第 5 章　J2EE 与.NET 平台

在软件开发领域中，目前主流的平台主要有.NET 和 J2EE，本章就简单地介绍这两个开发平台。

5.1　J2EE 平台简介

J2EE（Java 2 Platform Enterprise Edition）为设计、开发、装配和部署企业级应用程序提供了一个基于组件的解决方案。使用 J2EE 可以有效地减少费用，快速设计和开发企业级的应用程序。J2EE 平台提供了一个多层结构的分布式的应用程序模型，该模型具有重用组件的能力、基于扩展标记语言（XML）的数据交换、统一的安全模式和灵活的事务控制。使用 J2EE 不仅可以更快地发布新的解决方案，而且独立于平台的特性让使用 J2EE 的解决方案不受任何提供商的产品和应用程序编程界面（API）的限制。用户可以选择最适合自己的业务系统所需技术的产品和组件。

5.1.1　分布式的多层应用程序

J2EE 平台采用了多层分布式应用程序模型。实现不同逻辑功能的应用程序被封装到不同的组件中，处于不同层次的组件被分别部署到不同的机器中。图 5-1 表示了两个多层的 J2EE 应用程序根据下面的描述被分为不同的层。其中涉及的 J2EE 应用程序的各个部分将在 J2EE 组件中给出详细描述。

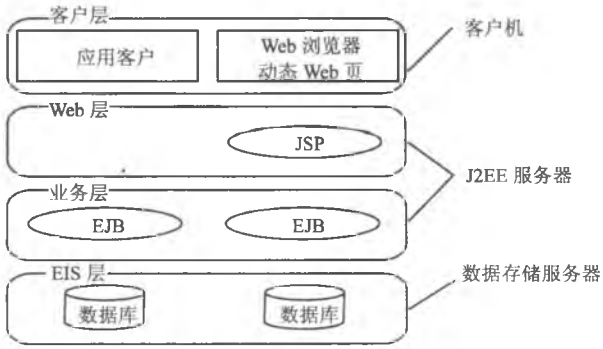

图 5-1　多层结构的应用程序

（1）运行在客户端机器的客户层组件。

（2）运行在 J2EE 服务器中的 Web 层组件。

（3）运行在 J2EE 服务器中的业务层组件。

（4）运行在 EIS 服务器中的企业信息系统（EIS）层软件。

从图 5-1 中可以看到 J2EE 应用程序既可以是三层结构，也可以是四层结构。一般来说，J2EE 应用程序经常分布于三个不同的位置，我们通常将 J2EE 应用程序的多层结构考虑为三层结构。这三个位置分别是：客户端机器、J2EE 服务器和在后端数据存储服务器。三层结构的应用程序可以理解为在标准的两层结构中的客户端程序和后端服务中间增加了应用服务器。

5.1.2　J2EE 组件

J2EE 应用程序由一系列的组件组合而成。一个 J2EE 组件就是一个软件单元，它随同它相关的类和文件被装配到 J2EE 应用中，并与其他组件通信。J2EE 组件由 Java 编程语言写成，并和用该语言写成的其他程序一样进行编译。J2EE 组件和“标准的”Java 类的不同点在于：它被装配在一个 J2EE 应用中，具有固定的格式并遵守 J2EE 规范，由 J2EE 服务器对其进行管理。J2EE 规范是这样定义 J2EE 组件的：客户端应用程序和 applet 是运行在客户端的组件；Java Servlet 和 Java Server Pages（JSP）是运行在服务器端的 Web 组件；Enterprise Java Bean（EJB）组件是运行在服务器端的业务组件。

1. J2EE 客户端

图 5-2 显示了客户层组成的多种方式。客户端可以直接和运行在 J2EE 服务器中的业务层通信，也可以通过运行在 Web 层中的 JSP 页面和 Servlet 同业务层组件进行通信。J2EE 客户端可以分为 Web 客户端、Applets 和 Java 应用程序。

