HZ BOOKS

微软公司核心技术书库

Microsoft Press

Inside Microsoft .NET IL Assembler

Microsoft .NET IL 汇编语言程序设计

（加） Serge Lidin 著

袁勤勇 何欣 卢冬梅 等译

Microsoft .net

机械工业出版社
China Machine Press

微软公司核心技术书库

Microsoft Press

Inside Microsoft .NET IL Assembler

Microsoft .NET IL 汇编语言程序设计

（加） Serge Lidin 著

袁勤勇 何欣 卢冬梅 等译

Microsoft .net

机械工业出版社
China Machine Press

许多有关 .NET 编程的书籍大都致力于介绍进行快速应用软件开发（RAD）的高级语言，而本书则深入剖析 .NET 公共语言运行环境的内部结构和操作，并且介绍了怎样驾驭详细描述这些结构和操作的 IL 汇编语言。实际上，.NET 公共语言运行环境所进行的任何工作，IL 汇编语言都能够进行解释。通过本书，可以学习到设计和实现 IL Assembler、IL Disassembler 及元数据验证工具的开发人员所需了解的有关 IL 汇编语言的各种内容。本书适用于编译器开发人员、多语言项目的开发人员及其他开发用于 .NET Framework 平台的更紧凑、快速代码的人员。想成为 .NET 高手，请读本书。

Serge Lidin: Inside Microsoft .NET IL Assembler (ISBN: 0-7356-1547-0).

本书版权登记号：图字：01-2002-6555

图书在版编目（CIP）数据

Microsoft .NET IL 汇编语言程序设计/（美）理丁（Lidin, S.）著；袁勤勇等译. -北京：机械工业出版社，2003.8

（微软公司核心技术书库）

书名原文：Inside Microsoft .NET IL Assembler

ISBN 7-111-12481-2

Ⅰ.M… Ⅱ.①理…②袁… Ⅲ.汇编语言-程序设计 Ⅳ.TP313

中国版本图书馆 CIP 数据核字（2003）第 050097 号

机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码 100037）

责任编辑：吴 怡 李云静

北京瑞德印刷有限公司印刷·新华书店北京发行所发行

2003 年 8 月第 1 版第 1 次印刷

787mm×1092mm 1/16·20.75 印张

印数：0 001—4 000 册

定价：45.00 元（附光盘）

前　言

为什么要写这本书

说实话，写这本书我责无旁贷。随着 Microsoft .NET 技术席卷全球，越来越多的信息技术人员开始涉足其中，涵盖这项技术各个方面的大量书籍也已经开始出现——恰如及时雨。然而，这些书籍都专注于将 .NET 作为一种高级编程语言和快速应用开发环境（RAD）进行讨论。无疑，这些内容非常重要，而且我相信这些书籍都要进行重印才能够满足大众的要求。但是，更深入的问题呢？

.NET 领域与其他信息技术领域类似，都像是一个庞大的倒立的“金字塔”，根植在它的核心技术之上。.NET 金字塔所仰赖的核心技术是公共语言运行环境（common language runtime）。这个运行环境会将中间语言（IL）二进制代码转换为特定平台的二进制机器代码，并且执行这些代码。位于运行环境之上的是 .NET Framework 类库、编译器以及类似于 Microsoft Visual Studio .NET 这样的环境。在它们之上的才是从辅助工具到面向终端用户的应用开发层。这个金字塔正在迅速地变高、变宽。

实际上，本书讲述的并不是公共语言运行环境——虽然运行环境是 .NET 金字塔的核心技术，但是它也是十分庞大的主题，以至于不可能在一本书的合理范围内对其进行详细描述。本书着重讨论另一个重要的主题：.NET IL Assembler（汇编语言）。IL 汇编语言（简写为 ILAsm）是一种低级语言，它专门用于描述公共语言运行环境的各种基本特性。如果运行环境具有这些特性，ILAsm 就必须能够对其进行解释。

