修炼之道

.NET开发

要点精讲

周见智 博图轩 编著

基础 不忘记基础学习，深入剖析.NET基础概念。

内容 不主张高深莫测，使用丰富的说明性插图。

提升 不抛弃进阶提升，详细阐述通用框架原理。

思想 不局限技术平台，起于.NET但不止于.NET。

清华大学出版社

修炼之道：.NET 开发要点精讲

周见智　博图轩　编著

清华大学出版社
北 京

内 容 简 介

这是一本注重实际开发、接地气的.NET 技术书籍。作者结合多年的开发经验，用通俗易懂的语言，深入浅出地讲解在.NET 实际开发工作中的实用知识点。全书分为基础篇和设计篇两大部分。在基础篇，解释了“原子操作”、“阻塞方法与非阻塞方法”、“框架与库”、“调用与回调”等术语，重点阐述.NET 开发的三大基础知识点：数据类型、对象的生命期以及委托与事件。在设计篇，主要讲解“泵”结构在一些主流框架中的应用，以及它在 Socket 网络编程、Web 服务器开发等实际项目中起到的关键作用；并从软件设计模式、软件设计原则以及代码依赖 3 个方面，对软件架构进行了深入浅出的阐释。

本书适合已经入门且有一定编程经验并准备向高手迈进的.NET 开发者。本书同时也可作为大中专院校和.NET 技术培训机构的参考教材。

图书在版编目(CIP)数据

修炼之道：.NET 开发要点精讲/周见智，博图轩编著. —北京：清华大学出版社，2015
ISBN 978-7-302-39330-6

Ⅰ. ①修…　Ⅱ. ①周…　②博…　Ⅲ. ①计算机网络—程序设计　Ⅳ. ①TP393

中国版本图书馆 CIP 数据核字(2015)第 024953 号

责任编辑：杨作梅
装帧设计：杨玉兰
责任校对：马素伟
责任印制：王静怡

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 刷 者：三河市君旺印务有限公司
装 订 者：三河市新茂装订有限公司
经　　销：全国新华书店
开　　本：185mm×240mm　　**印　张**：20　　**插　页**：1　　**字　数**：446 千字
版　　次：2015 年 5 月第 1 版　　**印　次**：2015 年 5 月第1 次印刷
印　　数：1～3000
定　　价：49.00 元

产品编号：061797-01

这是一本给具有一定.NET基础的开发者写的书，作者结合自己的开发经验，深入浅出地讲解.NET知识。还将架构方面的知识贯穿于其中，是一本接地气、实用的.NET技术书籍，值得一读。

——张善友 腾讯高级工程师 9任微软MVP

该书对于想深入学习.NET的同学无疑是一扇“门”，普通的.NET学习书籍均是从基础往上学。本书最独到的地方就是作者运用了直白朴实的语言和丰富的插图，从“上帝视角”来分解、掌握.NET实际开发中的核心要领和设计模式。每次开卷阅读，都有所收获，在此代表读者感谢作者辛苦努力！本书值得推荐！

——郭恒海 高级信息系统项目管理师，.Net高级程序员，现任中软公司技术主管

整本书虽然选择的是.NET平台，但是书中阐述的编程思想完全适用于其他任何编程语言。尤其是书中第8章到第12章，作者着重强调了代码中“泵”结构对软件框架的重要性，以及设计模式与原则对软件系统的重要意义。可以说本书起于.NET，但不止于.NET。

——胡勇 RDIFramework.NET框架作者，多年项目开发与管理经验，高级工程师、信息系统项目管理师、数据库系统工程师

本书除了使用浅显易懂的文字来解释难以理解的问题之外，还包含了丰富的说明性插图。作者对每个问题进行阐述的同时，都尽可能地配上了流程图或其他具有说明性的插图。这更便于我们理解本书的内容。

——杨明明 高级程序员，微软MVP

前　言

“勤于总结”一直都是学习的最好方式之一，学习编程也一样，总结的过程会促使你持续不断地思考，从一个知识点向另一个知识点扩散，对于之前已经了解的知识，你会理解得更加深刻，而对于那些在总结过程中又触发到的新知识，也是一种额外的收获。笔者对此深有体会，一个不会总结的技术开发者是难以成为一名优秀的程序员的。笔者习惯将工作中的技术心得记录在自己的博客中，本书便是笔者从事.NET 开发多年来记录在博客中的一些技术总结，经过重新整理、发掘和完善，便形成了本书。希望这些技术总结能够给.NET 开发者带来一些帮助。