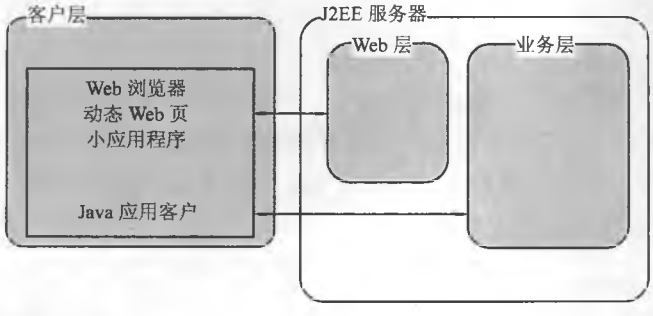

图 5-2　服务器通信

（1）Web 客户端。一个 Web 客户端也被称为瘦客户端，也就是显示由 JSP 或 Servlet 动态产生的 Web 页面的程序。瘦客户端一般不做像数据库查询、执行复杂的业务逻辑及连接传统应用程序这样的操作。当使用一个瘦客户端时，重量级的操作都被交

给在 J2EE 服务器执行的 EJB。这样可以充分发挥 J2EE 服务器端技术在安全性、速度、耐用性和可靠性方面的优势。

（2）Applets。Applets 也可以用于连接 J2EE 应用。一个 Applet 是一个用 Java 编程语言编写的小的客户端应用程序，它使用安装在 Web 浏览器的 Java 虚拟机运行。然而，为了在 Web 浏览器中成功地运行 Applet，客户端系统很可能需要 Java 插件和安全策略文件。

（3）应用程序客户端。一个 J2EE 应用程序客户端运行在客户端机器上，它使用户可以处理需要比标记语言所能提供的更丰富的用户界面的任务。具有代表性的是用 Swing 或抽象窗口工具包（AWT）API 建立的图形用户界面（GUI），当然也可能是一个命令行的界面。

应用程序客户端可以直接访问运行在业务层的 EJB。当然一个 J2EE 应用程序客户端也可以打开一个 HTTP 连接来与一个运行在 Web 层的 Servlet 建立通信。

2．J2EE 中间层

J2EE 中间层的内容极为丰富，也是 J2EE 平台中最重要的内容，EJB（Enterprise Java Beans）是 J2EE 规范中重要的组件。图 5-3 显示了一个 EJB 如何从客户端接受数据，对它进行处理，并将其发送到企业信息系统层以做存储。一个 EJB 也可以从存储器获取数据，对它进行处理（如果需要），并将其发送到客户端应用程序。

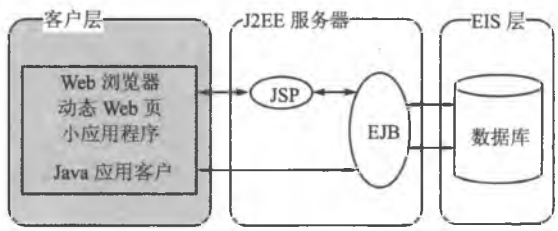

图 5-3　业务层和 EIS 层

SUN 在 EJB2.0 规范中对 EJB 定义如下：EJB 是用于开发和部署多层结构的、分布式的、面向对象的 Java 应用系统的跨平台的构建体系结构。使用 EJB 编写的应用程序具有可扩展性、交互性，以及多用户安全的特性。这些应用只需要写一次，就可以发布到任何支持 EJB 规范的服务器平台上。

3．企业信息系统层

企业信息系统层处理企业信息系统软件并包含诸如企业资源计划（ERP）、主机事务处理、数据库系统和其他传统系统这样的底层系统。J2EE 应用程序组件可能需要访问企业信息系统。J2EE1.3 支持 Connector 构架，该构架是将 J2EE 平台连接到企业信息系统上的一个标准 API。

5.1.3　J2EE 容器

如果从零开始，多层应用程序是很难编写的，开发者需要花费大量的精力来完成事务处理、状态管理、多线程、资源池和其他底层处理。基于组件并与平台无关的 J2EE 体系结构使 J2EE 应用程序易于编写，除了因为业务逻辑被封装到可重用的组件中外，J2EE 服务器以容器的形式为每一个组件类型提供底层服务。因此我们不需要自己开发这些服务，而是全力以赴地着手解决业务问题。

