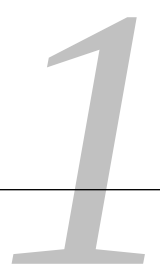
第 1 章 C#和.NET框架



本章内容

在.NET之前

.NET时代

编译成CIL

编译成本机代码并执行

CLR

CLI

缩写回顾

C#的演化

1.1 在.NET 之前

C#编程语言是为在微软公司的.NET框架[①]上开发程序而设计的。本章将简要介绍.NET从何而来，以及它的基本架构。在开始之前，我要指出C#的正确发音：see sharp[②]。

① 微软正式中文文献中一般称.NET Framework，本书考虑了国内读者习惯，统一译为.NET框架。——编者注

② 有一次我去应聘一个C#编程的职位，当时人力资源面试官问我从事“ see pound”（应为see sharp）的经验有多少！我过了一会儿才弄清楚他在说什么。

1.1.1　20 世纪 90 年代末的 Windows 编程

20世纪90年代末，使用微软平台的Windows编程分化成许多分支。大多数程序员使用Visual Basic(VB)、C或C++。一些 C和C++程序员在使用纯Win32 API ,但大多数人在使用MFC(Microsoft Foundation Class，微软基础类库)。其他人已经转向了COM (Component Object Model，组件对象模型)。

所有这些技术都有自己的问题。纯Win32 API不是面向对象的，而且使用它的工作量比使用MFC的更大。MFC是面向对象的，但是它却不一致，并逐渐变得陈旧。COM虽然概念简单，但它的实际代码复杂，并且需要很多丑陋的、不雅的底层基础代码。

所有这些编程技术还有一个缺点是它们主要针对桌面程序而不是Internet进行开发。那时，Web编程还是以后的事情，而且看起来和桌面编程非常不同。

1.1.2　下一代平台服务的目标

我们真正需要的是一个新的开始——一个集成的、面向对象的开发框架，它可以把一致和优雅带回编程。为满足这个需求，微软打算开发一个代码执行环境和一个可以实现这些目标的代码开发环境。这些目标列在图1-1中。

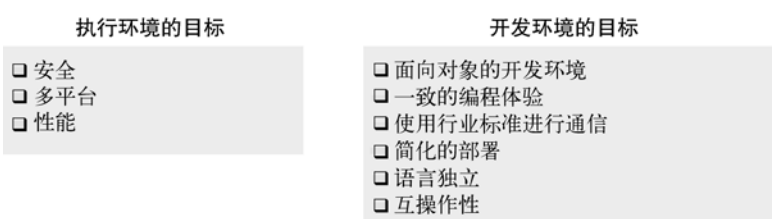


图1-1　下一代平台的目标

1.2 .NET 时代

2002年，微软发布了.NET框架的第一个版本，声称其解决了旧问题并实现了下一代系统的目标。.NET框架是一种比MFC和COM编程技术更一致并面向对象的环境。它的特点包括以下几点。

多平台 该系统可以在各种计算机上运行，从服务器、桌面机到PDA，还能在移动电话上运行。

行业标准 该系统使用行业标准的通信协议，比如XML、HTTP、SOAP、JSON和WSDL。

安全性 该系统能提供更加安全的执行环境，即使有来源可疑的代码存在。

1.2.1 .NET 框架的组成

.NET框架由三部分组成，如图1-2所示。[①] 执行环境称为CLR（Common Language Runtime，公共语言运行库）。CLR在运行时管理程序的执行，包括以下内容。

内存管理和垃圾收集。

代码安全验证。

代码执行、线程管理及异常处理。

① 严格地说，.NET框架由CLR和FCL（框架类库）两部分组成，不包括工具。FCL是BCL的超集，还包括Windows Forms、ASP.NET、LINQ以及更多命名空间。——编者注