1．本书特色

.NET 技术有着广泛的应用范畴，从 Windows 桌面应用，到 ASP.NET Web 应用，到 WCF 分布式应用，到 Windows Mobile 嵌入式应用，到 Windows Phone 移动应用，到 ADO.NET 数据库应用，到 XML Web Service，.NET 无处不在。因此，现在市面上关于.NET 开发的书籍可谓之多，但笔者在本书中没有介绍一些平时几乎用不到的高深莫测的技术，如代码安全性或者 IL 语言这些底层的内容，也避免介绍一些简单的基础内容，例如类型的声明、语法、开发环境或者关键字的介绍等。本书总结了笔者认为在实际 .NET 开发工作中比较实用的知识点，如解释了诸如“原子操作”、“阻塞方法与非阻塞方法”、“框架与库”、“调用与回调”等这些常见但没有标准定义的编程术语；重点阐述了.NET 开发的三大基础知识点：数据类型、对象的生命期以及委托与事件；详细介绍了“泵”结构在一些主流框架诸如 Windows 桌面 GUI 框架中的应用，以及它在 Socket 网络编程、Web 服务器开发等实际项目开发中起到的关键作用；此外在软件架构方面，本书分别从软件设计模式、软件设计原则以及代码依赖 3 个方面也一一进行了说明。总之，这是一本注重实际开发、接地气的 .NET 技术书籍。

2．读者对象

本书适合已经入门，有一定编程经验并准备向高手迈进的 .NET 开发者，包括以下人员：

- .NET 工程师
- 在校计算机相关专业 .NET 方向大学生
- 从其他面向对象语言转向学习 .NET 编程的开发者

3．主要内容

全书共分 12 章。第 1～7 章为基础篇，分别涉及 .NET 平台介绍、编程术语解释、数据类型、对象生命期、委托与事件、异步编程以及 .NET 中的组件；第 8～12 章为设计篇，主要介绍代码中的“泵”结构、软件设计模式、软件设计原则以及代码依赖。其中第 8 章和第 9 章主要是为了引入代码中“泵”的概念，为第 10 章讲“泵”结构做铺垫。各章的主要内容简述如下。

第 1 章主要讲述 .NET 平台的组成，强调 .NET 平台出现的意义是改善了传统 Windows 的开发模式，而非专门提供一个跨平台的开发支持。

第 2 章是笔者从这几年的工作经历中总结出来的有关 .NET 编程的一些经验，对 .NET 实际开发工作有所帮助。

第 3 章介绍 .NET 编程的基础，即数据类型。笔者分别从值类型与引用类型的内存分配、赋值与复制以及对象判等方面进行了说明。

第 4 章讲述 .NET 对象的“从生到死”，以及怎样管理好对象的非托管资源，最后提出正确实现 Dispose 模式的方法，并说明为什么 Dispose 一个对象并不代表该对象死亡。

第 5 章讲述委托的结构以及它在 .NET 编程中的重要地位。程序的执行过程就是方法的调用过程，委托作为调用方法的一种手段，贯穿着整个 .NET 编程领域，之后说明 .NET 中事件与委托的关系，并且提到了事件编程的标准规范。

第 6 章讲述“异步编程模型”，阐述异步编程的必要性，介绍怎样使用委托去异步调用一个方法，以及异步编程过程中需要注意的一些事项，比如跨线程访问、线程安全等。

第 7 章介绍 .NET 编程中组件的定义，以及组件存在的意义。还介绍了“容器-组件-服务模型”，并提到一般集成开发环境(如 Visual Studio)中窗体设计器的原理，之后介绍组件的两种状态：运行时和设计时，以及区分组件当前状态的方法。

第 8 章主要重温了经典桌面 GUI 框架，并介绍 “Windows 消息”、“消息泵”以及“窗口过程”等概念，本章引入了代码中的“泵”的概念。

第 9 章介绍两种 Socket 网络通信模式，即 TCP 通信与 UDP 通信，并强调了“泵”结构在网络通信中的应用。

第 10 章介绍“泵”结构的几个常见应用场景，如 Socket 通信、桌面 GUI 框架以及 Web 服务器的实现等。

第 11 章介绍软件开发中的设计模式与设计原则。

第 12 章讲述代码依赖不可完全避免的原因、负面影响以及如何降低代码依赖程度。

4．关于注释与配图

本书配备了大量的图解和代码。为了让读者更好地理解示例代码和配图，笔者在每段代码和配图下面都加上了详细的中文说明文字。因此书中所示代码均为“说明性”代码，

并不确保能在编译器中编译通过，代码语言为 C#，默认 .NET 版本为 .NET Framework 3.5，但笔者认为本书所讲到的内容受 .NET 版本及其语言限制对代码影响不大。另外书中以“注”字开头的提示性文字更值得阅读，很多都是笔者认为值得注意的地方。

