.NET Framework

Essentials

Second Edition

Thuan Thai & Hoang Q. Lam

O'REILLY®

Платформа .NET

Основы

Второе издание

Туан Тай, Хонг К.Лэм

Санкт-Петербург
2003

Туан Тай, Хонг К. Лэм

Платформа .NET. Основы, 2-е издание

Перевод Л. Фрейдина

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*А. Королев*
Редактор	*В. Овчинников*
Корректура	*С. Беляева*
Верстка	*Н. Гриценко*

Тай Т., Лэм Х. К.
Платформа .NET. Основы. – Пер. с англ. – СПб: Символ-Плюс, 2003. – 336 с., ил.
ISBN 5-93286-054-5

Перед вами краткое введение в платформу Microsoft .NET, призванное помочь разработчикам перейти от традиционного программирования для Windows к созданию приложений в среде .NET. В книге подробно описаны общеязыковая среда выполнения Common Language Runtime (CLR) и набор базовых классов, радикально упрощающих разработку крупномасштабных приложений. Рассмотрен механизм языковой интеграции и приведено описание цикла компонентной и корпоративной разработки с использованием .NET Framework. Кроме того, обсуждаются основы ключевых технологий .NET: работа с данными (ADO.NET) и XML, веб-службы (Web Services), веб-формы (Web Forms, ASP.NET) и Windows Forms.

Книга предназначена в основном для разработчиков, имеющих опыт в программировании COM, создании объектно-ориентированных, компонентных и корпоративных Windows-приложений с помощью языков Visual Basic, Visual C++, Java™ и C/C++, но может быть полезна всем желающим изучать Microsoft .NET Framework.

ISBN 5-93286-054-5
ISBN 0-596-00302-1 (англ)

Издательство «Символ-Плюс». 193148, Санкт-Петербург, ул. Пинегина, 4,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 14.10.2002. Формат 70x100 1/16 . Печать офсетная.
Объем 21 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие

Данная книга представляет собой краткое введение в Microsoft .NET Framework и предназначена для того, чтобы помочь программистам осуществить переход от традиционного программирования для Windows к миру программирования в среде .NET. Microsoft .NET Framework включает Common Language Runtime (CLR, общеязыковая среда выполнения) и набор базовых классов, радикально упрощающих разработку крупномасштабных приложений и сервисов. Книга детально описывает CLR, так что вы сможете активно применять ее новые возможности. В ней показано, как в действительности работает языковая интеграция, и приведено описание цикла компонентной и корпоративной разработки с использованием .NET Framework. Кроме того, имеется введение в четыре ключевых технологии .NET: работу с данными (ADO.NET) и XML, веб-сервисы (Web Services), веб-формы (Web Forms, ASP.NET) и Windows Forms.

Для подготовки рукописи, создания всех примеров и рисунков в этой книге мы использовали последнюю версию Microsoft Visual Studio .NET и .NET Framework SDK. Хотя мы старались сделать все возможное, чтобы техническое содержимое книги было актуальным, возможно, что кое-что с момента написания книги изменилось. Чтобы все время быть в курсе изменений, регулярно посещайте сайты *http://msdb.microsoft.com/net*, *http://www.gotdotnet.com* и страницу этой книги на сайте O'Reilly, *http://www.oreilly.com/catalog/dotnet_frmess2/*.

Наша аудитория

Хотя данная книга предназначена для всех желающих узнать о Microsoft .NET Framework, она нацелена на разработчиков с опытом создания Windows-приложений с помощью Visual Studio 6 и языков Visual Basic и Visual C++. Разработчики на Java™ и C/C++ также хорошо подготовлены для представленного здесь материала. Чтобы получить от книги максимум, вам следует иметь опыт в разработке объектно-ориентированных, компонентных, корпоративных и веб-приложений. Опыт в программировании COM также будет весьма полезным.

Об этой книге

Для тех, кто не знаком с каждой из описанных технологий, мы построили эту книгу, основываясь на кратком курсе, который Туан (Thuan) читал в нескольких компаниях с августа 2000 года, таким образом, что каждая глава базируется на знаниях из предыдущей главы. Чтобы дать вам общий обзор, приведем краткие аннотации глав и приложений этой книги.

Глава 1 «Обзор .NET» содержит краткое описание платформы Microsoft .NET. Она рассказывает о целях разработки .NET Framework и знакомит вас с компонентами .NET Framework.

В главе 2 «Common Language Runtime» мы приподнимем «капот» и заглянем внутрь CLR. Здесь рассматривается богатая библиотека времени выполнения CLR, а также другие средства.

В главе 3 «Программирование в .NET» содержится введение в .NET-программирование. Вы изучите простую программу, использующую концепцию объектно-ориентированной и компонентной разработки на четырех различных языках: Managed C++, VB.NET, C# и IL. Вы также сможете испытать преимущества языковой интеграции.

Глава 4 «Работа с .NET-компонентами» демонстрирует простоту компонентной и корпоративной разработки в .NET. Кроме рассмотрения средств развертывания компонентов, вы также исследуете полноценные программы, использующие преимущества транзакций, пула объектов, безопасности на основе ролей и очереди сообщений – и все в одной главе.