ILAsm 与高级语言有所不同，而类似于其他汇编语言，它是平台驱动而非概念驱动的。汇编语言通常会与附属平台之间存在精确的语言映射，在目前的情况下，附属平台就是公共语言运行环境。事实上，这种映射非常精确，以至于这种语言可以用来描述运行环境的各个方面，这里的运行环境是指讲述 .NET 公共语言运行环境附属结构的 ECMA 标准化文档中所讨论的运行环境（ILAsm 本身也是公共语言附属结构的一部分，也是这个标准化文档的一个主题）。由于这种关系密切的映射，如果不涉及附属平台的大量细节，也就不可能描述相应的汇编语言。所以，从某种程度上讲，这本书归根到底也是讨论了公共语言运行环境。

IL 汇编语言在 .NET 开发者中非常流行。这并不是说所有的 .NET 开发者都喜欢使用 ILAsm 编程，而不喜欢使用 Microsoft Managed C++、Microsoft Visual C# .NET 或者 Microsoft Visual Basic .NET，而是说所有的 .NET 开发者目前都会使用 IL Disassembler（ILDASM），而且许多人经常使用它。无论 .NET 开发者所偏好的语言和从事的开发领域是什么，在他们的计算机屏幕上都会

参加本书翻译的人员主要有：袁勤勇、何欣、卢冬梅、刘海舟、李凯、万里骥、袁旭、康明珠、于卫、赵虎生、林闻涛等。

有一个青色的闪电——ILDASM 图标（对 David Drake 的默默赞扬）——在闪光。ILDASM 文本的输出是什么？对，正是 ILAsm 源代码。

目前可以看到许多基于 .NET 编程的书籍都专注于高级编程语言，如 Visual C# .NET 或者 Visual Basic .NET，或者是 ADO.NET 这样的技术，只是在有些时候会提到 ILDASM，作为反汇编的可选工具，来分析 .NET IL 可执行程序的内部情况。但是这些书籍都没有解释这些反汇编文本的含义，以及怎样对它们进行解释。这是可以理解的，因为对这些书籍来说，对元数据结构和 IL 汇编语言的详细描述都超出了其范围。

现在，读者可能已经明白写作这本书的意义了。因为我负责设计和开发 ILAsm 以及 ILDASM，所以我有责任详尽地展示它的魅力。

ILAsm 的历史

ILAsm 和 ILDASM 的第 1 版（分别名为 Asm 和 Dasm）由 Jonathan Forbes 在 1998 年初开发成功。当前的语言与它最初的形式已经有了很大的差异，它们所具有的惟一显著的公共特性就是伪指令关键字中的前导点号。这些汇编和反汇编工具最初是作为纯粹的内部工具而构建的，主要用来方便公共语言运行环境的持续开发；其在运行环境开发小组中得到了相当广泛的使用。

在 1999 年初，Jonathan 开始负责 Microsoft Messenger，而这个汇编和反汇编工具交给了 Larry Sullivan，Larry 所领导的开发团队有一个很有意思的名字 CROEDT（Common Runtime Odds and Ends Development Team）。在那一年的 4 月，我加入了这个团队，Larry 将这个汇编和反汇编工具的任务交给了我。当 1999 年 5 月公共语言运行环境的 α 版本出现在技术预览会（Technical Preview）上的时候，Asm，特别是 Dasm 引起了人们极大的关注，我被告之要重新处理这个工具，将其发展到产品的水平。在 Larry、Vance Morrison 和 Jim Miller 的帮助下，我完成了这项工作。因为这些工具当时还只是内部组件，所以我们（Larry、Vance、Jim 和我）从根本上对这个语言进行了重新设计——并不是只将其作为工具实现。

主要的突破发生在 1999 年的后半年，当时 ILAsm 输入和 ILDASM 输出已经可以实现足够的同步，进而可以获得有限的双向解析（round-tripping）。双向解析意味着用户可以编译特定语言，获得托管的（IL）可执行程序，对其进行反汇编、增加或者修改一些 ILAsm 代码，然后重新将其组装到经过修改的可执行程序中。双向解析技术开创了新纪元，此后不久，它就开始应用于 Microsoft 及其伙伴的特定产品的开发过程中。

大约与此同时，使用 ILAsm 作为基本语言的第三方 .NET 编译器也已经开始出现。人所共知的可能就是 Fujitsu 的 COBOL.NET，它在 2000 年 7 月的专业开发者会议（Professional Developers Conference）上大出风头，这次会议向开发者们发布了最初的公共语言运行环境 pre-beta 版本并附带 .NET Framework 类库、编译器和工具。

