.NET 的本质可以在.NET 引入 Microsoft Windows 的一种新编程模型中得到体现。.NET 提供的这种编程模型适用于小屏幕的移动和嵌入式设备，也适用于个人计算机（无论是桌面系统，笔记本电脑还是平板计算机），还适用于服务器规模的系统。.NET 为多种软件应用程序的开发提供了统一的方法，包括传统 GUI 应用程序，通过 Web 浏览器访问的程序，以及后台 Web Services 程序。.NET 通过使用大量设备类统一了软件开发方法，并通过使用以下通用核心要素将大量的应用程序类型进行合并：

l 设计良好的编程接口

l 通用中间语言

l 通用语言运行时

l 通用语言规范

l 通用数据类型系统

l 通用语言架构

以下分小节依次介绍这些要素。

Well-Designed Programming Interface

设计良好的编程接口

在 20 世纪 80 年代早期，微软在开发 Windows 时的目标很简单，就是要做一个容易使用的 GUI 系统。当时的目标绝对不包括要使人们很方便的在这个操作系统中编写代码。相反，让那些职业程序员去管理复杂的计算机才是天经地义的。结果，Windows 就有了各种各样的 API，首先是 16 位的 Win16 API，然后是 32 位的 Win32 API。由于 Win32 也是 Windows CE 的 API，它也是我们介绍的重点（尽管经过了一次次的提炼与扩展，Win32 依然和 Win16 一样存在很多缺陷）。

www.docin.com

Win32 API 虽然在很多情况下可以使用而且必须使用，但它也有很多设计缺陷，这些缺陷在.NET 中得到了解决。

l 诡异的 API：Win32 有很多不一致，很多命名简单的函数，几个带有太多参数的函数，以及很多巨大而又不实用的数据结构。（很多情况下，一些多余的参数被保留着，但是却必须设置为 0；大型数据结构经常包含一些从未使用的成员。）

l 内存管理：需要清除系统对象的应用程序可能会导致 API 产生内存泄漏。

l 低层次的 API：Win32 API 有时被称为“Windows 的程序集语言”，因为它经常需要好几步才能完成看起来很简单的一项任务。

l 面向过程的 API：Win32 不提供任何面向对象语言提供的好处 29。

客观的说，开发 Windows 1 的开发团队成员相当不容易，他们在短时间内做了很多工作，克服了很多困难。他们在开发这种新 GUI 的同时，也开发了新编译器，新调试器，新的程序装载方法，以及新的内存管理程序。而且，他们的工作环境也很差，他们的计算机很慢（用的是 80286 和 80386 的 CPU），存储设备非常有限（那时最大也只有 20MB），还没有网络支持。考虑到他们所克服的困难以及他们最后所做出的成果，他们最终的产品简直可以说是奇迹。

Windows 一经发布，Windows API 就显得必不可少了。同时，程序员们也意识到需要一种更好的 API，并做过一些尝试。其中一次尝试就是 OS/2，这个项目的开发人员很多就是参与了以前 Windows 系统开发的。但是，我们前面也提到了，微软最终放弃了 OS/2 及其编程接口。其它几次尝试是要改进 Windows 编程的，包括 Visual Basic（1992 年首次发布）和 C++的 MFC 库。它们都部分改进了 Windows API，但却不能完全取代 Win32 API。

对于要开发 Microsoft Windows 软件的程序员来说，.NET 是替代 Win32 编程的一种新型编程接口，它设计良好，有着丰富的特性。这个接口在桌面系统中被称为.NET 框架，在移动和嵌入式设备中被称为.NET 精简框架。这些框架以完美的一致性方法向程序员提供操作系统的服务。

这些框架还是面向对象的。例如，.NET 中所有东西都是对象，从 Object 基类继承而来[35]。在这个面向对象的 API 中，所有支持的函数都包含在一些类之中。这些类又组成了命名空间，命名空间并不是什么新鲜事物，比如在 C++中就已经有了命名空间。但是在.NET 中，命名空间用于组织系统服务类，以便为操作系统函数提供可视化而且实用的组织形式。

命名空间和类的组织结构使得编程接口非常"清晰"，清晰一词通常用来形容一个设计良好的用户界面。如果你知道一个方法或者一个类，那么就很容易找到其它相关的方法或者类。例如，当你想要画出输出图形时，System.Drawing 命名空间就可以提供你所需要的一切。当你找到了 DrawString 方法来写出文本后，马上就能找到其它的文本输出要素，字体和画笔等，它们都组织到同一命名空间中。相反，Win32 程序员有时需要花费大量的时间与精力来组织需要的函数，即使是最简单的操作也不例外。