本书源代码下载地址为 http://www.cnblogs.com/cbook/p/3908393.html

5. 勘误与支持

本书由周见智、博图轩编著，参与本书编写的还有邓世健、赵晓芳、万建邦、刘耀宗。读者 QQ 群为 440170769，这里将随时发布与本书有关的信息。

由于笔者水平有限，书中难免会出现一些错漏，对此恳请各位读者多提宝贵意见，以便本书再版时能予以修正，意见可发送至邮箱 zhouzhi@syxysoft.com 与笔者探讨。同时可以访问作者博客：http://www.cnblogs.com/xiaozhi_5638，微博：http://weibo.com/netproduct。

编　者

目　录

第 1 章　另辟蹊径：解读. NET 1

1.1 前.NET 时代 2
1.2 .NET 的组成 3
1.2.1 .NET 中的语言 4
1.2.2 .NET 中的框架库 5
1.2.3 公共类型系统 5
1.2.4 公共语言规范 5
1.2.5 公共语言运行时 6
1.2.6 .NET 程序的运行流程 7
1.3 .NET 中的程序集 8
1.3.1 程序集与 EXE 文件的区别 8
1.3.2 程序集的组成 9
1.3.3 程序集的特点 10
1.4 .NET 的跨平台 10
1.4.1 Write Once, Run Anywhere 的真实现状 10
1.4.2 .NET 与 Java 平台出现的目的 11
1.4.3 重新看待.NET 12
1.5 .NET 平台出现的意义 12
1.6 本章回顾 14
1.7 本章思考 14

第 2 章　高屋建瓴：梳理编程约定 15

2.1 代码中的 Client 与 Server 16
2.2 方法与线程的关系 17
2.3 调用线程与当前线程 19
2.4 阻塞方法与非阻塞方法 20
2.5 UI 线程与线程 21
2.6 原子操作 22
2.7 线程安全 22
2.8 调用与回调 26
2.9 托管资源与非托管资源 27
2.10 框架与库 28
2.11 面向(或基于)对象与面向(或基于)组件 29
2.12 接口 30
2.13 协议 32
2.14 本章回顾 36
2.15 本章思考 36

第 3 章　编程之基础：数据类型 37

3.1 引用类型与值类型 38
3.1.1 内存分配 39
3.1.2 字节序 41
3.1.3 装箱与拆箱 42
3.2 对象相等判断 43
3.2.1 引用类型判等 43
3.2.2 简单值类型判等 44
3.2.3 复合值类型判等 45
3.3 赋值与复制 48
3.3.1 引用类型赋值 48
3.3.2 值类型赋值 49
3.3.3 传参 50
3.3.4 浅复制 53
3.3.5 深复制 55
3.4 对象的不可改变性 59
3.4.1 不可改变性定义 59
3.4.2 定义不可改变类型 60
3.5 本章回顾 62
3.6 本章思考 62

第 4 章　物以类聚：对象也有生命 65

4.1 堆和栈 66

4.2 堆中对象的出生与死亡.....69
4.2.1 引用和实例.....69
4.2.2 析构方法.....71
4.2.3 正确使用对象.....73
4.3 管理非托管资源.....75
4.3.1 释放非托管资源的最佳时间....75
4.3.2 继承与组合中的非托管资源....77
4.4 正确使用 IDisposable 接口.....82
4.4.1 Dispose 模式.....83
4.4.2 对象的 Dispose()和 Close() 87
4.5 本章回顾.....88
4.6 本章思考.....88

第 5 章 重中之重：委托与事件.....91

5.1 什么是.NET 中的委托.....92
5.1.1 委托的结构.....92
5.1.2 委托链表.....95
5.1.3 委托的“不可改变”特性.....101
5.1.4 委托的作用.....103
5.2 事件与委托的关系.....105
5.3 使用事件编程.....107
5.3.1 注销跟注册事件同样重要.....107
5.3.2 多线程中使用事件.....107
5.3.3 委托链表的分步调用.....109
5.3.4 正确定义一个使用了事件的类.....111
5.4 弱委托.....116
5.4.1 强引用与弱引用.....116
5.4.2 弱委托的定义.....119
5.4.3 弱委托的使用场合.....121
5.5 本章回顾.....121
5.6 本章思考.....122

