

.NET 框架程序开发指南

(上册)

张志学 等 编著

清华大学出版社

(京)新登字 158 号

内 容 简 介

.NET 框架是用于构建、配置、运行 Web 服务和应用程序的多语言环境,本书结合大量实例详细介绍了与.NET 框架应用程序开发相关的知识。全书共 13 章,主要内容包括:.NET 框架开发基础、构造和使用部件、配置应用程序域、收集冗码、使用基础类型、标志编程、操作字符串、使用集合组织数据、响应和激发事件、捕捉和抛出异常、对象序列化、操作和监测文件系统以及使用 ADO.NET 访问数据源等。

本书内容全面深入,适合中高级读者、大专院校师生、企业技术开发人员学习参考,也适合各类培训班学员学习.NET 框架编程技术。

版权所有,翻印必究。

本书封面贴有清华大学出版社激光防伪标签,无标签者不得销售。

图书在版编目(CIP)数据

.NET 框架程序开发指南. 上 / 张志学等编著. —北京:清华大学出版社, 2002.7
ISBN 7-302-05579-3

I. N... II. 张... III. 计算机网络-程序设计 IV. TP393

中国版本图书馆 CIP 数据核字(2002)第 041134 号

出 版 者: 清华大学出版社(北京清华大学学研大厦, 邮编 100084)

<http://www.tup.tsinghua.edu.cn>

责任编辑: 胡先福

印 刷 者: 北京鑫丰华彩印有限公司

发 行 者: 新华书店总店北京发行所

开 本: 787×1092 1/16 印张: 23.5 字数: 582 千字

版 次: 2002 年 7 月第 1 版 2002 年 7 月第 1 次印刷

书 号: ISBN 7-302-05579-3/TP·3295

印 数: 0001~5000

定 价: 35.00 元

前 言

.NET 框架是用于构建、配置、运行 Web 服务和应用程序的多语言环境。作为下一代互联网的主要开发平台，.NET 框架代表了一场新的技术革命。.NET 框架引入了将软件作为一种服务 (software as a service) 的新观点，其体系结构以 XML 为基础。

.NET 框架为开发者提供了统一的、面向对象的一组可扩展的层次类库 (APIs)。目前，C++ 开发者使用 MFC 类库，Java 开发者使用 WFC (Windows Foundation Classes) 类库，Visual Basic 开发者使用 Visual Basic APIs，而 .NET 框架则将这些完全不同的库统一起来。通过创建一组超越所有编程语言的通用 API，.NET 框架允许跨语言的继承、错误处理和调试。这样，从 JavaScript 到 C++ 的所有编程语言之间都被划上了等号，而开发者则可以自由地选择自己最拿手的语言进行开发。

为了帮助开发人员从总体上了解 .NET 框架，我们编写了《.NET 框架程序开发指南》，结合大量实例详细介绍了与 .NET 框架应用程序开发相关的知识。本书根据内容分为两个分册，上册介绍了 .NET 框架编程的基础，例如部件、应用程序域以及文件系统等的使用；下册主要涉及一些常用专题的开发，如数据库操控技术、GDI+ 编程技术、异步调用等。读者在使用本书学习 .NET 框架编程时，可以参考《.NET 框架开发人员参考手册》(清华大学出版社出版，分为系统构架、数据库、网络和 Internet、编程要素 4 个分册)，以便了解 .NET 框架编程所需的工具和函数。

本书并非只是知识点的简单罗列，而是通过实例全面讲解 .NET 开发的知识点和编程技巧，使得读者能够掌握并灵活运用这些知识点，迅速掌握这门新兴技术，从而能够开发出功能强大的 Windows/Web 常规应用程序和数据访问应用程序。

本书由张志学博士主要编写，参与写作、整理、调试的还有李瑞芬、胡佳禾、许少斌、杨威、钟心颜、谢雅丽、白雪松、潘银盆、孙一兵、刘伟、严丽芳、刘莹、罗靖、彭少民、郭燕、林彩霞、周涛、李韶辉、张秀霞等人。由于作者水平和经验所限，不足之处在所难免，恳请广大读者批评指正。