与命名空间组织类和代码一样，容器类也帮助程序员组织他们的数据。.NET 框架和.NET 精简框架都在很多设置中使用了容器类，这在写代码时增强了一致性和可用性。例如，一个窗口可以包含其它窗口。一个 ControlCollection 类型的容器类就体现了.NET 代码的包含关系，

www.docin.com

它可以使用名如 Add（添加），Remove（删除），Contains（包含），和 Clear（清除）的方法来访问集合。listbox 或者 combobox 中的元素也一样可以通过集合类进行访问，方法的名字也是一样的：Add，Remove，Contains 和 Clear。这种一致性使得.NET 成为了一个设计良好的，易于编程的 API。

Common Intermediate Language

通用中间语言

当你开发.NET 程序文件（.exe）或共享库文件（.dll）时，生成的可执行文件看起来和 Win32，MFC 或者编译的 Visual Basic 程序完全一样。这在技术上称为 Win32 可移植可执行（PE，portable executable）文件 31。每个 PE 文件都对应着一定的指令集。一些指令集与特定的 CPU 系列相关，例如 Intel x86，MIPS R4000 和 Toshiba SH4 指令集等。

.NET 可执行文件独特的一点是这些可执行文件所使用的指令集都与 CPU 没有关系。这些指令集称为中间语言（IL，Intermediate Language）指令集，也可以叫做微软中间语言（MS IL，Microsoft Intermediate Language）或者通用中间语言（CIL，Common Intermediate Language），IL 这个名字是受到 Ecma International32 标准约束而使用的。

程序员有时会搞不清楚 IL 究竟是解释性的还是编译性的。这关系到程序的性能，这是因为解释性语言（如 Smalltalk 和一些版本的 BASIC）比编译语言运行得更慢。解释性语言执行得慢是因为很多源代码级的指令处理要在运行时进行。但同时，解释性语言也更容易开发，其开发环境有良好的交互性，可以让程序员修改代码之后继续运行。因此，受到性能的影响，解释性语言一般只用于原型系统中，而很少用于开发产品代码。而 IL 则同时具备了解释性语言的可移植性和编译语言的速度优势。

www.docin.com

虽然使用 IL 解释器是可行的，但是所有已发布的.NET 运行时实现都将 IL 编译为本地指令再运行。从 IL 到本地 CPU 指令的转换是在运行时逐函数进行的。桌面版本的.NET 允许一个完整的 IL 可执行文件转换为本地指令并添加到本地映像缓存中（其中用到一种名为 ngen.exe 的工具）。但是.NET 精简框架并不具有这一特性，因为一般的移动设备没有那么多存储空间来存储本地映像缓存。

顺便提一下，IL 并不是世界上第一个可移植的指令集，在 IL 之前已经有了好几种。例如，Java 程序就经常编译为 Java 字节代码可执行文件，在运行时，Java 虚拟机（Java Virtual Machine）将这些可移植的指令集转换为本地机器指令。

可移植性指令到本地指令的转换称为 JIT33（Java 和.NET 程序都可以使用这个术语）。使用 IL，JIT 在桌面和.NET 精简框架环境中都可以逐方法的进行。这样，只有被调用的方法才会被转换，未被调用的方法则不会被编译执行，可以节省处理时间并缩小可执行文件的大小。

一旦一个方法由 IL 转换为了本地指令，这个方法的本地代码版本就滞留在内存中。被编译执行过的本地代码集合被引用作为本地映像缓存。这样，对于经常被调用的方法，JIT 所需要耗费的时间和空间平均下来就很少了。同时，本地方法在内存中也不会被锁定，他们所占用的内存还受到垃圾回收器的控制，可以像回收不可能用到的数据对象一样将它们回收再利用。

> Windows CE 3.0，第一个兼容 IL 的平台
> 大多数程序员都没有想到，微软第一个使用 IL 的产品是 Windows CE 3.0，使用的名字是通用可执行格式（CEF，Common Executable Format）。Pocket PC 支持从 IL 到本地 CPU 指令的转换。Windows CE 开发团队采用 IL 是因为它提供了一种可移植、与 CPU 无关的语言，使用它，一套独立的安装文件就可以支持各种不同的 CPU 了。
> 在 IL 和 CEF 之间还有一些重要的差异。其中之一就是 CEF 提供安装时的转换，而不是运行时的 JIT。而.NET 运行时，无论是桌面系统还是智能设备，都提供了运行时的 JIT 支持。

