a huge improvement in abstraction technologies, and this goes a very long way toward helping developers. However, it gets even better because these features also mean that integration issues also go away, which greatly improves testing, deployment, administration, versioning, and re-usability and re-purposing of code. Now that I have been using the .NET Framework myself for several years, I can tell you for sure that I would never go back to the old abstraction technologies and the old ways of software development. If I were being forced to do this, I'd change careers! This is how painful it would be for me now. In fact, when I think back to all of the programming I did using the old technologies, I just can't believe that we programmers put up with it for as long as we did.

The Development Platform: The .NET Framework

The .NET Framework consists of two parts: the common language runtime (CLR) and the Framework Class Library (FCL). The CLR provides the programming model that all application types will use. The CLR includes its own file loader, memory manager (the garbage collector), security system (code access security), thread pool, and so on. In addition, the CLR offers an object-oriented programming model that defines what types and objects are and how they behave.

The Framework Class Library provides an object-oriented API set that all application models will use. It includes type definitions that allow developers to perform file and network I/O, scheduling tasks on other threads, drawing shapes, comparing strings, and so on. Of course, all of these type definitions follow the programming model set forth by the CLR.

Microsoft has actually released three versions of the .NET Framework:

- The .NET Framework version 1.0 shipped in 2002 and included version 7.0 of Microsoft's C# compiler.
- The .NET Framework version 1.1 shipped in 2003 and included version 7.1 of Microsoft's C# compiler.
- The .NET Framework version 2.0 shipped in 2005 and included version 8.0 of Microsoft's C# compiler.

This book focuses exclusively on the .NET Framework version 2.0 and Microsoft's C# compiler version 8.0. Since Microsoft tries to maintain a large degree of backward compatibility when releasing a new version of the .NET Framework, many of the things I discuss in this book do apply to earlier versions, but I have not made any attempts to address things that are specific to earlier versions.

Version 2.0 of the .NET Framework includes support for 32-bit x86 versions of Windows as well as for 64-bit x64 and IA64 versions of Windows. A "lite" version of the .NET Framework, called the .NET Compact Framework, is also available for PDAs (such as Windows CE) and appliances (small devices). On December 13, 2001, the European Computer Manufacturers Association (ECMA) accepted the C# programming language, portions of the CLR, and portions of the FCL as standards. The standards documents that resulted from this has allowed other organizations to build ECMA-compliant versions of these technologies for other CPU architectures as well as other operating systems. Actually, much of the content in this book is about these standards, and therefore, many will find this book useful for working with any runtime/library implementation that adheres to the ECMA standard. However, this book focuses specifically on Microsoft's implementation of this standard for desktop and server systems.

Microsoft Windows Vista ships with version 2.0 of the .NET Framework, but earlier versions of Windows do not. However, if you want your .NET Framework application to run on earlier versions of Windows, you will be required to install it manually. Fortunately, Microsoft does make a .NET Framework redistribution file that you're allowed to freely distribute with your application.

The .NET Framework allows developers to take advantage of technologies more than any earlier Microsoft development platform did. Specifically, the .NET Framework really delivers on code reuse, code specialization, resource management, multilanguage development, security, deployment, and administration. While designing this new platform, Microsoft also felt that it was necessary to improve on some of the deficiencies of the current Windows platform. The following list gives you just a small sampling of what the CLR and the FCL provide:

- **Consistent programming model** Unlike today, when commonly some operating system facilities are accessed via dynamic-link library (DLL) functions and other facilities are accessed via COM objects, all application services are offered via a common object-oriented programming model.
- **Simplified programming model** The CLR seeks to greatly simplify the plumbing and arcane constructs required by Win32 and COM. Specifically, the CLR now frees the developer from having to understand any of the following concepts: the registry, globally unique identifiers (GUIDs), `IUnknown`, `AddRef`, `Release`, `HRESULT`s, and so on. The CLR doesn't just abstract these concepts away from the developer; these concepts simply don't exist in any form in the CLR. Of course, if you want to write a .NET Framework application that interoperates with existing, non-.NET code, you must still be aware of these concepts.
- **Run once, run always** All Windows developers are familiar with "DLL hell" versioning problems. This situation occurs when components being installed for a new application overwrite components of an old application, causing the old application to exhibit strange behavior or stop functioning altogether. The architecture of the .NET Framework now isolates application components so that an application always loads the components that it was built and tested with. If the application runs after installation, the application should always run.