作 者

2002 年 5 月

目 录

第 1 章 .NET 框架开发基础	1
1.1 .NET 框架概述	1
1.1.1 .NET 框架的设计目标	1
1.1.2 公用语言运行库的性能	2
1.1.3 公用层次类库	4
1.2 .NET 框架的系统需求	6
1.2.1 操作系统需求	6
1.2.2 其他软件需求	6
1.2.3 硬件需求	7
1.3 .NET 框架编程简介	7
1.3.1 开发所需的工具	7
1.3.2 常用开发概念	8
1.3.3 客户应用程序开发	8
1.3.4 服务器应用程序开发	9
1.3.5 开发语言的差别	10
1.3.6 C#程序编码简介	12
1.3.7 理解受控执行	15
1.3.8 公用语言规范	16
1.4 ASP.NET 简介	19
本章小结	20
第 2 章 构造和使用部件	21
2.1 部件编程的基本知识	21
2.1.1 部件概述	21
2.1.2 元素清单	23
2.1.3 部件命名	24
2.1.4 部件标志	26
2.1.5 部件版本	27
2.1.6 载入优化	27
2.1.7 部件安全	28
2.1.8 创建工具	28
2.1.9 部署位置	29
2.2 构造部件	30
2.2.1 构造单文件部件	30

2.2.2	构造多文件部件	31
2.3	设置部件标志	33
2.3.1	部件身份标志	34
2.3.2	信息标志	34
2.3.3	部件元素清单标志	35
2.3.4	强名称标志	35
2.4	创建和使用强名称部件	35
2.4.1	何时使用强名称	35
2.4.2	创建密匙对	36
2.4.3	为部件指定强名称	37
2.4.4	引用强名称部件	37
2.4.5	延迟签署部件	38
2.5	使用公用部件缓存	39
2.5.1	为何使用公用部件缓存	39
2.5.2	将部件安装到公用部件缓存中	40
2.5.3	查看公用部件缓存中的内容	40
2.5.4	从公用部件缓存中删除部件	41
2.5.5	服务组件和公用部件缓存	41
2.6	查看部件内容	41
2.7	部件引用和解析	42
2.7.1	确定部件版本	43
2.7.2	定位部件	44
2.7.3	执行版本策略	47
2.8	发布动态部件	47
2.8.1	映像发布抽象	48
2.8.2	映像发布的使用环境	48
2.8.3	映像发布安全	50
2.9	使用映像发布	50
2.9.1	定义动态部件	51
2.9.2	定义动态模块	52
2.9.3	定义类型	52
2.9.4	定义枚举	53
2.9.5	定义构造函数	53
2.9.6	定义方法	54
2.9.7	定义字段	55
2.9.8	定义属性	55
2.9.9	定义事件	55
2.9.10	定义参数	55
2.9.11	定义字符串常量	56

2.9.12 发送 MSIL 指令	56
2.9.13 发送资源	56
2.9.14 发布符号信息	57
本章小结	57
第 3 章 配置应用程序域	58
3.1 应用程序域概述	58
3.2 应用程序域编程	59
3.2.1 创建应用程序域	59
3.2.2 卸载应用程序域	61
3.2.3 配置应用程序域	62
3.2.4 获取应用程序域的设置信息	63
3.2.5 将部件载入应用程序域	65
3.2.6 获取部件信息	66
3.3 运行库宿主的工作机制	67
3.3.1 载入运行库	67
3.3.2 受控宿主代码转换	67
3.3.3 确定应用程序域的边界	68
3.3.4 创建和配置应用程序域	69
3.3.5 载入和执行用户代码	69
3.3.6 设置应用程序域级的安全策略	70
3.3.7 设置角色安全策略和规则	70
3.3.8 卸载域和终止进程	71
本章小结	71
第 4 章 收集冗码	72
4.1 内存管理回顾	72
4.1.1 COM 开发	72
4.1.2 C++ 开发	72
4.1.3 Visual Basic 开发	73
4.2 冗码收集机制	73
4.2.1 冗码收集概述	73
4.2.2 析构函数	74
4.2.3 Visual Basic 对象的销毁	75
4.3 清除非受控资源	78
4.3.1 实现 Dispose 方法	79
4.3.2 实现 Close 方法	81
4.3.3 清除使用封装资源的对象	81
4.3.4 执行冗码收集	82
本章小结	83