第 6 章 线程的升级：异步编程模型 123

6.1 异步编程的必要性.....124
6.1.1 同步调用与异步调用 124
6.1.2 异步调用的优点 125
6.2 委托的异步调用 126
6.2.1 BeginInvoke 与 EndInvoke..... 126
6.2.2 AsyncCallback..... 128
6.2.3 处理异步调用时的异常 131
6.2.4 异步调用的应用 132
6.3 非委托的异步调用 134
6.3.1 异步方法 134
6.3.2 FileStream.BeginRead/FileStream.BeginWrite..... 136
6.4 异步编程时的注意事项 137
6.5 本章回顾 138
6.6 本章思考 138

第 7 章 可复用代码：组件的来龙去脉 141

7.1 .NET 中的组件 142
7.1.1 组件的定义 142
7.1.2 Windows Forms 中的组件..... 142
7.1.3 Windows Forms 中的控件..... 143
7.2 容器-组件-服务模型..... 144
7.2.1 容器的另类定义 144
7.2.2 容器与组件的合作 145
7.2.3 窗体设计器 150
7.3 设计时与运行时 154
7.3.1 组件的设计时与运行时 154
7.3.2 区分组件的当前状态 155
7.3.3 组件状态的应用 157
7.4 控件 157
7.4.1 控件基类 157
7.4.2 用户自定义控件 158
7.5 本章回顾 160
7.6 本章思考 160

第 8 章 经典重视：桌面 GUI 框架揭秘 163

8.1 了解传统 Win32 应用程序的必要性 164
8.2 Win32 应用程序的结构 165
8.2.1 运行平台决定程序结构 165
8.2.2 Windows 消息循环 167
8.2.3 窗口过程 169
8.2.4 创建基于 Win32 的单窗体应用程序 171
8.2.5 创建基于 Win32 的多窗体应用程序 178
8.3 .NET Winform 程序与传统 Win32 程序的关联 184
8.4 Windows Forms 框架 184
8.5 Winform 程序的结构 187
8.5.1 UI 线程 188
8.5.2 消息循环 189
8.5.3 创建窗体 195
8.5.4 窗口过程 197
8.6 模式窗体与非模式窗体 200
8.7 本章回顾 205
8.8 本章思考 205

第 9 章 沟通无碍：网络编程 207

9.1 两种 Socket 通信方式 208
9.1.1 IP 与端口 208
9.1.2 基于连接的 TCP 通信 209
9.1.3 基于无连接的 UDP 通信 211
9.1.4 应用层协议 212
9.1.5 .NET 中 Socket 编程的相关类型 215
9.2 TCP 通信的实现 217
9.2.1 定义 TCP 通信应用层协议 218
9.2.2 编写 TCP 通信服务端 219
9.2.3 编写 TCP 通信客户端 225
9.3 UDP 通信的实现 227
9.3.1 定义 UDP 通信应用层协议 228
9.3.2 编写 UDP 通信客户端 229
9.4 异步编程在网络编程中的应用 235
9.4.1 异步发送数据 237
9.4.2 异步实现“泵”结构 238
9.5 本章回顾 241
9.6 本章思考 241

第 10 章 动力之源：代码中的“泵” 243

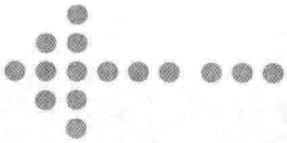
10.1 “泵”的概念 244
10.1.1 现实生活中的“泵” 244
10.1.2 代码中的“泵” 245
10.1.3 代码中“泵”的作用 247
10.2 常见的“泵”结构 249
10.2.1 桌面 GUI 框架中的“泵” 249
10.2.2 Socket 通信中的“泵” 250
10.2.3 Web 服务器中的“泵” 251
10.3 “泵”对框架的意义 258
10.3.1 重新回到框架定义 258
10.3.2 框架离不开“泵” 258
10.4 本章回顾 259
10.5 本章思考 259

第 11 章 规绳矩墨：模式与原则 261

11.1 软件的设计模式 262
11.1.1 观察者模式 262
11.1.2 Windows Forms 中的观察者模式 265
11.1.3 设计模式分类 267
11.2 软件的设计原则 267
11.2.1 SOLID 原则概述 267
11.2.2 单一职责原则 268
11.2.3 开闭原则 270