В главе 5 «Данные и XML» описана архитектура ADO.NET и ее преимущества. Кроме возможности отсоединения, что способствует улучшению масштабируемости, набор данных ADO.NET также тесно интегрирован с XML, что улучшает возможности взаимодействия между платформами. Глава знакомит вас с объектами .NET для доступа к данным, а также с пространством имен XML.

В главе 6 «Веб-службы» описано следующее поколение программных компонентов, к которым можно осуществлять доступ через Интернет. В этой главе мы обсудим протоколы, поддерживаемые веб-службами, а также опишем, как публиковать и находить их. Вы узнаете, каким образом XML, используемый в связке с HTTP, ломает закрытую природу текущей модели компонентно-ориентированной разработки программного обеспечения и обеспечивает лучшие возможности межсетевого взаимодействия.

В главе 7 «Веб-формы» дано введение в среду ASP.NET, которая теперь поддерживает объектно-ориентированное и управляемое событиями программирование, в отличие от традиционной разработки ASP.

В этой главе центральное место занимают веб-формы и серверные элементы управления. Кроме того, вы узнаете, как создавать собственные серверные элементы управления, выполнять привязку данных к различным элементам управления .NET, и изучите средства управления состоянием в ASP.NET.

Глава 8 «Windows Forms» переносит традиционное программирование на основе форм на шаг в будущее с помощью классов пространства имен System.Windows.Forms. Аналогично Win32-приложениям, Windows Forms лучше всего использовать для так называемых мощных или «толстых» клиентов, однако с новой, упрощенной процедурой инсталляции в .NET и приходом веб-сервисов Windows Forms стали пригодными для более широкого круга приложений.

Приложение A «Языки .NET» содержит список ссылок на веб-сайты с информацией о языках, использующих CLR, включая некоторые находящиеся в зачаточном состоянии проекты с открытым исходным кодом.

Приложение B «Основные сокращения» содержит список часто используемых сокращений, применяемых в литературе о .NET и на презентациях.

Приложение C «Общие типы данных» содержит несколько списков наиболее часто используемых в .NET типов данных. В этом приложении также проиллюстрировано применение некоторых классов коллекций.

В приложении D «Основные утилиты» приведен обзор наиболее важных утилит, предоставляемых .NET SDK для облегчения разработки в среде .NET.

Теперь, когда вы знаете, о чем эта книга, мы должны объяснить, чего в ней нет. В этой книге мы не обращаемся к маркетинговым аспектам .NET или других компонентов платформ .NET, таких как .NET Enterprise Servers, .NET Building Block Services или операционных систем .NET. Кроме того, не рассмотрен недавно объявленный сервис HailStorm, а также деятельность Microsoft по обеспечению доступности .NET Framework на многочисленных типах устройств.

Требования к читателю

В этой книге предполагается, что вы разработчик Windows- и веб-приложений, свободно разбирающийся в объектно-ориентированном и компонентном программировании. Также предполагается, что вы имеете базовые знания по XML. Хотя знание COM не является обязательным требованием, если вы имеете опыт в программировании COM, то в еще большей степени оцените эту книгу и .NET Framework.

Соглашения, используемые в этой книге

В этой книге приняты следующие соглашения о шрифтовом оформлении.

Курсив используется для:

- имен каталогов, файлов и программ;
- интернет-адресов, таких как имена доменов и URL;
- новых терминов там, где они определяются.

`Моноширинный шрифт` применяется для:

- командных строк и параметров, которые требуют дословного написания;
- прямых цитат и конкретных имен методов из примеров кода, а также конкретных значений атрибутов и параметров в коде;
- тегов XML-элементов.

`Моноширинный жирный шрифт` применяется для:

- вводимых пользователем данных в коде, которые должны печататься дословно;
- мест в коде, на которые мы бы хотели обратить внимание читателя.

`Моноширинный курсив` применяется для элементов кода, которые следует заменить на конкретные значения.

В примерах синтаксиса кода мы иногда будем использовать обозначение `[`*`value`*`]+` для одного или более экземпляров значения value и `[`*`value`*`]*` для обозначения нуля или более экземпляров значения value.

Как с нами связаться

Мы, насколько могли, аккуратно протестировали и сверили информацию из этой книги, однако вы можете обнаружить произошедшие изменения (или даже допущенные нами ошибки!). Просьба сообщать нам о любых найденных ошибках, а также о предложениях по будущим изданиям, написав по адресу:

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (FAX)

Вы также можете присылать нам и электронные сообщения. Чтобы включить свой адрес в список рассылки или запросить каталог, отправьте письмо по адресу

info@oreilly.com

Чтобы задать технические вопросы или прислать комментарии к книге, отправьте письмо по адресу

bookquestions@oreilly.com

У нас есть веб-сайт этой книги, где мы поместим примеры, найденные ошибки и любые планы будущих изданий. Эта страница доступна по адресу

http://www.oreilly.com/catalog/dotnetfrmess2/

Дополнительную информацию по этой и другим книгам можно найти на веб-сайте издательства O'Reilly

http://www.oreilly.com

Дополнительная информация о среде .NET в целом приведена на .NET-центре издательства O'Reilly *http://dotnet.oreilly.com* и на *.NET DevCenter по адресу http://www.oreillynet.com/dotnet/*.