第 5 章 使用基础类型	84
5.1 通用类型系统	84
5.1.1 类类型	84
5.1.2 接口类型	85
5.1.3 数值类型	85
5.1.4 delegate	86
5.2 类型定义层次	87
5.3 执行类型转换	92
5.3.1 转换概述	92
5.3.2 类型转换表	93
5.3.3 使用 System.Convert 进行转换	94
5.3.4 显式转换	95
5.3.5 基础类型编码	96
5.4 执行类型格式化	97
5.4.1 格式化概述	98
5.4.2 格式符和格式提供者	98
5.4.3 数字格式符	99
5.4.4 时间和日期格式符	99
5.4.5 枚举格式符	105
5.4.6 基础格式化	106
5.4.7 复合格式化	106
5.4.8 格式化多个对象	107
5.4.9 对齐	107
5.4.10 不同文化的格式化	108
5.4.11 定制格式	109
5.5 在运行时获取类型信息	112
5.5.1 映像概述	112
5.5.2 查看类型信息	113
5.5.3 映像的安全考虑	118
5.5.4 链接请求检查	119
5.5.5 类型信息的获取和浏览示例	119
5.6 动态载入和使用类型	124
5.6.1 定制绑定	125
5.6.2 访问默认成员	128
5.6.3 访问默认参数值	129
5.6.4 访问定制标志	130
5.7 指定全限定类型名	131
5.7.1 Backus-Naur 形式的语法和类型名	132
5.7.2 指定部件名	133

5.7.3 指定指针	134
5.7.4 指定引用	134
5.7.5 指定数组	134
本章小结	134
第 6 章 标志编程	136
6.1 标志概述	136
6.1.1 标志的用途	136
6.1.2 标志基类	137
6.2 使用标志	138
6.3 定制标志	140
6.4 获取标志信息	144
6.4.1 获取单个标志实例	145
6.4.2 获取施用于相同作用域的多个标志实例	146
6.4.3 获取施用于不同作用域的多个标志实例	146
本章小结	148
第 7 章 操作字符串	149
7.1 正则表达式	149
7.1.1 正则表达式概述	149
7.1.2 正则表达式的引擎	150
7.2 基础字符串操作	151
7.2.1 创建新字符串	151
7.2.2 裁减和删除字符	155
7.2.3 填充字符串	157
7.2.4 比较字符串	159
7.2.5 改变字符串的大小写	164
7.3 使用 <code>StringBuilder</code> 类	165
7.3.1 设置字符串的容量和长度	166
7.3.2 修改 <code>StringBuilder</code> 字符串	166
7.4 解析字符串	171
7.4.1 解析数字字符串	171
7.4.2 解析日期和时间字符串	172
7.4.3 解析其他字符串	173
本章小结	174
第 8 章 使用集合组织数据	175
8.1 集合概述	175
8.1.1 集合的特点	175
8.1.2 集合分类	176
8.2 常用集合类型编程	176

8.2.1	Array 集合类型	177
8.2.2	ArrayList 集合类型	182
8.2.3	Hashtable 集合类型	185
8.2.4	SortedList 集合类型	188
8.2.5	Queue 集合类型	190
8.2.6	Stack 集合类型	193
8.2.7	位集合	196
8.2.8	专用集合	200
8.3	创建和操作集合	200
8.3.1	选择集合类	201
8.3.2	枚举集合	201
8.3.3	集合同步化 (线程安全)	202
8.3.4	集合比较和排序	202
	本章小结	203
第 9 章	响应和激发事件	204
9.1	事件和 Delegate	204
9.2	响应事件	206
9.2.1	Web 窗体中的事件响应	206
9.2.2	Windows 窗体中的事件响应	209
9.2.3	常规事件模式	211
9.3	激发事件	212
9.3.1	激发单个事件	213
9.3.2	激发多个事件	220
	本章小结	221
第 10 章	捕捉和抛出异常	222
10.1	异常概述	222
10.1.1	异常分类	222
10.1.2	运行库如何管理异常	223
10.1.3	筛选运行库异常	224
10.1.4	异常基类	225
10.1.5	异常层次	225
10.2	捕捉异常	227
10.2.1	使用 try/catch 块捕捉异常	227
10.2.2	在 catch 块中捕捉指定异常	228
10.2.3	使用 finally 块	229
10.2.4	Visual Basic 的异常捕捉处理机制	230
10.3	抛出异常	235
10.4	定制异常	236