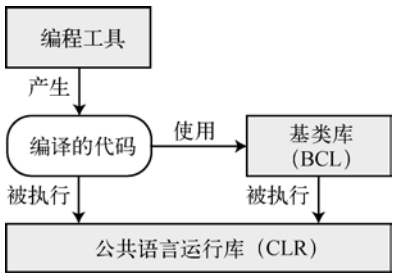


图1-2　.NET框架的组成

编程工具涵盖了编码和调试需要的一切，包括以下几点。

Visual Studio集成开发环境（IDE）。

.NET兼容的编译器（例如：C#、Visual Basic .NET、F#、IronRuby和托管的C++）。

调试器。

网站开发服务器端技术，比如ASP.NET或WCF。

BCL（Base Class Library，基类库）是.NET框架使用的一个大的类库，而且也可以在你的程序中使用。

1.2.2　大大改进的编程环境

较之以前的Windows编程环境，.NET框架为程序员带来了相当大的改进。下面几节将简要阐述它的特点及其带来的好处。

1. 面向对象的开发环境

CLR、BCL和C#完全是面向对象的，并形成了良好的集成环境。

系统为本地程序和分布式系统都提供了一致的、面向对象的编程模型。它还为桌面应用程序、移动应用程序和Web开发提供了软件开发接口，涉及的目标范围很广，从桌面服务器到手机。

2. 自动垃圾收集

CLR有一项服务称为GC（Garbage Collector，垃圾收集器），它能为你自动管理内存。

GC自动从内存中删除程序不再访问的对象。

GC使程序员不再操心许多以前必须执行的任务，比如释放内存和检查内存泄漏。这可是个很大的改进，因为检查内存泄漏可能非常困难而且耗时。

3. 互操作性

.NET框架专门考虑了不同的.NET语言、操作系统或Win32 DLL和COM之间的互操作性。

.NET语言的互操作性允许用不同的.NET语言编写的软件模块无缝地交互。

一种.NET语言写的程序可以使用甚至继承用另外一种.NET语言写的类，只需要遵循一定的规则即可。

正因为能够很容易地集成不同编程语言生成的模块，.NET框架有时被称为是语言无关的。

.NET提供一种称为平台调用（platform invoke, P/Invoke）的特性，允许.NET的代码调用并使用非.NET的代码。它可以使用标准Win32 DLL导出的纯C函数的代码，比如Windows API。

.NET框架还允许与COM进行互操作。.NET框架软件组件能调用COM组件，而且COM组件也能调用.NET组件，就像它们是COM组件一样。

4. 不需要COM

.NET框架使程序员摆脱了COM的束缚。作为一个C#程序员，你肯定很高兴不需要使用COM

编程环境，因而也不需要下面这些内容。

IUnknown接口 在COM中，所有对象必须实现IUnknown接口。相反，所有.NET对象都继承一个名为object的类。接口编程仍是.NET中的一个重要部分，但不再是中心主题了。

类型库 在COM中，类型信息作为.tlb文件保存在类型库中，它和可执行代码是分开的。在.NET中，程序的类型信息和代码一起被保存在程序文件中。

手动引用计数 在COM中，程序员必须记录一个对象的引用数目以确保它不会在错误的时间被删除。在.NET中，GC记录引用情况并只在合适的时候删除对象。

HRESULT COM使用HRESULT数据类型返回运行时错误代码。.NET不使用HRESULT。相反，所有意外的运行时错误都产生异常。

注册表 COM应用必须在系统注册表中注册。注册表保存了与操作系统的配置和应用程序有关的信息。.NET应用不需要使用注册表，这简化了程序的安装和卸载。(但有功能类似的工具，称为全局程序集缓存，即GAC，我会在第21章阐述。)

尽管现在不太需要编写COM代码了，但是系统中还是在使用很多COM组件，C#程序员有的时候需要编写代码来和那些组件交互。C# 4.0引入了几个新的特性，来简化这个工作，我们将在第25章讨论。

5. 简化的部署

为.NET框架编写的程序进行部署比以前容易很多，原因如下。

.NET程序不需要使用注册表注册，这意味着在最简单的情形下，一个程序只需要被复制到目标机器上便可以运行。

.NET提供一种称为并行执行的特性，允许一个DLL的不同版本在同一台机器上存在。这意味着每个可执行程序都可以访问程序生成时使用的那个版本的DLL。

6. 类型安全性

CLR检查并确保参数及其他数据对象的类型安全，不同编程语言编写的组件之间也没有问题。

7. 基类库

.NET框架提供了一个庞大的基础类库，很自然地，它被称为基类库（Base Class Library, BCL）。（有时称为框架类库——Framework Class Library，FCL。）[①]在写自己的程序时，可以使用其中的类，如下所示。