Common Language Runtime

通用语言运行时

通用语言运行时指的是支持.NET 程序执行的一套服务。它提供的服务包括将一个可执行文件装载到内存，由 IL 到本地指令集的 JIT，以及分配和释放对象等。

www.docin.com

通用语言运行时处理的任务中，介绍得最多的是内存管理，特别是对不可达对象的回收，叫做垃圾回收。.NET 框架并不是第一个实现这个服务的，Smalltalk，Visual Basic 和 Java 都或多或少的支持垃圾回收。

.NET 垃圾回收的重要之处在于它与多种编程语言兼容，在这方面它还是第一个。用一种语言写的程序可以调用另一种语言写的组件，它们之间可以随意的共享对象，这样就有可能共用同一个由通用语言进行时提供的内存分配器。这些内存可以适当的被垃圾回收器回收，并交由通用语言运行时重新分配使用。我们将在第 3 章中更详细的介绍垃圾回收。

Common Language Specification

通用语言规范

在.NET 术语中经常出现的"通用"一词有很多含义，包括"明白"和"普通"的意思。这其实并不算是.NET 中的"通用"，一个更恰当的词应该是"标准"，因为这些元素定义了一套标准，允许各种编程语言进行互操作。

特别是通用语言规范（CLS，Common Language Specification），它是指用来促进不同编程语言之间互操作性的一套规范。它是.NET 一项很重要的架构方面的特性。它使得已有的代码库更容易被作为托管代码库以供使用。它还允许程序员使用各种编程语言，而不仅仅局限于很少的几种。在.NET 之前，Win32 API 首先适用于 C 和 C++程序，Visual Basic 程序可以算作是二等公民，因为部分的 Win32 和 COM 不能从 Visual Basic 访问，从而导致使用时出现问题。随着微软不断发布新版本的 Visual Basic，C 程序员和 Visual Basic 程序员之间的差异越来越小，但是在非托管代码中依然存在隔阂。而在托管代码编程中，这样的差异基本已经消除了。

在.NET 中，Visual Basic .NET 也因为支持 CLS 而成为了头等公民。它和其它像 C#之类的语言完全平等。例如，.NET 精简框架的程序员在写程序和库代码时既可以使用 Visual Basic 语言，又可以使用 C#语言。这两种语言都使用同样的运行时（通用语言运行时），都支持一套通用的数据类型（我们马上要介绍的通用数据类型系统）。在 Visual Basic .NET 中编写的库可以从 C#程序中调用，反过来也没有问题。.NET 兼容语言所使用的通用标准使得这样的互操作性成为可能。

www.docin.com

> 与.NET 精简框架兼容的编译器
> 在本书准备印刷时，我们听说了其它几种与.NET 精简框架兼容的编译器。这些类型产品的出现使得.NET 更具活力，它们对所有的.NET 程序员都有帮助。
> 到目前为止，Borland 的 C# Builder 和 Delphi 8 都支持桌面版本.NET 框架的托管代码开发。Borland 还宣布更高版本的 C# Builder 和 Delphi 将支持.NET 精简框架。

在桌面版本的.NET 框架中，可用的编程语言种类更多。桌面版本和.NET 精简框架一样，支持 C#和 Visual Basic .NET。另外，.NET 兼容版本的 C++编译器还支持叫做托管 C++（Managed C++）的.NET 兼容模式。在.NET 最初的宣言中，有一个.NET 兼容的网站示例，它是用 COBOL 编写的。其它我们知道的.NET 语言还有 Curriculum，Dyalog APL，Eiffel，Forth，Fortran（Salford FTN95），Haskell，Java，J#，F#，JScript .NET，Mercury，Mondrian，Oberon，Oz，Pascal，Perl，RPG，Scheme，Smalltalk 以及 Standard ML 等。

Common Type System

通用数据类型系统

通用数据类型系统（CTS，common type system）是.NET 编程的一个基础构造块，它定义了所有.NET 兼容的编译器都支持的一套标准数据类型。使用不同语言开发的.NET 可执行文件之所以可以随意进行互操作，都有赖于 CTS 定义的通用数据类型集合的支持。我们将在第 3 章中更全面的介绍 CTS。