在容器中可包含若干组件，并为这些组件提供服务。Web 组件、EJB 等都必须首先被装配到一个 J2EE 应用程序中，并且部署到相应的容器，才可以执行。部署时会将 J2EE 应用程序组件安装到 J2EE 容器中，如图 5-4 所示。

（1）J2EE 服务器：J2EE 服务器是 J2EE 产品的运行容器。一个 J2EE 服务器提供 EJB 容器和 Web 容器。

（2）EJB 容器：EJB 容器管理它所包含的 EJB，容器负责对象的注册、提供远程接口、创建和清除对象实例、检查对象安全性、管理对象的活动并协调分布式事务处理。

（3）Web 容器：Web 容器管理 JSP 页面和 Servlet 组件的执行。Web 组件和 Web 容器运行在 J2EE 服务器中。

（4）客户端应用程序容器：管理应用程序客户端组件的运行。应用程序客户端和它的容器运行在客户端中。

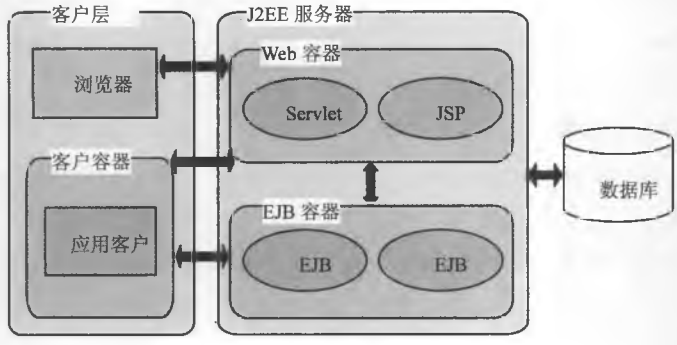

图 5-4　J2EE 服务器和容器

5.1.4　J2EE 的部署

J2EE 组件被分别打包并绑定到一个 J2EE 应用中以供部署。每一个组件、组件相关资源，例如，GIF、HTML 文件和一个部署说明组成一个模块并被添加到 J2EE 应用程序中。一个 J2EE 应用由一个或几个 EJB 组件、Web 组件或应用程序客户端组成。根据不同的设计需求，最终的企业解决方案可以是一个 J2EE 应用程序，也可以由多个 J2EE 应用程序组成。

一个 J2EE 应用程序，以及它的每一个模块都有它自己的部署说明。一个部署说明就是一个 XML 文件，它描述了一个组件的部署设置。例如，它可以描述一个 EJB 事务属性和安全性授权。部署说明信息是公开的，改变部署说明不必修改源代码。在运行时，J2EE 服务器将按照部署说明中的描述执行 J2EE 应用。

一个 J2EE 应用及它的所有模块被提交到一个 Enterprise Archive（EAR）文件中。一个 EAR 文件就是一个具有.ear 扩展名的标准的 Java Archive(JAR)文件。在 J2EE SDK 中有程序部署工具的 GUI 版本。通过这个部署工具可以建立 EAR 文件，并在其中添加 JAR 文件和 Web Archive（WAR）文件。

（1）每一个 EJB JAR 文件包含一份部署说明、一组 EJB，以及相关的文件。

（2）每一个应用程序客户端的 JAR 文件包含一份部署说明、应用程序客户端的类文件，以及相关的文件。

（3）每一个 WAR 文件包含一份部署说明、Web 组件，以及相关的资源。

使用模块和 EAR 文件可以很方便地使用同一组件装配出不同的 J2EE 应用。不需要额外的编程工作，唯一要做的是在 J2EE EAR 文件中添加各种 J2EE 模块。

5.2　.NET 平台简介

微软在 2000 年 7 月发布了新的应用平台.NET，整个.NET 平台包括 4 部分产品。

（1）.NET 开发工具。.NET 开发工具由.NET 语言（C#、VB.NET）、一个集成的 IDE（Visual Studio.NET）、类库和通用语言运行时（CLR）构成。