通用基础类 这些类提供了一组极为强大的工具，可以应用到许多编程任务中，比如文件操作、字符串操作、安全和加密。

集合类 这些类实现了列表、字典、散列表以及位数组。

线程和同步类 这些类用于创建多线程程序。

XML类 这些类用于创建、读取以及操作XML文档。

1.3 编译成 CIL

.NET语言的编译器接受源代码文件，并生成名为程序集的输出文件。图1-3阐述了这个过程。

程序集要么是可执行的，要么是DLL。

① 严格地说，BCL并不等同于FCL，而只是FCL的一个子集，包括System、System.IO、System.Resources、System.Text等FCL中比较底层和通用的功能。——编者注

程序集里的代码并不是本机代码，而是一种名称为CIL（Common Intermediate Language，公共中间语言）的中间语言。

程序集包含的信息中，包括下列项目：

程序的CIL；

程序中使用的类型的元数据；

对其他程序集引用的元数据。

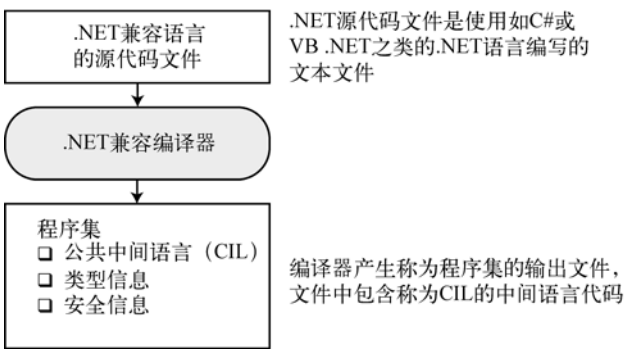


图1-3　编译过程

说明　随着时间的推移，公共中间语言的缩写已经改变，而且不同的参考书可能使用不同的术语。大家经常遇到的与CIL有关的另外两个术语是IL（Intermediate Language）和MSIL（Microsoft Intermediate Language），它们在.NET发展初期和早期文档中频繁使用，不过现在已经用得很少了。

1.4 编译成本机代码并执行

程序的CIL直到它被调用运行时才会被编译成本机代码。在运行时，CLR执行下面的步骤（如图1-4所示）：

检查程序集的安全特性；

在内存中分配空间；

把程序集中的可执行代码发送给实时（Just-in-Time, JIT）编译器，把其中的一部分编译成本机代码。

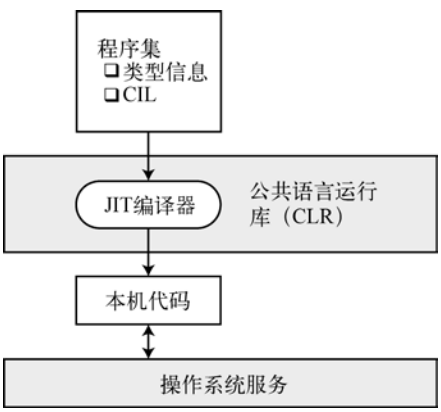


图1-4 运行时被编译成本机代码

程序集中的可执行代码只在需要的时候由JIT编译器编译，然后它就被缓存起来以备在后来的程序中执行。使用这个方法意味着不被调用的代码不会被编译成本机代码，而且被调用到的代码只被编译一次。