10.5 定制异常筛选	236
10.6 处理 COM 互用异常	237
10.7 异常处理设计示例.....	238
本章小结	240
第 11 章 对象序列化.....	241
11.1 二进制序列化.....	241
11.1.1 二进制序列化简介.....	241
11.1.2 基础序列化.....	242
11.1.3 选择性序列化.....	244
11.1.4 定制序列化.....	244
11.1.5 格式化器的序列化过程.....	246
11.1.6 版本检查	246
11.2 XML 序列化.....	247
11.2.1 XML 序列化简介	247
11.2.2 XML 序列化的优势.....	248
11.2.3 XSD 数据类型映射.....	249
11.2.4 序列化和反序列化对象.....	249
11.2.5 XML 名称空间.....	250
11.2.6 XML 序列化标志.....	252
11.2.7 XML 纲要定义工具.....	257
11.2.8 重载 XML 序列化.....	258
11.2.9 XML 序列化示例.....	259
11.3 XML Web 服务中的 XML 序列化	264
11.3.1 值和编码风格.....	264
11.3.2 使用 XML 序列化生成 SOAP 消息	266
11.3.3 控制编码 SOAP 序列化的标志	266
本章小结	268
第 12 章 操作和监测文件系统.....	269
12.1 基础文件 I/O	269
12.1.1 I/O 类派生自 System.Object.....	269
12.1.2 I/O 和安全	271
12.1.3 I/O 编程示例	271
12.2 异步文件 I/O	278
12.3 隔离存储	285
12.3.1 隔离存储简介	286
12.3.2 隔离存储的适用情况.....	286
12.3.3 隔离类型	287
12.3.4 用户或部件隔离.....	288

12.3.5	用户、域和部件隔离.....	289
12.3.6	隔离存储和漫游.....	289
12.3.7	隔离存储的配额.....	289
12.3.8	隔离存储的安全性.....	290
12.3.9	可执行的操作和安全风险.....	290
12.3.10	执行隔离存储任务.....	290
12.4	监测文件系统.....	301
12.4.1	文件系统事件监测简介.....	302
12.4.2	文件系统事件.....	302
12.4.3	配置 FileSystemWatcher 组件实例.....	303
12.4.4	创建文件系统事件处理函数.....	304
12.4.5	限制将监测的文件变化量.....	305
12.4.6	等待指定文件系统事件的发生.....	306
12.4.7	创建并监测定时器.....	306
	本章小结.....	308
第 13 章	使用 ADO.NET 访问数据源.....	309
13.1	ADO.NET 概述.....	309
13.1.1	ADO.NET 的设计目标.....	309
13.1.2	ADO.NET 结构.....	310
13.1.3	ADO.NET 平台需求.....	311
13.1.4	选择数据阅读器或数据集.....	311
13.2	.NET 数据提供者.....	312
13.2.1	SQL Server .NET 数据提供者.....	312
13.2.2	OLE DB .NET 数据提供者.....	313
13.2.3	选择 .NET 数据提供者.....	313
13.2.4	由 OLE DB .NET 数据提供者使用的 OLE DB 接口.....	314
13.2.5	使用 .NET 数据提供者访问数据.....	314
13.2.6	代码访问安全性.....	315
13.2.7	实现 .NET 数据提供者.....	317
13.3	连接 SQL Server 数据源.....	320
13.3.1	SqlConnection 对象.....	321
13.3.2	SQL Server .NET 数据提供者的连接池.....	322
13.3.3	添加连接.....	323
13.3.4	删除连接.....	323
13.3.5	事务支持.....	324
13.3.6	连接字符串.....	324
13.4	连接 OLE DB 数据源.....	324
13.4.1	OleDbConnection 对象.....	324