（2）.NET 专用服务器。.NET 专用服务器由一些.NET 企业服务器组成，如 SQL Server 2000、Exchange 2000、BizTalk 2000 等。这些企业服务器可以为数据存储、E-mail、B2B 电子商务等专用服务提供支持。

（3）.NET Web 服务。虽然 Web Service 不是.NET 所特有（关于 Web Service 内容请见第 6 章），但.NET 为 Web Service 提供了强有力的支持。开发者使用.NET 平台可以很容易的开发 Web Service。

（4）.NET 设备。作为同 J2ME（Java 2 Micro Edition）竞争的部分，.NET 还为手持设备，如手机等，提供了支持。

完整的.NET 平台涵盖了 JVM、J2SE 和 J2EE 全部的内容。.NET 平台出现的时间较短，大多数读者对.NET 底层的了解少于 Java 虚拟机。故 5.2 节主要介绍.NET 的底层机制，基本上同 JVM 相对应。在 5.3 节的对比分析中会引入.NET 平台在开发企业应用中的框架和作用。

5.2.1　.NET 平台概述

Microsoft .NET 平台包括五个部分，如图 5-5 所示。

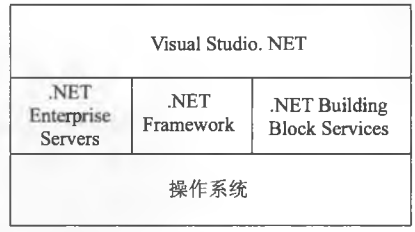

图 5-5　Microsoft .NET 平台

- 操作系统是.NET 平台的基础，在操作系统方面，Microsoft 有着强大的开发能力，目前的.NET 平台可以运行在包括 Windows 2000 Server 在内的多个 Microsoft 提供的操作系统中。
- .NET Enterprise Servers 提供了一系列的.NET 服务器产品，包括：Application Center 2000、BizTalk Server 2000、Commerce Server 2000 等一系列产品。通过这些产品可以缩短构建大型企业应用系统的周期。
- .NET Building Block Services 指的是一些成型的服务，如由 Microsoft 提供的 NET Passport 服务。.NET 的开发者可以以付费的方式直接将这些服务集成在自己的应用程序中。
- .NET Framework 位于整个.NET 平台的中央，它不但是技术界讨论的热点，也是本章比较的重点。.NET Framework 为开发.NET 应用提供了低层的支持，如 CLR 等。事实上，即使没有位于顶层的 Visual Studio.NET，只要有了.NET Framework，开发者一样可以开发.NET 应用程序。
- Visual Studio.NET 是.NET 应用程序的集成开发环境，它位于.NET 平台的顶端。Visual Studio.NET 是一个强大的开发工具集合，里面集成了一系列.NET 开发工具，如 C#.NET、VB.NET、XML Schema Editor 等。

5.2.2　.NET Framework

.NET Framework 中引入一系列的新技术和新概念，图 5-6 给出了.NET Framework 的结构图。其中核心的部分就是通用语言运行时——CLR（Common Language Runtime）。CLR 是.NET 程序的执行引擎，.NET 的众多优点也是由 CLR 所赋予的。CLR 同 JVM 的功能类似，提供了单一的运行环境。任何.NET 应用程序都会被最终编译为中间语言 IL（Intermediate Language），并在这个统一的环境中运行。也就是说 CLR 可以用于任何针对它的编程语言，这也就是.NET 的多语言支持。CLR 还负责.NET 应用程序的内