一旦CIL被编译成本机代码，CLR就在它运行时管理它，执行像释放无主内存、检查数组边界、检查参数类型和管理异常之类的任务。有两个重要的术语由此而生。

托管代码 为.NET框架编写的代码称为托管代码（managed code），需要CLR。

非托管代码 不在CLR控制之下运行的代码，比如Win32 C/C++ DLL，称为非托管代码（unmanaged code）。

微软公司还提供了一个称为本机映像生成器的工具Ngen ,可以把一个程序集转换成当前处理器的本机代码。经过Ngen处理过的代码免除了运行时的JIT编译过程。

编译和执行

无论原始源文件的语言是什么，都遵循同样的编译和执行过程。图1-5说明了3个用不同语言编写的程序的完整编译时和运行时过程。

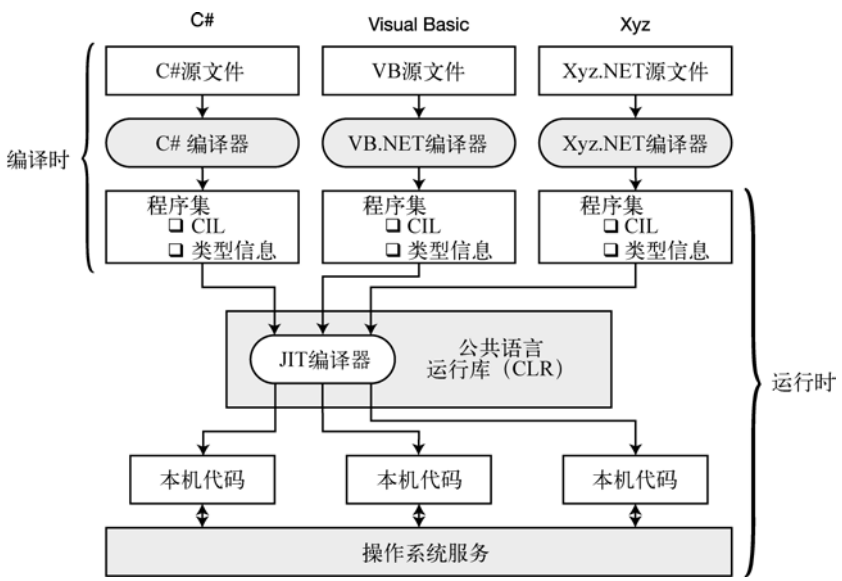


图1-5　编译时和运行时过程概览

1.5　CLR

.NET框架的核心组件是CLR，它在操作系统的顶层，负责管理程序的执行，如图1-6所示。

CLR还提供下列服务：

自动垃圾收集；

安全和认证；

通过访问BCL得到广泛的编程功能，包括如Web服务和数据服务之类的功能。

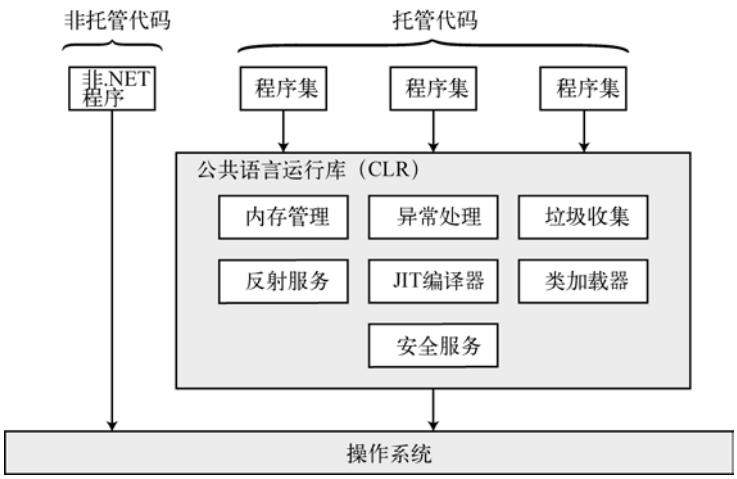


图1-6 CLR概览

1.6　CLI

每种编程语言都有一组内置的类型，用来表示如整数、浮点数和字符等之类的对象。过去，这些类型的特征因编程语言和平台的不同而不同。例如，组成整数的位数对于不同的语言和平台就有很大差别。

然而，这种统一性的缺乏使我们难以让使用不同语言编写的程序及库一起良好协作。为了有序协作，必须有一组标准。

CLI(Common Language Infrastructure ,公共语言基础结构)就是这样一组标准 ,它把所有.NET 框架的组件连结成一个内聚的、一致的系统。它展示了系统的概念和架构，并详细说明了所有软件都必须坚持的规则和约定。CLI的组成如图1-7所示。