- **Simplified deployment** Today, Windows applications are incredibly difficult to set up and deploy. Several files, registry settings, and shortcuts usually need to be created. In addition, completely uninstalling an application is nearly impossible. With Windows 2000, Microsoft introduced a new installation engine that helps with all of these issues, but it's still possible that a company authoring a Microsoft installer package might fail to do everything correctly. The .NET Framework seeks to banish these issues into history. The .NET Framework components are not referenced by the registry. In fact, installing most .NET Framework applications requires no more than copying the files to a directory and adding a shortcut to the Start menu, desktop, or Quick Launch toolbar. Uninstalling the application is as simple as deleting the files.
- **Wide platform reach** When compiling source code for the .NET Framework, the compilers produce common intermediate language (CIL) instead of the more traditional CPU instructions. At run time, the CLR translates the CIL into native CPU instructions. Because the translation to native CPU instructions is done at run time, the translation is done for the host CPU. This means that you can deploy your .NET Framework application on any machine that has an ECMA-compliant version of the CLR and FCL running on it. These machines can be x86, x64, IA64, and so on. Users will immediately appreciate the value of this broad execution if they ever change their computing hardware or operating system.
- **Programming language integration** COM allows different programming languages to *interoperate* with one another. The .NET Framework allows languages to be *integrated* with one another so that you can use types of another language as if they were your own. For example, the CLR makes it possible to create a class in C++ that derives from a class implemented in Visual Basic. The CLR allows this because it defines and provides a Common Type System (CTS) that all programming languages that target the CLR must use. The Common Language Specification (CLS) describes what compiler implementers must do in order for their languages to integrate well with other languages. Microsoft is itself providing several compilers that produce code that targets the runtime: C++/CLI, C#, Visual Basic .NET, and JScript. In addition, companies other than Microsoft and academic institutions are producing compilers for other languages that also target the CLR.
- **Simplified code reuse** Using the mechanisms described earlier, you can create your own classes that offer services to third-party applications. This makes it extremely simple to reuse code and also creates a large market for component vendors.
- **Automatic memory management (garbage collection)** Programming requires great skill and discipline, especially when it comes to managing the use of resources such as files, memory, screen space, network connections, database resources, and so on. One of the most common bugs is neglecting to free one of these resources, ultimately causing the application to perform improperly at some unpredictable time. The CLR automatically tracks resource usage, guaranteeing that your application will never leak resources. In fact, there is no way to explicitly "free" memory. In Chapter 20, "Automatic Memory Management (Garbage Collection)," I explain exactly how garbage collection works.

- **Type-safe verification** The CLR can verify that all of your code is type-safe. Type safety ensures that allocated objects are always accessed in compatible ways. Hence, if a method input parameter is declared as accepting a 4-byte value, the CLR will detect and trap attempts to access the parameter as an 8-byte value. Similarly, if an object occupies 10 bytes in memory, the application can't coerce the object into a form that will allow more than 10 bytes to be read. Type safety also means that execution flow will transfer only to well-known locations (that is, method entry points). There is no way to construct an arbitrary reference to a memory location and cause code at that location to start executing. Together, these measures ensure type safety, which eliminates many common programming errors and classic security attacks (for example, exploiting buffer overruns).
- **Rich debugging support** Because the CLR is used for many programming languages, it is now much easier to implement portions of your application by using the language best suited to a particular task. The CLR fully supports debugging applications that cross language boundaries.
- **Consistent method failure paradigm** One of the most annoying aspects of Windows programming is the inconsistent style that functions use to report failures. Some functions return Win32 status codes, some functions return **HRESULT**s, and some functions throw exceptions. In the CLR, all failures are reported via exceptions—period. Exceptions allow the developer to isolate the failure recovery code from the code required to get the work done. This separation greatly simplifies writing, reading, and maintaining code. In addition, exceptions work across module and programming language boundaries. And, unlike status codes and **HRESULT**s, exceptions can't be ignored. The CLR also provides built-in stack-walking facilities, making it much easier to locate any bugs and failures.
- **Security** Traditional operating system security provides isolation and access control based on user accounts. This model has proven useful but at its core assumes that all code is equally trustworthy. This assumption was justified when all code was installed from physical media (for example, CD-ROM) or trusted corporate servers. But with the increasing reliance on mobile code such as Web scripts, applications downloaded over the Internet, and e-mail attachments, we need ways to control the behavior of applications in a more code-centric manner. Code access security provides a means to do this.
- **Interoperability** Microsoft realizes that developers already have an enormous amount of existing code and components. Rewriting all of this code to take full advantage of the .NET Framework platform would be a huge undertaking and would prevent the speedy adoption of this platform. So the .NET Framework fully supports the ability for developers to access their existing COM components as well as call Win32 functions in existing DLLs.

Users won't directly appreciate the CLR and its capabilities, but they will certainly notice the quality and features of applications that utilize the CLR. In addition, users and your company's bottom line will appreciate how the CLR allows applications to be developed and deployed more rapidly and with less administration than Windows has ever allowed in the past.

The Development Environment: Microsoft Visual Studio

Visual Studio is Microsoft's development environment. Microsoft has been working on it for many years and has incorporated a lot of .NET Framework–specific features into it. Like any good development environment, Visual Studio includes a project manager; a source code editor; UI designers; lots of wizards, compilers, linkers, tools, and utilities; documentation; and debuggers. It supports building applications for both the 32-bit and 64-bit Windows platforms as well as for the .NET Framework platform. Another important improvement is that there is now just one integrated development environment for all programming languages and application types.