自从 2000 年底 beta1 版发布以来，ILAsm 和 ILDASM 的功能日渐完善，它们已经可以反映元数据和 IL 的所有特性，并支持完整的双向解析，而且能够将自身的改变与运行环境的改变保持同步。

谁该阅读本书

本书面向所有 .NET 的开发人员，因为他们在非常高级的层次上工作，所以可能需要考虑

程序的编译结果，或者希望分析所编译程序的最终结果。这些读者可以在本书中找到解释反汇编文本以及元数据结构概要的必要信息，这可以帮助他们开发出更有效率的应用。

因为分析反汇编信息和元数据结构对于评估任何面向 .NET 的编译器的正确性和效率都十分关键，所以这本书也非常适用于 .NET 编译器的开发者。还有一些目前数量较少但在逐渐增加的读者，包括直接使用 IL 汇编语言的开发者，例如，将 ILAsm 作为中间步骤的编译器开发人员、策划多语言项目的开发人员，以及希望提高驾驭公共语言运行环境能力（这种能力无法从高级语言中获得）的开发人员。

最后，本书对于从概念设计到实现和维护的软件开发的所有阶段都很有价值。

本书的组织

第一部分“快速入门”首先基于一个简单的示例程序对 ILAsm 和公共语言运行环境的特性进行简要概述。这个概述并不完善；然而，它可以用来描述运行环境和作为语言的 ILAsm 总体情况。

接下来的部分将会使用自底向上的方式详细讨论运行环境的特性以及相应的 ILAsm 结构。第二部分“底层结构”将会描述托管可执行文件的结构以及总体的元数据组织。第三部分“基本组件”专门介绍构成应用的必备基础组件：配件（assembly）、模块（module）、类（class）、方法（method）、字段（field）以及相关主题。第四部分将会带领用户深入讨论执行引擎，分析 IL 指令的执行以及托管异常处理。第五部分将会讨论其他组件的元数据表示和使用：事件（event）、特性（property）以及定制和安全属性。第六部分将会描述托管代码和非托管代码之间的互操作，并且讨论 ILAsm 和 ILDASM 在多语言项目中的实际应用。

本书的五个附录包含有关 ILAsm 语法、元数据组织以及 IL 指令集和工具特性的参考，这些工具包括 ILAsm、ILDASM 和离线元数据验证工具。

关于附带的光盘

本书包含有附带的光盘。如果用户启用了 Microsoft Windows 的 Autorun（自动运行）特性，那么当用户将光盘插入 CD-ROM 驱动器的时候，光盘自动运行界面就会启动；也可以手工运行光盘的根目录下的 StartCD.exe。StartCD 菜单将会提供与光盘上使用 eBook 格式存储的图书的链接；用于本书示例文件的安装程序；以及到 Microsoft 开发者网络（MSDN）的链接，可以从这里下载 Microsoft .NET Framework 软件开发工具箱（SDK），用户需要使用这个工具箱来编译和运行示例（要注意这个链接只能够由 MSDN 订阅者使用）。

安装示例文件

本书的示例文件位于 Code 文件夹中。用户可以从光盘中查看示例，或者也可以使用 StartCD 中的安装程序将其安装到硬盘上。安装示例文件大概需要 18 KB 的磁盘空间。如果在运行这些文件的时候遇到问题，可以参考附带光盘的根目录中的 Readme.txt 文件。

eBook

附带光盘包含有本书的电子版本。eBook（电子图书）可以让用户在屏幕上阅读本书，并

且搜索内容。有关安装和使用 eBook 的信息，可以参看 \ eBook 文件夹中的 Readme.txt 文件。

系统要求

为了运行示例，用户需要安装 .NET Framework 及其 SDK。最低要求，用户需要有公共语言运行环境、.NET Framework 类库以及 SDK，而不需要 Visual Studio .NET 和命令行编译器——当然，ILAsm 编译器除外。

致谢

首先，我要感谢与我共同编写本书的 Microsoft 出版社的编辑成员：Danielle Bird、Alice Turner、Robert Lyon、Mary Renaud、Julie Xiao（他在为我的表格和示例寻找错误的时候学习了 ILAsm，现在可能正在 Microsoft 申请开发人员的职位），特别是 Devon Musgrave。Devon，他是我的编辑，在处理本书的过程中无疑增添了许多白发，我是一个非常称职的专业程序员，但是我使用人类语言书写的内容往往仅限于写电子邮件。