13.4.2	OLE DB .NET 数据提供者的连接池	326
13.4.3	处理连接事件	326
13.5	执行命令	327
13.5.1	获取数据	328
13.5.2	获取纲要信息	330
13.5.3	使用存储过程	330
13.5.4	使用参数	332
13.5.5	从数据库中获得单个值	334
13.5.6	从数据中获得 BLOB 值	334
13.5.7	执行数据库操作和修改数据	336
13.5.8	执行编目操作	337
13.5.9	以 XML 的形式获取 SQL Server 数据	337
13.6	使用数据适配器	338
13.6.1	使用单个数据适配器填充数据集	338
13.6.2	使用多个数据适配器填充数据集	339
13.6.3	数据类型映射	341
13.6.4	更新数据库	343
13.6.5	添加现存约束	346
13.6.6	设置数据表和数据列映射	346
13.6.7	在数据适配器中使用参数	348
13.6.8	控制映射方式	351
13.6.9	自动生成命令	353
13.6.10	响应数据适配器事件	355
13.6.11	获取纲要信息	358
13.6.12	执行事务	358
13.7	示例应用程序	360
	本章小结	361

第 1 章 .NET 框架开发基础

.NET 框架是用于构建、配置、运行 Web 服务和应用程序的多语言环境。作为下一代互联网的主要开发平台，.NET 框架代表了一场新的技术革命。.NET 框架引入了将软件作为一种服务 (software as a service) 的新观点，其体系结构以 XML 为基础。本章将向读者介绍 .NET 框架的概述、组成结构以及内部机制，从而使读者对 .NET 框架有深入的理解，这也是在 .NET 框架中进行软件开发的重要基础。

本章要点：

- ❖ .NET 框架的基础知识
- ❖ 在 .NET 框架中进行开发的基础知识
- ❖ .NET 框架的系统需求

1.1 .NET 框架概述

.NET 框架是一种新型计算机平台，它能够简化高度分布式的 Internet 环境下的应用程序开发。.NET 框架包括两个主要组件：公用语言运行库和 .NET 框架类库。公用语言运行库是 .NET 框架的基础。您可以将运行库看作在执行时，用于管理代码的媒介。运行库能够提供诸如内存管理、线程管理等核心服务，并能强制施用严格的类型安全和其他形式的代码精度规则，从而确保代码的安全和稳定。事实上，代码管理的概念是运行库的基础准则。根据运行库设计的代码被称为受控代码，反之则被称为非受控代码。.NET 框架的另一个主要组件——类库，是一种面向对象的、全面的可用类型集合。使用此类库，能够开发从传统的命令行或图形用户界面 (GUI) 应用程序，到基于最新的 ASP.NET 的应用程序，例如 Web 窗体和 XML Web 服务。

1.1.1 .NET 框架的设计目标

.NET 框架是为了满足以下目标而设计的：

- 提供可靠的面向对象的编程环境，无论对象代码是在本地储存和运行，或分布在 Internet，并在本地运行；还是分布在 Internet 上，并在远程运行。
- 提供能最大限度减少开发和版本冲突的代码运行环境。
- 提供能确保代码安全执行（包括由未知或不具备完全信任度的第三方创建的代码）的代码运行环境。
- 提供消除了脚本和解释环境的性能问题的代码运行环境。
- 使得开发者在不同类型的应用程序（例如基于 Windows 的应用程序和基于 Web 的应用程序）的开发中得到一致的体验。

- 根据工业标准构造所有通信，以确保基于 .NET 框架的代码能与其他任何代码相互整合。

1.1.2 公用语言运行库的性能

.NET 框架提供了一种运行时环境——公用语言运行库，它对于组件的运行和开发起着重要的作用。公用语言运行库管理内存、线程运行、代码运行、代码安全验证、编译和其他系统服务。这些是受控代码的内建性能，并运行于公用语言运行库中。当组件在运行时，运行库负责管理内存分配、启动、结束进程、执行安全策略以及满足组件间可能存在的依赖性。而在开发时，运行库的作用会有所变化。由于其高度的自动化程度（例如内存管理），运行库使程序员感觉开发变得异常简单，在与目前使用的 COM 相比时尤其如此。除此之外，诸如映像这样的功能，极大地降低了将商用逻辑转化为可重用的组件时的代码编写量。

公用语言运行库负责管理代码的运行，并提供使开发得到简化的服务。编译器和工具负责解释运行库的功能，并允许用户编写能够从这种管理运行环境中得益的代码——受控代码（Managed Code）。受控代码能够使用跨语言集成、异常处理、安全强化、版本和安装支持、简化的组件交互模型以及调试和优化等由公用语言运行库提供的服务。