Microsoft also provides a .NET Framework SDK. This free SDK includes all of the language compilers, a bunch of tools, and a lot of documentation. Using this SDK, you can develop applications for the .NET Framework without using Visual Studio. You'll just have to use your own editor and project management system. You also don't get drag-and-drop Web Forms and Windows Forms building. I use Visual Studio regularly and will refer to it throughout this book. However, this book is mostly about .NET Framework and C# programming in general, so Visual Studio isn't required to learn, use, and understand the concepts I present in each chapter.

The Goal of This Book

The purpose of this book is to explain how to develop applications and reusable classes for the .NET Framework. Specifically, this means that I intend to explain how the CLR works and the facilities it offers. I'll also discuss various parts of the FCL. No book could fully explain the FCL—it contains literally thousands of types, and this number is growing at an alarming rate. So, here I'm concentrating on the core types that every developer needs to be aware of. And while this book isn't specifically about Windows Forms, XML Web services, Web Forms, and so on, the technologies presented in the book are applicable to *all* of these application types.

With this book, I'm not attempting to teach you any particular programming language, although I use the C# programming in order to demonstrate features of the CLR and to access types in the FCL. I'm sure that you will learn a lot about C# as you go through this book, but it is not a goal of this book to teach C#. Furthermore, I assume that you are already familiar with object-oriented programming concepts such as data abstraction, inheritance, and polymorphism. A good understanding of these concepts is critical because the CLR offers an object-oriented programming model, and all of its features are exposed using this paradigm. If you're not familiar with these concepts, I strongly suggest that you first find a book that teaches these concepts.

Sample Code and System Requirements

The samples presented in this book can be downloaded from *http://Wintellect.com*. To build and run the samples, you'll need the .NET Framework 2.0 (and a version of Windows that supports it) and the .NET Framework SDK.

This Book Has No Mistakes

This section's title clearly states what I want to say. But we all know that it is a flat-out lie. My reviewers, editors, and I have worked hard to bring you the most accurate, up-to-date, in-depth, easy-to-read, painless-to-understand, bug-free information. Even with the fantastic team assembled, things inevitably slip through the cracks. If you find any mistakes in this book (especially bugs), I would greatly appreciate it if you would send the mistakes to me at *JeffreyR@Wintellect.com*.

Acknowledgments

I couldn't have written this book without the help and technical assistance of many people. In particular, I'd like to thank the following people:

- **My family** The amount of time and effort than goes into writing a book is hard to measure. All I know is that I could not have produced this book without the support of Kristin (my wife) and Aidan (my son). There were many times when we wanted to spend time together but were unable to due to book obligations. Now that the book project is completed, I really look forward to adventures we will all share together.
- **My technical reviewers and editors** For this book revision, I truly had some fantastic people helping me. Christophe Nasarre has done just a phenomenal job of verifying my work and making sure that I'd said everything the best way it could possibly be said. He has truly had a significant impact on the quality of this book. Also, I'd like to extend a special thanks to Jamie Haddock. Jamie read the first edition of my book and e-mailed me numerous suggestions for ways to improve it. I saved all of these and then asked him to be part of the formal review process while writing the second edition of this book. Jamie's contribution is also quite significant. I'd also like to thank Stan Lippman and Clemens Szyperski for their review and the lively discussions we had. Finally, I'd like to thank Paul Mehner for his feedback.
- **Members of the Microsoft Press editorial team** The Microsoft Press people that I had the most contact with are Devon Musgrave and Joel Rosenthal. Both of them were an extreme pleasure to work with and made sure that things ran smoothly and did their best to make my words read good (except for this sentence ☺). Of course, I'd also thank Ben Ryan, my acquisitions editor, for ushering the contract through. Finally, I'd also like to thank other Microsoft Press people who had a hand in this project, including Kerri Devault, Elizabeth Hansford, Dan Latimer, Patricia Masserman, Bill Myers, Joel Panchot, Sandi Resnick, and William Teel.
- **Wintellectuals** Finally, I'd like to thank the members of my extended Wintellect family for being patient as I took time away from the business to work on this project. In particular, I'd like to thank Jim Bail, Jason Clark, Paula Daniels, Peter DeBetta, Sara Faatz, Todd Fine, Lewis Frazer, Dorothy McBay, Jeff Prosise, John Robbins, and Justin Smith.

Support

Every effort has been made to ensure the accuracy of this book. Microsoft Press provides corrections for books through the World Wide Web at the following address:

http://www.microsoft.com/mspress/support/

To connect directly to the Microsoft Press Knowledge Base and enter a query regarding a question or issue that you may have, go to:

http://www.microsoft.com/mspress/support/search.asp

If you have comments, questions, or ideas regarding this book, please send them to Microsoft Press using either of the following methods:

Postal Mail:

Microsoft Press
Attn: *CLR Via C#* Editor
One Microsoft Way
Redmond, WA 98052-6399

E-Mail:

mspinput@microsoft.com

Please note that product support is not offered through the above mail addresses. For support information regarding C#, Visual Studio, or the .NET Framework, visit the Microsoft Product Standard Support Web site at:

http://support.microsoft.com

Contents