我还要感谢我的同事，尽管他们十分繁忙，但是他们还是同意审阅本书的草稿，并且对内容提出了很好的建议，他们是：开发领导人员 Larry Sullivan（运行环境平台服务）、Bill Evans（运行环境元数据）、Chris Brumme（运行环境执行引擎）、Vance Morrison（运行环境 JIT 编译器）、程序管理人员 Erik Meijer（运行环境）和 Ronald Laeremans（Visual C++），以及我们最好的测试工程师 Kevin Ransom。我非常感谢他们的帮助，以及那些写满了“当你写作的时候，你应该考虑……”以及“不，一定还有其他的方式”的电子邮件。

当然，我还要感谢公共语言运行环境小组的所有成员，是他们为我提供了帮助，回答了我的问题、与我讨论规范文档以及深入分析源代码，他们是：Suzanne Cook、Shajan Dasan、Jim Hogg、Jim Miller、Craig Sinclair、Mei-Chin Tsai 等。

Microsoft 出版社支持信息

Microsoft 出版社已经尽了各种努力来确保本书以及它的附带光盘内容的正确性，Microsoft 出版社将会通过网站提供本书的更正信息，网址为：

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

如果用户对本书或者光盘有什么意见、问题或者想法，或者通过查询 Knowledge Base 而不能够回答的疑问，均可以通过电子邮件将它们发往 Microsoft 出版社：