Common Language Infrastructure

通用语言架构

通用语言架构（CLI，Common Language Infrastructure）描述了.NET 框架的一个子集，微软将它提交作为 ECMA 的标准。它不包括 Windows 窗体类、Web 窗体类和 Web Services 类。它提供了足够的底层细节以支持第三方开发.NET 兼容的编译器、调试器和其它工具。CLI 还包括我们刚才介绍过的 CLS 和 CTS。

希望了解更多 CLI 知识的程序员可以找到一套 Word 文件，它们详细描述了 CLI 的各种要素。这些文件在 Visual Studio .NET 2003 中就有，可以在"..\SDK\v1.1\Tool Developers Guide\docs"的程序文件目录中找到。

Three .NET Application Classes

三种**.NET** 应用程序

当你刚刚接触到.NET 的时候，就可以发现有三种应用程序类型：Windows 窗体，Web 窗体和 Web Services。前两种类型在用户界面以及用户界面之间传递信息的机制方面有所差异。Windows 窗体作为独立的客户端应用程序运行，而 Web 窗体则是从 Web 浏览器启动的。另外，Web Services 是没有用户界面的 27。实际上它是一种无需终端的应用程序，为另外两种应用程序类型提供支持。

Windows Forms and Web Forms

Windows 窗体和 **Web** 窗体

Windows 窗体是传统的 GUI 应用程序。它既可能是一个像文字处理程序一样的独立应用程序，又可能作为联网的客户端/服务器结构应用程序中的客户端部分。Windows 窗体包括丰富的一直与 GUI 系统相关联的图形要素和交互要素。图 1.1 给出了一个桌面 Windows 窗体应用程序的例子，每个 Windows 用户都熟知的 Windows Explorer。

www.docin.com

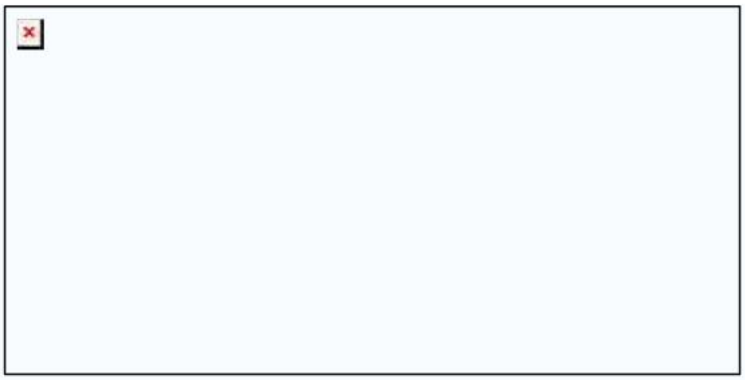

图 1.1 Windows Explorer，Windows 窗体应用程序的例子

Web 窗体是在 Web 浏览器中呈现的 HTML/脚本应用程序。根据定义，这是一种客户端/服务器结构的应用程序，需要到 Web 服务器端的实时连接来进行操作。绝大多数的计算工作都在服务器端进行处理，客户端则负责显示 HTML，在网页之间进行切换，并处理数据项表单。图 1.2 给出了一个在 Internet Explorer 中运行的 Web 窗体应用程序的例子，这是一个网站的登录页面，用来获取读者的反馈意见。

www.docin.com

图 1.2 一个简单的 Web 窗体应用程序

当我们刚看到.NET 的时候，对终端用户和程序员，无论是 Windows 窗体还是 Web 窗体看起来都没什么新鲜东西。图 1.1 的 Windows 窗体应用程序中看起来没有任何东西是.NET 所特有的。并且，Windows XP Explorer 实际并没有采用.NET 框架。所有这些一样的元素，组成了传统 GUI 应用程序，这正是人们经常用来指代 Windows 窗体应用程序的又一术语。对图 1.2 中的 Web 窗体应用程序来说也是一样的，对终端用户来说，它看上去和其它网页完全一样 28。实际上，图中所显示的应用程序是一个 ASP.NET 应用程序，使用 Visual Stu

dio .NET 生成的，并部署到一台安装了.NET 框架的 Web 服务器上。但是，尽管存在所有这些差异，对终端用户来说，看起来还是没有任何新的或者改进过的东西。