只有当编译器将元数据（Metadata）翻译出后，运行库才能为受控代码提供服务——描述代码中的引用、成员和类型。元数据与代码保存在一起；每个可载入的公用语言运行库的映像中都包含元数据。运行库使用元数据来完成类的定位和载入、实例在内存中的分布以及设置运行环境边界等操作。

运行库能够自动处理对象规划，并管理对对象的引用。当对象使用完毕后，运行库负责将其释放。这种由运行库管理其生命周期的对象被称为受控数据（Managed Data）。使用受控数据时，自动内存管理将负责消除内存漏洞，以及其他常见的编程错误。如果 .NET 应用程序使用的是受控代码，则可在其中使用受控数据，当然如果需要也可以使用非受控数据，或者同时使用这两种数据。由于语言编译器往往都有自己的类型，因此可能无法得知自己的数据是否处于受控状态。

公用语言运行库简化了需要进行跨语言交互的组件和应用程序的设计。这不但使得由不同语言编写的对象之间能够进行通信，而且还能牢固地整合其行为。例如，可以使用一种语言定义一个类，然后使用另一种语言生成该类的派生类，或调用该类中的方法；也可以将某类的实例传递给用不同语言编写的另一个类的方法。正是由于语言编译器和工具使用了由运行库定义的通用类型系统，才使得这种跨语言的整合得以实现。语言编译器和工具必须遵循运行库规则进行类型的定义、创建、使用、维护和绑定。

作为元数据的一部分，所有的受控组件都包含组件本身及构建所用的资源信息。运行库使用这些信息，来确定组件或应用程序是否满足特定版本所需的条件，这样就会大大降低由于版本冲突等依赖性问题所导致的运行失败。注册信息和状态数据不再保存在难于构建和维护的注册表中，而是作为元数据储存在代码里，从而简化组件的复制和删除。

语言编译器和工具的运行库功能，是通过有利于开发者的途径来表现的。这意味着一些运行库功能可能会由于环境的不同而体现出不同的重要性。因此，用户对于运行库的感受与其所用的语言编译器和工具息息相关。例如，对于一个 Visual Basic 开发者来说，可能

会注意到由于公用语言运行库的使用,使得 Visual Basic 语言具有了更多的面向对象的特性。下面列出了一些使用运行库可能带来的极具吸引力的方面:

- 程序性能得到提高。
- 更加容易使用由其他语言开发的组件。
- 类库提供的可扩展的类型。
- 语言功能扩展得更广。

如果使用 Visual C++, 则可以使用 C++ 受控扩展来编写受控代码。这样就能使 Visual C++ 程序也能得到受控运行环境所提供的服务, 并可以使用更强大的功能和数据类型:

- 多语言整合, 尤其是跨语言继承。
- 自动内存管理, 它负责控制对象的生命周期, 这使得不再需要使用引用数。
- 自描述对象, 这使得不再需要使用 IDL (接口定义语言, Interface Definition Language)。
- 应用程序可以只编译一次, 而能够运行于任何支持运行库的操作系统和 CPU 之上。

在安全方面, 受控组件能采用多种级别, 这取决于很多因素, 其中包括它们的源 (例如 Internet、企业网络或本地网络)。这意味着受控组件可能或不可能执行文件访问操作、注册表访问操作, 或其他敏感功能, 即使它正被同一活动应用程序使用。

运行库使得代码访问安全。例如, 用户可以信任 Web 页中嵌入的、能播放动画或乐曲的可执行模块, 但该模块不能访问用户的私人数据、文件系统或网络。因此, 运行库的安全性能使合法的 Internet 软件具有非凡、丰富的功能。

运行库还通过实现精确类型和代码验证架构——公用类型系统 (CTS, Common Type System), 保证了代码的稳固和强壮。CTS 确保了所有受控代码都是自描述的。各种微软和第三方语言编译器都生成遵守 CTS 的受控代码。也就是说, 受控代码能够使用其他受控类型和实例, 并保证类型精度和类型安全。

此外, 运行库的受控环境消除了许多常用软件问题。例如, 运行库自动处理对象配置、管理对象引用, 并当对象使用完毕后释放它们。这种自动内存管理解决了两个常见的应用程序错误: 内存漏洞和无效内存引用。