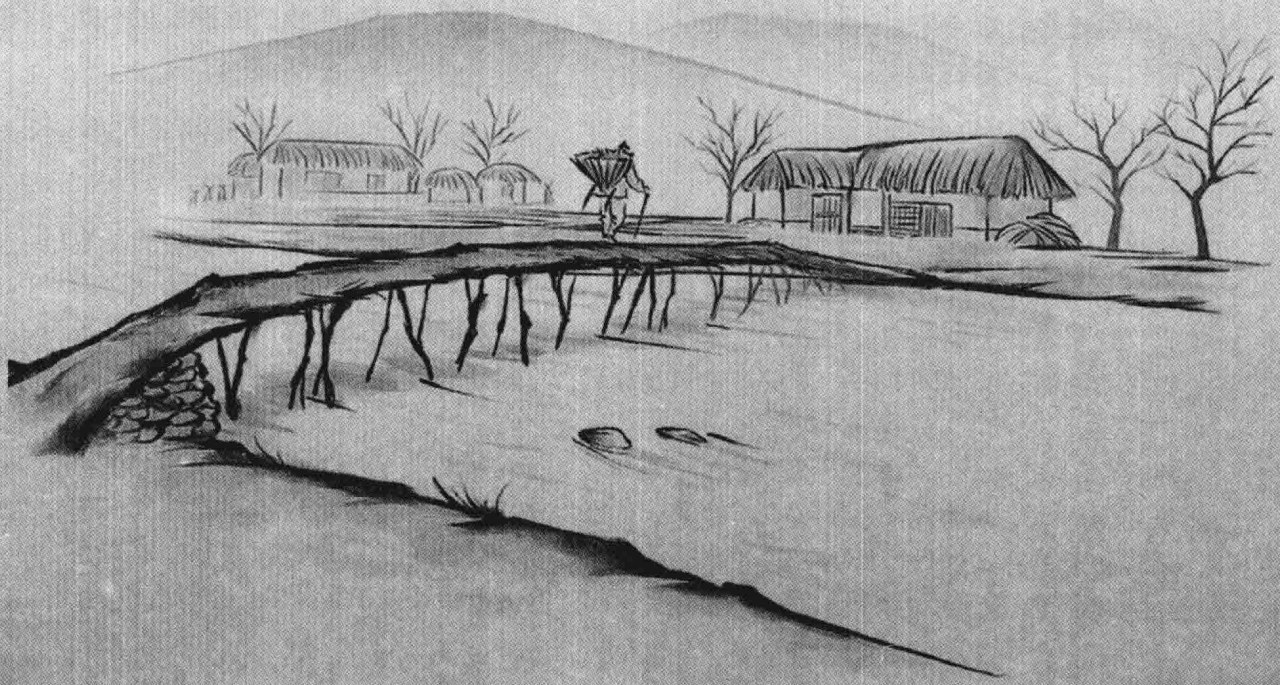
11.2.4 里氏替换原则.....272
11.2.5 接口隔离原则.....275
11.2.6 依赖倒置原则.....276
11.3 设计模式与设计原则对框架的意义.....278
11.4 本章回顾.....279
11.5 本章思考.....279
第 12 章 难免的尴尬：代码依赖.....281
12.1 从面向对象开始.....282
12.1.1 对象基础：封装.....282
12.1.2 对象扩展：继承.....286
12.1.3 对象行为：多态.....289
12.2 不可避免的代码依赖.....292
12.2.1 依赖存在的原因.....292
12.2.2 耦合与内聚.....294
12.2.3 依赖造成的“尴尬”.....295
12.3 降低代码依赖.....297
12.3.1 认识“抽象”与“具体”.....297
12.3.2 再看“依赖倒置原则”.....299
12.3.3 依赖注入.....302
12.4 框架的“代码依赖”.....305
12.4.1 控制转换.....305
12.4.2 依赖注入对框架的意义.....306
12.5 本章回顾.....306
12.6 本章思考.....306

第1章



另辟蹊径：解读.NET

本章主要说明.NET平台的组成及其出现的意义，讲述了程序集同时具备可读性(对于开发者)和可执行性(对于用户)、CLR的作用和它在.NET平台中所处的重要位置，以及.NET平台是怎样改变传统Windows的开发模式的。