存管理、对象生命期的管理、线程管理、安全等一系列的服务。我们将在 5.2.3 节详细介绍 CLR。

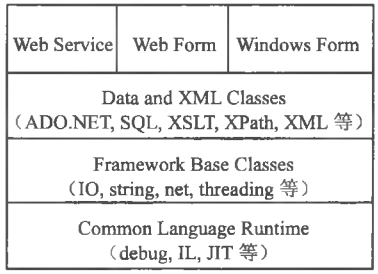

图 5-6　.NET Framework 组成结构

除了 CLR 外，.NET 提供了.NET 类库，每一种.NET 语言都可以使用该类库。基本类库中包含了大量的类供开发者使用，如图 5-6 所示。除此之外，使用某种.NET 语言开发的类可以被其他的.NET 语言直接使用，从而充分利用各种语言的优点。这也就是说我们可以使用 VB.NET 书写 UI（User Interface）相关的内容，而底层的计算功能是用 C++开发。.NET Framework 还对命名空间提供了支持，熟悉 Java 的程序员一定非常欣赏 Java 中清晰的类的层次结构，.NET Framework 中的命名空间与之类似，非常适合组织大规模的类的层次结果。如 System.Data，或者由开发者自定的 Abc. Accounting. Service。

5.2.3　通用语言运行时 CLR

1．托管

托管是.NET 的一个专门概念，它是融于通用语言运行时(CLR)中的一种新的编程理念，使用托管意味着代码可以被 CLR 所管理，使用 CLR 提供的各种服务。无论是用什么语言，只要采用了.NET 的托管机制，就能开发出具有最新特性如垃圾自动收集、程序间相互访问等的.NET 框架应用程序。

所有的 C#、VB.NET、JScript.NET 默认时都是托管的，但 Visual C++默认时不是托管的，也就是非托管 C++，必须在编译器中使用选项才能产生托管代码。由托管概念所引发的托管应用程序包括托管代码、托管数据和托管类三个组成部分。

（1）托管代码：.Net 环境提供了许多核心的运行（Runtime）服务，比如异常处理和安全策略。为了能使用这些服务，必须给运行环境提供一些信息代码（元数据），这种代码就是托管代码。

（2）托管数据：与托管代码密切相关的是托管数据。托管数据是由公共语言运行的垃圾回收器进行分配和释放的数据。在默认情况下，C#、Visual Basic 和 JScript.NET

数据是托管数据。不过，通过使用特殊的关键字，C# 数据可以被标记为非托管数据，Visual C++数据在默认情况下是非托管数据。

（3）托管类：尽管 Visual C++数据在默认情况下是非托管数据，但是在使用 C++的托管扩展时，可以使用“_gc”关键字将类标记为托管类。就像该名称所显示的那样，它表示类实例的内存由垃圾回收器管理。另外，一个托管类也完全可以成为 .NET 框架的成员，由此可以带来的好处是，它可以与其他语言编写的类正确地进行相互操作，如托管的 C++类可以从 Visual Basic 类继承等。但同时也有一些限制，如托管类只能从一个基类继承等。需要说明的是，在托管 C++应用程序中既可使用托管类也可以使用非托管类。这里的非托管类不是指标准 C++类，而是使用托管 C++语言中的_nogc 关键字的类。

2. 程序集

程序集是一个抽象的概念，也比较难于理解。我们可以这样理解程序集的概念：首先，程序集是一个或多个托管模块或资源文件的逻辑分组；其次，程序集是可重新使用、确保安全和版本控制的最小单元。图 5-7 是把托管模块组合称为程序集的示意图。

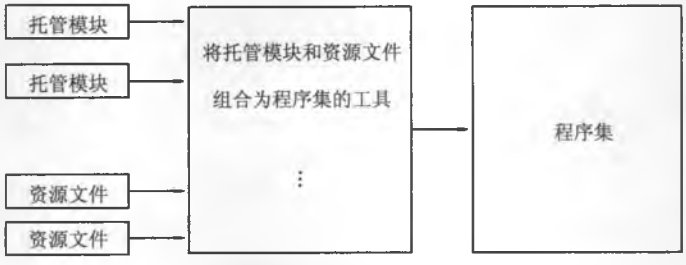

图 5-7　将托管模块组合为程序集

程序集允许我们将可重用、可部署的组件的逻辑部分和物理部分相分离。例如，可以将不常用的类型或资源放到独立的程序集文件中，这些独立的文件可以根据需要从 Web 站点上动态下载。如果不需要使用这些程序集，那么就永远不需要下载这些文件。