运行库还能提高开发者的工作效率。例如, 程序员能够选择合适的编程语言编写应用程序, 并能充分利用运行库的优势、类库和由其他开发者编写的组件。所有利用 .NET 框架的编译器生产商, 都能为代码使用可用的 .NET 框架性能, 从而简化了现存应用程序的升级进程。

虽然运行库是为未来的应用程序设计的, 它依然能支持当今和以前的软件。受控和非受控代码间的互用性, 允许开发者能继续使用必需的 COM 组件和 DLL。

运行库的设计目的之一是为了提高性能。虽然公用语言运行库提供了许多标准运行库服务, 但是受控代码从不会被解释。一种被称为即时 (JIT, Just In Time) 编译的性能, 允许所有受控代码能以系统的原始机器语言的形式运行。同时, 内存管理器删除了内存碎片, 并提高了内存引用定位以进一步提高性能。

最后, 运行库能够由高性能的服务端应用程序管理, 例如 SQL Server 和 IIS。这种架构允许您使用受控代码来编写商业逻辑, 同时依然能利用支持运行库的工业最佳企业服务器的性能。

如果非受控组件将公用语言运行库载入其进程, 并初始化受控代码的执行, 那么非受

控组件也可以作为 .NET 框架的宿主, 从而创建出能同时利用受控和非受控特性的软件环境。 .NET 框架不但提供了一些运行库宿主, 而且支持开发第三方运行库宿主。例如, ASP.NET 管理运行库可为受控代码提供可缩放的、服务器方环境, 它与运行库一起运作 Web 窗体应用程序和 XML Web 服务。

IE 是一个管理运行库的非受控应用程序的例子 (一种 MIME 类型的扩展形式)。这种方式允许将受控组件或 Windows 窗体控件嵌入到 HTML 文档中, 从而使受控活动代码 (类似于 ActiveX 控件) 成为可能, 并提供了只有受控代码才具有的许多改进功能 (例如, 半信任执行和安全隔离文件存储)。

1.1.3 公用层次类库

.NET 框架为开发者提供类统一的、面向对象的一组可扩展的层次类库 (APIs)。目前, C++ 开发者使用 MFC 类库; Java 开发者使用 WFC (Windows Foundation Classes) 类库; Visual Basic 开发者使用 Visual Basic APIs; 而框架则将这些完全不同的库统一起来。通过创建一组超越所有编程语言的通用 API, .NET 框架允许跨语言的继承、错误处理和调试。这样, 从 JavaScript 到 C++ 的所有编程语言之间都被划上了等号, 而开发者则可以自由地选择自己最拿手的语言进行开发。

.NET 框架类库是与公用语言运行库紧密整合的、可重用类型的集合。类库是面向对象的, 并为受控代码提供它们能使用的功能。这不但使得 .NET 框架类型更易于使用, 并且减少了学习 .NET 框架新性能的时间。此外, 第三方组件还能与 .NET 框架类无缝整合。

例如, .NET 框架类集合实现了一套接口, 以供程序员开发自己的类集合。您的类集合将能与 .NET 框架类无缝互用。

就像您对面向对象的类库的期望那样, .NET 框架类型能完成很多常用编程任务, 这包括字符串管理、数据集合、数据库连接以及文件访问等。除了这些常用任务外, 类库 (包括类型) 还支持很多特定的开发任务。例如, 可以使用 .NET 框架开发以下类型的应用程序和服务:

- 控制台应用程序。
- 脚本或受控应用程序。
- Windows GUI 应用程序 (Windows 窗体)。
- ASP.NET 应用程序。
- XML Web 服务。
- Windows 服务。

例如, Windows 窗体类为能简化 Windows GUI 开发的一套可重用类型。在编写 ASP.Web 窗体应用程序时, 您可以使用 Web 窗体类。

.NET 框架中包含类、接口和数值类型, 它们可以加速和优化开发进程, 并允许程序员利用系统功能。为了支持语言互用, .NET 框架类型都与 CLS (Common Language Specification, 公用语言规范) 兼容, 并且能用于任何支持动态语言运行库的编译器。

.NET 框架中的类型有很多的功能, 例如, 封装数据结构、执行 I/O 操作、访问数据、控制服务器、获取类信息以及激活安全检查等等。类型是 .NET 框架应用程序、组件和控件