1.1 前.NET 时代

在传统应用程序开发中，有各种各样的计算机开发语言，每种语言的使用方法并不完全一样，使用不同语言开发的人员之间很难形成交互点。于是不同的开发人员使用各种高级计算机语言编写出来的源代码，经过编译器编译之后，直接生成与平台(操作系统)关联的各种机器指令，如图 1-1 所示。

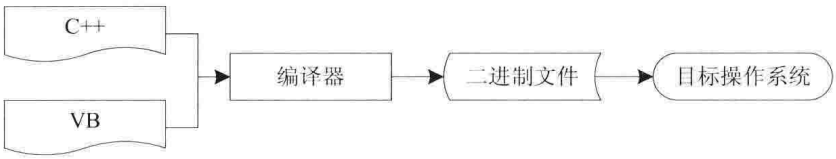


图 1-1 传统开发流程

由于不同的操作系统使用不同的指令集，因此使用编译器编译出来的二进制指令(文件)只能运行在某一特定的操作系统之中，如果想要让它运行在另外一种操作系统中，那么开发人员必须重新编译源代码，生成与另外一种操作系统相匹配的二进制指令。

出现这种情况的一个最主要的原因是在编译环节，因为编译器生成的二进制指令只能针对特定的操作系统。我们称这种编译方式为早期关联。也就是说，我们开发的程序在编译时就已经决定了它未来的运行环境。

随着计算机行业不停地发展，出现了各种各样的操作系统，以前一些不流行的操作系统也慢慢地开始流行了起来。这对开发人员来讲无疑是不利的，因为我们必须让我们现在开发出来的程序去适应各种各样的操作系统，而前面我们讲到，要想让我们的程序在不同的系统中运行，我们必须重新编译我们的源代码。

注：重新编译我们的源代码只是很笼统的一种说法，更多的时候是要修改源代码，原因将在后面的章节中提到。

为此，20 世纪 90 年代中期，Sun 公司推出了一个名叫 Java 的开发平台，主要目的是想让我们编写出来的程序能在任何一个操作系统之中运行，而不需要开发人员重新编译它的源代码。这个做法无疑是空前绝后的，开发人员不再需要为了让他们的程序去适应各种各样的操作系统而多做任何工作，“编译一次，到处运行”即是当时 Java 平台推广中最有名的一句广告语。

那么，到底 Sun 公司是怎样做到的“一次编译，到处运行”的呢？前面笔者提到过，之所以要为适应不同的操作系统而重新编译源代码，是因为编译器在编译阶段就把我们的源代码转换成了特定操作系统的二进制指令，在编译阶段，我们的程序就跟某一个操作系统已经关联上了，因此，编译出来的二进制指令只能被该操作系统识别。Sun 公司要做的，

就是在这个“编译阶段”让编译器编译出来的中间件不跟任何操作系统相关联，当这个中间件真正运行在某一个操作系统之中时，再由专门的程序将它翻译成与这个操作系统相匹配的指令。这样给人的感觉便是，我们只需要编译一次我们的程序，之后它就可以运行在任何一个系统之中了。

Java 平台程序从开发到运行的流程如图 1-2 所示。

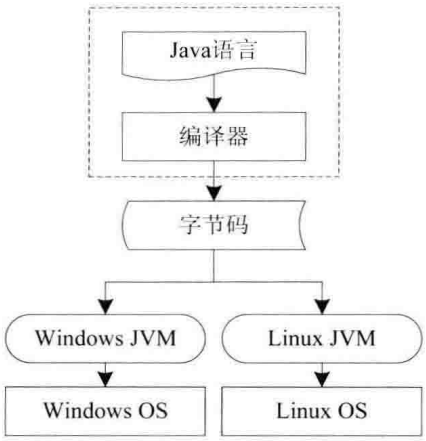


图 1-2　Java 平台程序开发运行流程图

图中虚线框内为开发人员负责的部分。理论上，不管最终程序运行在哪里，开发人员只要做一次同样的事情，剩下的环节对于开发人员来讲，都是不可见的。图中“字节码”就是前面提到过的中间件，它既与任何操作系统无关联，也不等同于传统开发过程中的“二进制文件”。图中 JVM 就是前面提到过的“专门的程序”，它能够将中间件翻译成与操作系统相匹配的指令。

Java 平台的出现在当时改变了整个软件行业的开发模式，尤其对开发人员来讲，是个极大的福利，开发人员从面向操作系统编程转变为面向平台编程。与此同时，微软紧跟其后，推出了与 Java 平台非常相似的一个开发平台，取名叫 Dot Net，简写为.NET。.NET 平台是微软的一个全新开发平台，虽然结构与 Java 平台相似，但是功能和出现的意义与 Java 千差万别，本章接下来几节会讲到这些。