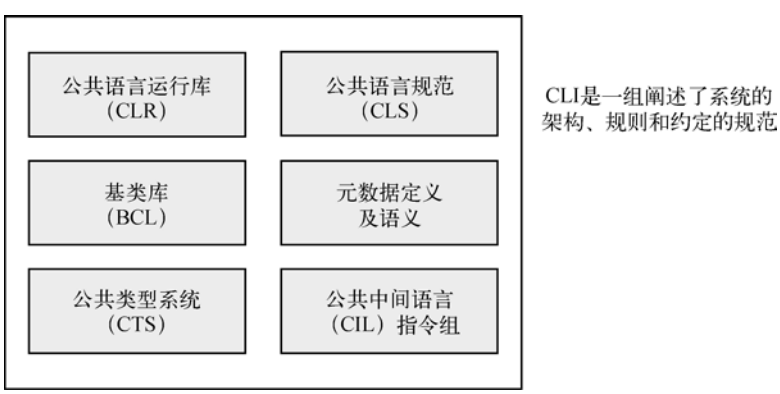


图1-7　CLI的组成

CLI和C#都已经被Ecma International批准为开放的国际标准规范。[Ecma本来是Europen Computer Manufacturer Association（欧洲计算机制造商协会）的缩写，但现在已经不是缩写了，它就是一个词。] Ecma的成员包括微软、IBM、惠普、Adobe等众多和计算机及消费性电子产品

有关的公司。

CLI 的重要组成部分

虽然大多数程序员不需要了解CLI规范的细节，但至少应该熟悉公共类型系统和公共语言规范的含义和用途。

1. 公共类型系统

CTS（Common Type System，公共类型系统）定义了那些在托管代码中一定会使用的类型的特征。CTS的一些重要方面如下。

CTS定义了一组丰富的内置类型，以及每种类型固有的、独有的特性。

.NET兼容编程语言提供的类型通常映射到CTS中已定义的内置类型集的某一个特殊子集。

CTS最重要的特征之一是所有类型都继承自公共的基类——`object`。

使用CTS可以确保系统类型和用户定义类型能够被任何.NET兼容的语言所使用。

2. 公共语言规范

CLS（Common Language Specification，公共语言规范）详细说明了一个.NET兼容编程语言的规则、属性和行为，其主题包括数据类型、类结构和参数传递。

1.7 各种缩写

本章包含了许多.NET缩写，图1-8将帮助你直观地理解它们。

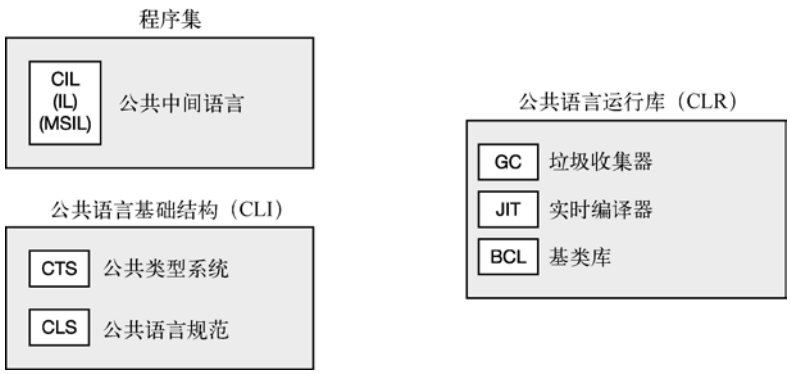


图1-8　.NET缩写

1.8　C#的演化

C#的最新版本是5.0。每个新版本在新添加的特性中都有一个焦点特性。5.0的焦点特性是异步编程，将在第20章展开介绍。

图1-9阐明了C#每个版本的焦点特性以及本书哪些章节会讨论它们。

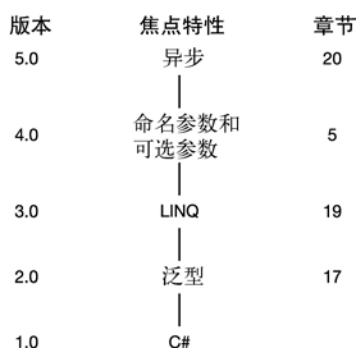


图1-9　C#各版本的焦点特性