mspinput@microsoft.com

或者可以将邮件寄往：

Microsoft Press

Attn：Inside Microsoft .NET IL Assembler Editor

One Microsoft Way

Redmond，WA 98052-6399

请注意，以上地址不提供对产品的支持。

目　录

第四部分　执行引擎剖析

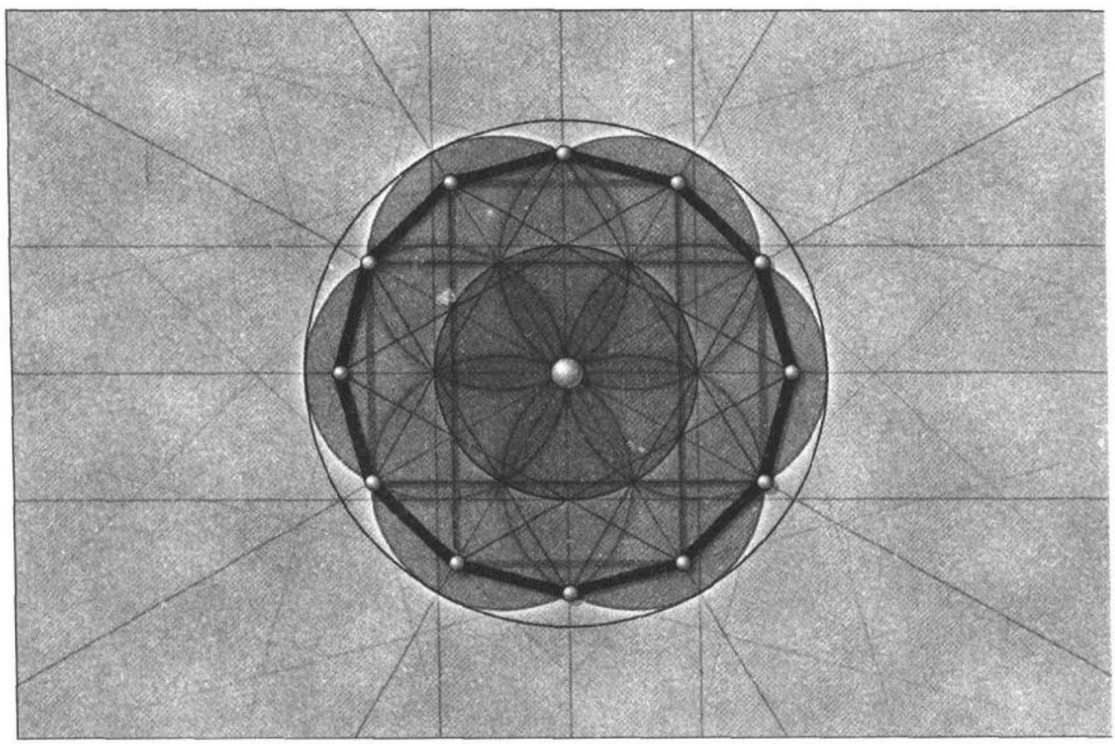
第五部分 特殊组件

第六部分 互操作性

第七部分 附 录

第一部分

快 速 入 门



第1章 简单示例

本章将会提供MSIL汇编语言（ILAsm）的总体概述（MSIL代表Microsoft中间语言）。首先分析一个使用ILAsm编写的相对简单的程序，然后提出一些修改意见，以展示怎样在这种语言中表述Microsoft .NET编程的概念和元素。

本章并不讲怎样使用ILAsm编写程序。但是它会帮助用户理解ILAsm编译器和IL反编译器（Disassembler，ILDASM）所做的工作，进而了解基于.NET的程序的内部结构。用户还会了解到一些发生在表象背后，出现在公共语言运行环境中神秘而有趣的事实——我希望它们具有足够的魅力，能够吸引读者阅读本书的其余部分。

注意 为了方便起见，本书将IL汇编语言缩写成ILAsm。不要将其与ILASM混淆，ILASM在.NET文档中是ILAsm编译器的缩写。

1.1 公共语言运行环境简介

尽管.NET公共语言运行环境只是.NET众多概念之一，但是它构成了.NET的核心（要注意，出于语言表达变化的考虑，有的时候我也会将公共语言运行环境称为运行环境）。这里不会着重介绍.NET平台的整体描述，而是会将重点放到.NET中实际进行操作的部分：公共语言运行环境。

更多信息 有关.NET及其组件整体结构的详尽讨论，可以参见David S. Platt编写的《Introducing Microsoft .NET》（Microsoft Press，2001）以及Tom Archer编写的《Inside C#》（Microsoft Press，2001）[⊖]。

简而言之，公共语言运行环境是.NET应用的运行环境。它在.NET应用和附属的操作系统之间提供了一个操作层。从原理上讲，公共语言运行环境类似于GBasic或者Smalltalk这样的解释语言的运行环境，或者类似于Java虚拟机。但是这种相似性只是在原理上，其实公共语言运行环境不是一个解释器。

由面向.NET的编译器（Microsoft Visual C# .NET、Microsoft Visual Basic .NET、ILAsm以及其他编译器）生成的.NET应用会表示为一种抽象的中间形式，这种形式独立于原始的编程语言，也独立于目标计算机和它的操作系统。正因为它们使用了这种抽象形式进行表示，所以使用不同语言编写的.NET应用都可以进行紧密的互操作，不仅可以彼此调用函数，而且还可以彼此进行类继承。

当然，由于编程语言之间存在差异，所以构建应用必须遵守一些规则才可以与其他应用进行协调。例如，如果用户使用Visual C# .NET编写了一个应用，并且命名了三个项MYITEM、

⊖ 以上两本书已由机械工业出版社引进出版。——编者注

MyItem 和 myitem，而 Visual Basic .NET 并不区分大小写，所以它就很难区分这些项。同样，如果用户使用 ILAsm 编写程序，并且定义了全局方法，那么 Visual C# .NET 就不能够调用这个方法，这是因为它没有全局（类以外）项的概念。

保证 .NET 应用互操作性的规则集合称为公共语言规范（CLS），在欧洲计算机制造商协会（ECMA）的公共语言基础结构（Common Language Infrastructure）标准提案的第一部分对其进行了介绍。它限制了命名规则、数据类型、函数类型以及其他的元素，为不同的语言提出了一个公共的规则。然而，需要记住的是，这个 CLS 只是一个推荐稿，还没有对公共语言运行环境的功能施加任何影响。如果用户的应用与 CLS 不兼容，它也有可能符合公共语言运行环境的标准，但是不能够担保它可以与其他应用在所有层次上进行互操作。

公共语言运行环境的 .NET 应用的抽象中间形式包括两个主要的组件：元数据（metadata）和托管代码（managed code）。元数据是表述应用中所有结构项——类、它的成员以及属性、全局项等等——的描述符和它们之间关系的系统。本章将会提供一些元数据示例，随后的章节将会讨论所有元数据结构。