注：上文提到的操作系统和平台的区别，为了避免混淆，本书中把操作系统称之为“操作系统”，而把 Java 和.NET 等称之为“平台”。另外，请注意 Java 语言和 Java 平台的区别。

1.2　.NET 的组成

何为平台？平台就是我们进行某项工作所需要的环境和条件。.NET 平台的出现彻底改

变了传统 Windows 的开发模式，它重新定义了 Windows 的开发流程，解决了之前程序开发中遇到的问题，比如 COM 时代的 DLL 地狱(DLL HELL)、C++编程中的内存泄露、VB 编程中使用 COM 的困难等。它还集成了各种各样的语言，无论哪种语言都可以开发.NET 程序，并且每种语言编写出来的组件(程序集)之间可以毫无障碍地通信，甚至还可以从 VB.NET 编写的程序集 DLL 中派生出一个新的 C#类型，而并不需要知道该类型基类的 VB 源代码，这在以前是完全不可能做到的。除此之外，.NET 还真正实现了二进制兼容性，比如修改一个 DLL 中的公共类型，只要使用了该 DLL 的客户端没有使用该公共类型，那么客户端就不需要重新编译，这在之前也是完全不敢想象的。

.NET 平台之所以具备以上这些功能，完全得益于它的组成结构。.NET 平台主要由丰富的框架库、各种各样的开发语言、各种语言同时遵守的规范、公共语言运行时等组成。正是这些组成部分相互协调工作，才使得.NET 平台具有如此强大功能，如图 1-3 所示。

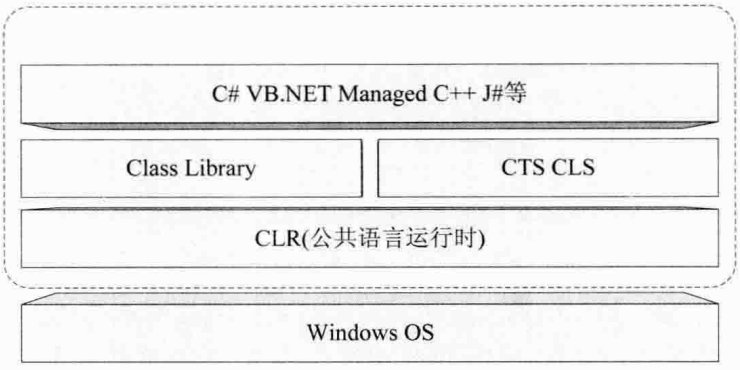


图 1-3　.NET 平台组成

图中列出的语言虽然仅仅为微软官方支持的几种语言，但理论上，任何一种语言只要遵守 CTS(公共类型系统)、CLS(公共语言规范)，并且有相应的编译器，那么它就可以成为.NET 平台支持的语言之一。

1.2.1　.NET 中的语言

.NET 平台官方给出的语言有 C#、VB.NET、Managed C++、J#和 F#，其中 J#主要是为了移植一些 J++项目，除了 C#和 VB.NET 之外，其他几种语言虽然平时也用得非常少，但是理论上，任何计算机语言只要能遵守公共类型系统和公共语言规范，并且有相应的.NET 编译器，那么这些语言就都可以成为.NET 平台语言大家庭中的一员。

.NET 中虽然语言种类繁多，但是各种语言之间却可以毫无障碍地进行交互，也就是说使用 C#编写的程序集，跟 VB.NET 编写的程序集是一样的，它们之间可以相互调用，这正是.NET 平台中的“语言集成”特性。

注：J++为微软早年以 Java 语言为基础开发出来的一种语言，在 JAVA 语言中增加了.NET 中特有的委托、事件等特性，后来由于版权问题，停止了支持该语言。

此外，上文中提及的程序集，是一种类似前面介绍 Java 平台时说到的“中间件”，将在下一节中讲到。

1.2.2　.NET 中的框架库

何为库？库就是别人已经写好了的，有组织有结构的代码集合，程序员可以直接拿过来使用，大到传统流行一时的 MFC、ATL 等框架，小到自己公司编写的一些通用工具类代码。.NET 平台本身是一个包含内容非常丰富的类库，范围涉及数据结构、网络、图形处理、加解密、线程、I/O、数据库等各个方面。作为一个开发人员，对“库”的概念应该不会陌生。