基于以上介绍，我们可能很容易得出这样的结论：.NET 只是一个新名词，没有任何新东西。但是，对软件开发人员来说，它和以前的东西却有着很大的差异。其中之一就是我们马上要介绍的第三种应用程序，Web Services。

Web Services

Web Services

第三种.NET 应用程序是 Web Services（以前称之为 XML Web Services）。Web Services 没有用户界面，但它通过提供网络中两台计算机通信的标准机制来支持其它两种.NET 应用程序类型。

这听起来似乎也没有什么新鲜的。毕竟，自从有了网络，就有了网络协议。你可以通过 FTP 协议传送文件，可以通过 TELNET 协议登录进命令提示行，可以通过 SMB 协议将文件提交到网络打印机或者文件服务器上，还可以通过 HTTP 协议浏览网页。

Web Services 建立在一系列标准协议基础上，特别值得一提的是简单对象访问协议（SOAP，Simple Object Access Protocol）。这个协议的目的是提供远程过程调用（RPC，Remote Procedure Call）。Web Services 是新一代基于网络客户端/服务器模式通信的技术。在此之前，采用的技术包括微软的分布式 COM（DCOM，Distributed COM），业内标准 DCE RPC 协议，还有 CORBA 等。

www.docin.com

现在很多大型计算机硬件和软件公司都采用 Web Services 作为一种标准。考虑到以前的 RPC 技术在兼容性问题上遇到的麻烦（比如 COM 与 CORBA 的兼容性问题），Web Services 主要采用与平台无关的 RPC 机制。

那么程序员为什么又要关心 Web Services 呢？很有可能一个程序员在他的编程生涯中都只开发传统的 GUI 应用程序或者 Web 应用程序，而不会从事一些分布式，面向网络的工作。在短期内，这种情况可能还会出现，但是 Web Services 正在变得更加重要。

一个原因是和快速发展的网络速度、可靠性以及安全性有关。计算机硬件惊人的发展使得 PC 革命成为可能，这些硬件的发展和计算机通信技术的进步也促进了 Internet 业务的增长。而随着数字通信技术的持续发展，更快的速度和更低的成本必将促使 Web Services 成为很多新型应用程序软件的重要组成部分。

Web Services 变得重要的另一个原因和万维网（World Wide Web）的发展与成功有关。万维网是计算机工业界迄今为止最成功的产品。如果你对此还有疑问，那么你可以去数数消费产品上的 URL，还有其它什么计算机产品能够达到这样的普及水平吗？

所有的大型公司，政府机构和非政府组织都在其网站方面做出了并将继续做出长期而巨大的投资。网站汇聚了大量信息，这些信息只有公布出来才能显现其价值。这些数据中一部分是用 HTML 呈现的，这是一种方便人们阅读的形式。

正因为 HTML 要方便人们阅读，它就很难用程序来访问。一些程序员开发了一些 HTML 脚本工具，这些工具可以将基于 Web 服务器的数据中所包含的 HTML 剥离出去，但这是一种很脆弱的访问数据的方法。其中一个原因是因为一旦这个网站发生变化，HTML 就将受到破坏。同时，法律方面的问题也需要考虑，因为网站的拥有者通常也拥有他们网站内容的版权。

Web Services 的一个用途就是要提供对海量网站数据的访问。回想你最近浏览过的两三个网站。你是不是正在在线阅读纽约时代周刊（New York Times）呢？或许你正在查询一场体育比赛的比分，或者是你要去访问的城市的天气情况。如果你住在如西雅图之类的大城市，你也许还能在线查询交通信息。这些情况下，你所搜索的数据可能都已经做到可以用程序访问了。既然使用搜索引擎来查找信息有时是必要的（当然，有时仅仅是为了娱乐），那么你所访问的任何信息都可以通过 Web Services 程序访问的一种选择。

但是并非所有的 Web 浏览器的 HTML 数据对 Web Services 都有效，到目前为止，只有几个大型网站的 Web 内容是可用的。以下是部分网站的列表，它们通过 Web Services 提供外部的接入服务。

www.docin.com

- Amazon：将第三方的产品清单数据库连接到 Amazon 的产品数据库中。
- eBay：程序化的访问拍卖信息。
- Google：提供网站搜索。
- MapPoint .NET：使用微软的这一服务可以自动访问地理数据。

docin 豆丁
www.docin.com