托管代码指以 Microsoft 中间语言（MSIL）或者公共中间语言（CIL）的抽象二进制形式编码的应用方法（函数）的功能性。为了进行简化，我们简单地将这些编码看作是中间语言（IL）。当然，世界上还有其他中间语言，但是当我们进行讨论的时候，假定 IL 代表 CIL/MSIL，除非另行规定。

IL 代码由运行环境“托管”。公共语言运行环境管理包括了至少三个主要的活动：类型控制、结构异常处理、垃圾收集。类型控制涉及执行期间的项类型的验证和转换。结构化异常处理在功能上与“非托管”结构化异常处理（C++ 风格）相似，但是它是由运行环境执行，而不是由操作系统执行。垃圾收集是对不再使用的对象进行标识和处理。

在公共语言运行环境下设计的 .NET 应用会由若干个托管执行模块构成，每一个执行模块都包含有元数据和托管代码（可选）。托管代码为可选是因为可以建立不包含方法的托管执行模块（很明显，这样的执行模块只能够用作应用的辅助部分）。托管 .NET 应用被称为配件（这种表述进行了一定程度的简化，有关配件、应用域以及应用的更详尽的讨论，可以参见第 5 章）。托管可执行模块称为模块。用户可以建立单模块配件，也可以建立多模块配件。如图 1-1 所示，每一个配件都包含有一个主要模块，该主要模块在它的元数据中包含了配件的标识信息。

图 1-1 还表明了托管可执行模块的两个主要组件是元数据和 IL 代码，分别负责处理这两个组件的主要公共语言运行环境子系统是加载程序（loader）和 JIT（just-in-time，即时）编译器。

简而言之，加载程序会读取元数据，在内存中建立其内部表示，并且分配类及其成员。它会根据要求执行这个任务，也就是说只有当对类进行引用的时候，才会对其进行加载和分配。没有受到引用的类也不会被载入。当载入类的时候，加载程序会对相关元数据进行一系列一致性检查。

JIT 编译器会基于加载程序操作的结果，将使用 IL 编码的方法编译为附属平台的内部代码。因为运行环境不是解释器，所以它不能够执行 IL 代码。相反，要将 IL 代码在内存中编译为内部代码，然后执行内部代码。JIT 也会根据要求进行处理，这意味着只有调用方法的时候才会对其进行编译。已编译方法会在内存中缓存。然而，如果使用掌上 PDA 或者智能电话这