注：库和框架本身没有明显的区分定义，如果想了解更多，请参见本书第 2 章内容。

1.2.3　公共类型系统

前面讲到在.NET 平台中可以使用各种各样的语言进行程序开发，也就是说.NET 并不区分我们编写的某个类到底是使用 C#还是 Managed C++。使用 C#定义的一个接口和使用 VB.NET 定义的一个相同接口，最后产生的效果是一样的，都可以被 F#使用。那么这些语言定义的类型可以被.NET 平台相同对待的前提是什么呢？当然是必须遵守同一个类型规范，如果 C#中有 Interface 类型，而 VB.NET 中却没有 Interface 类型，那么就谈不上相同对待。

微软为.NET 平台定义了一套所有语言都必须遵守的规范——公共类型系统(Common Type System，CTS)，.NET 平台语言大家庭中所有语言都必须遵守这个规范，比如我们常见的值类型(ValueType)、引用类型(ReferenceType)、接口(Interface)、属性(Property)以及委托(Delegate)等。

1.2.4　公共语言规范

.NET 平台允许我们使用不同的语言开发应用程序，各种语言之间可以进行无障碍的交互。比如我们既可以在 C#中派生出一个使用 VB.NET 编写类型的子类，也可以在 VB.NET 中捕获 Managed C++中抛出的异常，也就是说继承、多态、异常处理等这些功能之间都可以交叉使用。然而，每一种语言本身的规则又是不相同的，比如在 Managed C++语言中对区分大小写很敏感，而在 VB.NET 语言中则对区分大小写不敏感。正因为不同的语言之间

可能支持不同的特性，而这些不同的特性又会影响语言之间的交互，所以微软为.NET 平台定义了一套所有语言必须遵守的规范——公共语言规范(Common Language Specification，CLS)，在.NET 平台语言大家庭中，所有语言都必须遵守这个规范。

最后需要指出的是，不仅开发人员需要了解以上规范，各语言编译器开发商同样也需要知道这些规范。

1.2.5 公共语言运行时

公共语言运行时(Common Language Runtime，CLR)是.NET 平台的核心。在使用各种语言开发的程序经过编译之后生成的中间件(程序集)，并不是传统意义上的二进制指令，而是一个类似代码数据的集合。这个代码数据集合并不能直接在操作系统中运行，如果说它还需要像在 Java 平台中一样，通过某种专门的程序将其转换成与操作系统相匹配的二进制指令，那么，CLR 就是.NET 平台中的这个“专门的程序”。由此来讲，.NET 平台中的程序需要经过两次“翻译”之后才能运行，一次由开发人员使用编译器进行编译，将源代码编译成中间件，另一次则由 CLR 将中间件翻译成与操作系统相匹配的二进制指令。

在传统程序开发过程中源代码的编译有且仅有一次，且直接生成二进制指令，一步到位。如果我们把传统编译称为“完全编译”，那么.NET 平台中的第一次编译就可以称之为“非完全编译”，如图 1-4 所示。

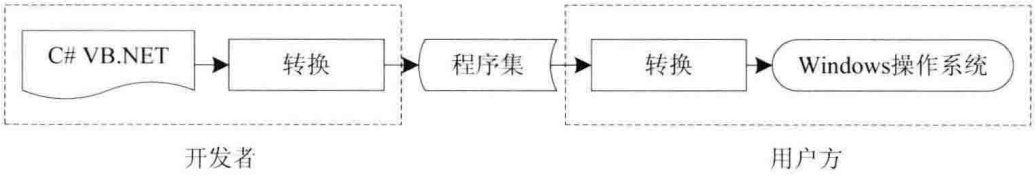


图 1-4 .NET 平台程序的两次转换

图 1-4 中第一次转换发生在开发者方，第二次转换发生在用户方。

注：上文说到的用户方是指程序的最终用户，CLR 一般安装在最终用户的操作系统中，它会自动运行，但 CLR 的工作过程对最终用户是不可见的。Windows Vista 以及之后的操作系统默认安装有不同版本的.NET CLR 程序。这便是为什么我们经常在 Windows XP 系统中运行一个.NET 程序时，系统会出现类似“没有相关运行环境”的错误提示。

在 CLR 中包含着一个名叫 JIT(Just-In-Time)的即时编译器，专门负责将中间件(程序集)转换成与操作系统相匹配的二进制指令，然后执行。CLR 的工作流程如图 1-5 所示。

JIT 编译器不会将所有的托管代码一次性编译成本地代码，而是当它需要执行某部分代