程序集中还包含被引用的程序集的自描述信息——如版本号。由于 CLR 可以通过这些自描述信息执行程序而不需要其他的附加信息。因此程序集使得.NET 应用更容易部署。

3. 中间语言

当托管代码被编译后，并不产生本机的二进制代码，而是产生包含中间语言 IL（Intermediate Language）的程序集。每种托管语言（C#，VB.NET 等）都产生中间语言程序集，IL 为被编译的代码提供一种通用的表示。Microsoft 宣称，编译为 IL 的代码可以运行在任何使用.NET Framework 的处理器和操作系统中（其实也只用 Windows 和

X86－笔者注），不过包含IL的程序集可以非常容易地运行于Windows 98或者Windows ME，从而为.NET提供更多的应用平台。

被编译为IL语言的程序集不能够直接运行，CLR中使用JIT（just-in-time）编译器将IL转化为本机的CPU指令。图5-8显示了首次调用一个方法的执行过程。

JIT(just-in-time) Compiler负责将IL代码编译成本机的CPU指令，因为JIT是在程序第一次运行时实时的编译，所以JIT也称作实时编译器。只有当程序第一次运行时，JIT才会工作，它会把编译好的本机指令放在内存中，也就是说，程序在第一次运行的时候速度相对较慢，而以后的运行速度会加快很多。但是，如果程序终止，随着程序退出内存空间，编译好的本机指令也会随之退出内存，那么重新启动应用程序会再一次启动编译过程。如果程序的运行平台已经确定同时对性能有较高的要求，.NET Framework提供了一个小工具——NGen.exe，这个工具可以将所有程序集的IL代码编译成本机代码并将编译好的本机代码保存在磁盘中。这样，CLR可以在加载程序集时加载这个预编译的代码，从而提高运行速度。

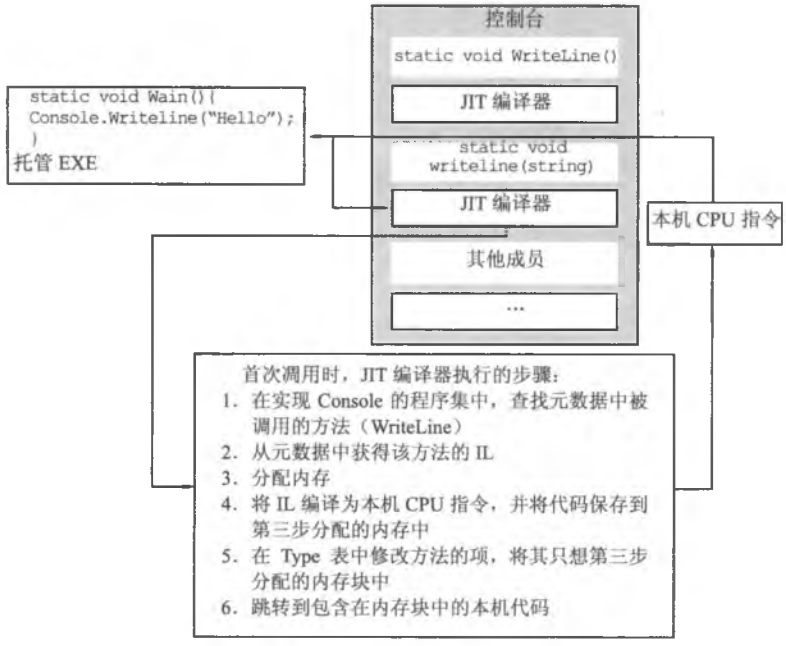

图5-8　首次调用方法的执行过程

当把IL编译成本机代码时，IL会执行验证的过程。验证会检查IL的代码并确保它们的安全性，例如：验证过程会检查被调用的方法参数和参数的类型是否匹配，检查方法的返回值是否被正确使用等。如果验证结果认为IL代码是“不安全的”则会抛出System.Security.VerificationException异常并阻止方法的执行。