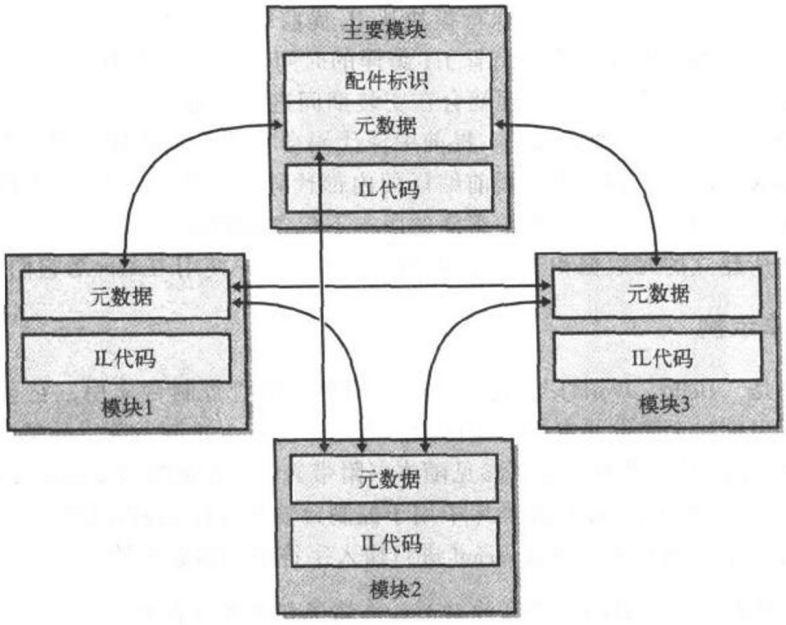


图 1-1　多模块 .NET 配件

样的小型计算设备，因为内存有限，就会在不再使用方法的时候将其抛弃。如果在方法被抛弃之后对其再次调用，就会对其进行重新编译。

图 1-2 中所示的图表展示了托管 .NET 的建立和执行序列。

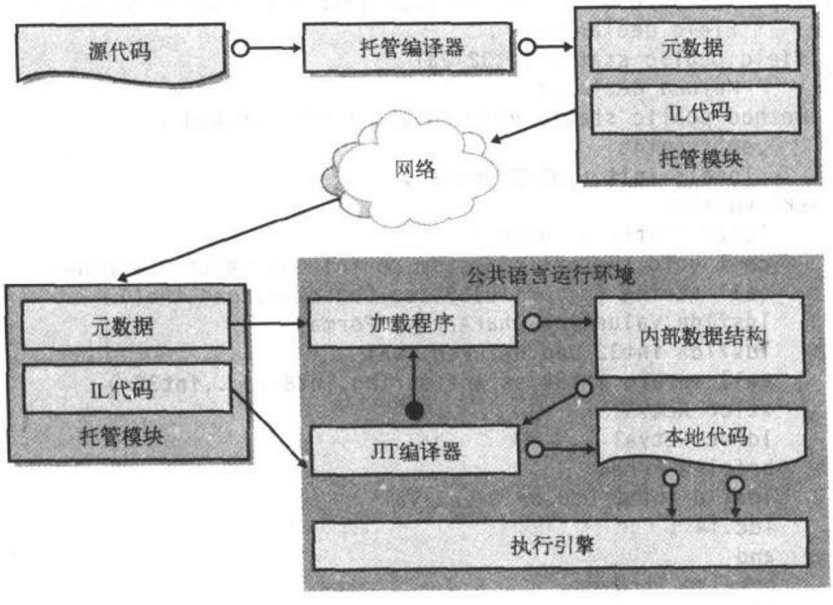


图 1-2　托管 .NET 应用的建立和执行

注：带有中空圆圈的箭头表示数据传输，带有实心圆圈的箭头表示请求和控制消息

可以使用 NGEN 实用工具将托管执行模块从 IL 提前编译成内部代码。如果用户希望从本地磁盘中重复运行可执行模块，为了节省 JIT 编译的时间，就可以使用这种方法。例如，对于 .NET Framework 的托管组件的标准过程就会在安装期间进行提前编译（Tom Archer 将这个过程称为安装时代码生成）。在这种情况下，提前编译代码会存储于本地磁盘或者其他介质上，每次调用可执行模块的时候，都会使用提前编译的内部代码版本替代初始的 IL 版本。然而，初始文件还是必须要提供，这是因为提前编译的版本不包含元数据。

让我们根据所建立的元数据和 IL 代码的角色来考虑怎样使用 ILAsm 对它们进行描述。

1.2　开始简单示例

这个示例不是“Hello, World!”，它只是一个简单的托管控制台应用，它会提示用户输入一个整数，然后识别这个整数是奇数还是偶数。当用户输入十进制数字以外的内容的时候，应用就会回答“How rude!”，并且终止（参见随本书附带光盘上的源文件 Simple.il）。

这个示例使用了 .NET Framework 类库中用于控制台输入和输出的托管控制台 API，并且它还使用了 C 运行环境中的非托管函数 sscanf 进行输入字符串到整数的转换。

注意　为了更容易阅读代码，所有的 ILAsm 关键字都以黑体表示。

```
//----------- Program header
.assembly extern mscorlib { }
.assembly OddOrEven  { }
.module OddOrEven.exe
//----------- Class declaration
.namespace Odd.or {
    .class public auto ansi Even extends [mscorlib]System.Object {
//----------- Field declaration
        .field public static int32 val
//----------- Method declaration
        .method public static void check( ) cil managed {
            .entrypoint
            .locals init (int32 Retval)
        AskForNumber:
            ldstr "Enter a number"
            call void [mscorlib]System.Console::WriteLine(string)
            call string [mscorlib]System.Console::ReadLine()
            ldsflda valuetype CharArray8 Format
            ldsflda int32 Odd.or.Even::val
            call vararg int32 sscanf(string,int8*,...,int32*)
            stloc Retval
            ldloc Retval
            brfalse Error
            ldsfld int32 Odd.or.Even::val
            ldc.i4 1
            and
            brfalse ItsEven
            ldstr "odd!"
            br PrintAndReturn
        ItsEven:
```