4．通用类型系统

因为.NET 统一地对待所有的语言，所以在.NET 中，要求使用 C#书写的类同使用 VB.NET 书写的类能够互相使用，使用托管 C++和托管 COBOL 编写的接口也完全一样。由于所有的语言必须拥有共同的标准才能够顺利地集成到一起，因此 Microsoft 定义了通用类型系统 CTS（Common Type System）来规范每一个.NET 语言。有人认为 CTS 是 CLR 中最核心的部分，因为如果没有 CTS 的存在，.NET 的语言无关性就得不到体现。

CTS 中包括很多类型，下面我们介绍其中较为重要的几种类型：值类型、引用类型、类和接口，以及委托。

5．通用语言规范

我们都知道，每种语言都有自己的特性，例如有些语言（C++）是大小写敏感的，而 VB 则不区分大小写，为了实现语言级的继承，Microsoft 还制订了通用语言规范（Common Language Specification，CLS）。CLS 提供了可以作为.NET 语言的最小特性集，任何.NET 语言都必须支持这些特性。从语言特性角度考虑，CLR/CLS 与各种.NET 语言和 CLS 的关系如图 5-9 所示。

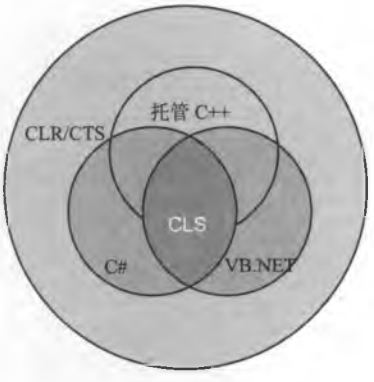

图 5-9　CLS 与.NET 语言的关系

CLR/CLS 提供了最广泛的语言特性，仅有极个别的语言可以完全实现这些特性，如中间语言 IL，一般的.NET 语言都是 CLR/CLS 的子集，这些语言都必须包含一个共同的子集 CLS——也是.NET 语言的最小特性集。如果需要使用多种语言混合开发，那么程序员就需要遵守 CLS 以保证书写的程序的互操作性。

6．垃圾收集

.NET 的垃圾收集机制与 Java 非常类似，但也有一些区别。.NET 将内存分为托管区域和非托管区域，.NET 中的垃圾收集器仅负责托管区域中的垃圾收集，同 Java 中的

技术类似，.NET 的垃圾收集器也是自动回收不可达的对象，同样也采用了分代复制的算法。但在具体实现中有细微的差别。

5.3　J2EE 和.NET 平台的异同

首先我们需要明确的是 J2EE 和.NET 的目标，这两个平台都是为了解决构建企业计算等大型平台而出现的。在这两个平台中都包含了一系列的技术，通过这些技术可以缩短开发周期，提高开发效率，节省构造成本，同时这两个平台都在安全性、扩展性、性能方面做出了努力，都提供了一系列的技术可供选择。从这个角度来说，这两个平台都实现了他们的目标，都是成功的。因为这两个平台要解决的问题类似，所以很多技术也非常类似，有些概念甚至仅仅是名称上的差别而已：Java 中包的概念和.NET 中的命名空间的概念。两个平台的类似之处远远多于相异之处。本章对这两个平台进行对比并不是想说明这两个平台的哪一个更优秀，事实上无论是 J2EE 还是.NET 都是优秀的平台解决方案，我们仅仅是通过对比的手段加深读者对 J2EE 和.NET 技术的理解，能够在工作中根据实际需要确定选用的平台和技术，构造合理的解决方案。J2EE 与.NET 平台的不同之处，见表 5-1。

表 5-1　J2EE 与.NET 对比表

	J2EE	.NET
跨平台	跨平台能力强	不具备跨平台能力，仅支持 Windows 系统
支持语言	Java	VB、C++、C#、Jscript，通过组件还可支持 Java
安全性	相对较好	一般
稳定性	相对较好	一般
服务器端 Unix 系统应用	很好	差
与 Windows 桌面系统集成性	一般	好
与 Windows 软件集成性	差	很好

对于需要进行平台选择的企业和开发者来说，根据自己的实际需要，才能做出最恰当的选择。