Благодарности

Сотрудники издательства O'Reilly не устают поражать нас предоставляемой ими поддержкой. Я хотел бы поблагодарить Джона Осборна (John Osborn), продлившего договор для написания этой книги, за постоянную поддержку проекта. Мы также хотим поблагодарить Нэнси Котари (Nancy Kotary) за выполненную трудную работу, позволившую обеспечить выход книги в установленные жесткие сроки. Нэнси провела большую работу по рецензированию наших материалов и координации проекта. Без Джона и Нэнси эта книга никогда не была бы написана. Спасибо сотрудникам O'Reilly, занимающимся производством и дизайном, за воплощение этой книги в реальность: Эмме Колби (Emma Colby), Татьяне Диаз (Tatiana Diaz), Дэвиду Футато (David Futato), Колину Горману (Coleen Gorman), Роберту Романо (Robert Romano), Майку Сьерра (Mike Sierra), Элли Волькхаузен (Ellie Volckhausen) и Джо Визда (Joe Wizda).

Спасибо Брайану Джепсону (Brian Jepson), внесшему значительный вклад в эту книгу с самого начала проекта. Брайан проделал, безуслов-

но, выдающуюся работу по чтению, тестированию и обеспечению соответствия технического содержимого каждой главы последнему релизу. Мы также хотим поблагодарить Деннис Анжелин (Dennis Angeline) и Брэда Меррила (Brad Merrill) из Microsoft за ответы на технические вопросы по CLR и языкам программирования.

Хоанг (Hoang) хочет поблагодарить своих родителей и семью за их поддержку и понимание того, что он выпал из их жизни на несколько месяцев. Мама и папа, ваши постоянные усилия, чтобы дети оказались там, где они сейчас, никогда не могут быть оплачены. Хоанг также благодарит свою жену, ВанДу (VanDu), источник своего вдохновения. Не надо недооценивать ее вклад в эту книгу. И, наконец, последнее, но не менее важное, персональная *благодарность* Туану (Thuan), который все время подталкивал меня к долгожданному завершению книги.

1

Обзор .NET

Корпорация Microsoft объявила об инициативе .NET в июле 2000 года. Платформа .NET – это новая среда разработки с новым интерфейсом программирования для доступа к службам и API Windows, интегрированная с рядом технологий, вышедших из Microsoft в конце 1990-х годов. В .NET объединены службы компонентов COM+; среда веб-разработки ASP; приверженность XML и объектно-ориентированному дизайну; поддержка новых протоколов веб-сервисов, таких как SOAP, WSDL и UDDI; а также ориентация на Интернет.

Платформа состоит из четырех отдельных групп продуктов:

Средства разработки

Набор языков, включающий C# и VB.NET; набор инструментальных средств разработки, в том числе Visual Studio.NET; всеобъемлющая библиотека классов для построения веб-сервисов, веб- и Windows-приложений; а также среда Common Language Runtime (CLR, общеязыковая среда выполнения) – для исполнения объектов, построенных в рамках этой структуры.

Специализированные серверы

Семейство серверов .NET Enterprise Servers, ранее известных как SQL Server 2000, Exchange Server 2000, BizTalk 2000 и т. д., обеспечивающих хранение реляционных баз данных, функции электронной почты и B2B-коммерции.

Веб-сервисы

Предоставление коммерческих веб-сервисов, в частности инициатива .NET My Services (ранее называвшаяся HailStrom); за плату разработчики могут применять эти сервисы для построения приложений, требующих идентификации пользователей.

Устройства

Новые устройства, поддерживающие .NET, от мобильных телефонов до игровых приставок.

Microsoft выделяет значительные ресурсы для разработки и поддержания успеха .NET и связанных с ней технологий: корпорация делает ставку на .NET как на следующий шаг в развитии компьютеризации.

Microsoft .NET

Microsoft потратила последние четыре года на создание платформы Microsoft .NET, представленной на конференции PDC 2000 в Орландо, штат Флорида. Хотя основной стратегией .NET было создание программного обеспечения в виде сервисов, .NET – это нечто большее. Кроме того, что она ориентирована на работу со Всемирной паутиной, Microsoft .NET соответствует следующим современным тенденциям программной индустрии:

Распределенные вычисления

Упрощает разработку надежных приложений клиент-сервер. Современные технологии распределенных вычислений требуют значительного сближения разных разработчиков и обеспечения возможности взаимодействия разных платформ. Microsoft .NET предоставляет архитектуру взаимодействия, использующую открытые стандарты Интернета, включая HTTP (Hypertext Transfer Protocol), XML (Extensible Markup Language) и SOAP (Simple Object Access Protocol).

Компонентное представление

Упрощение интеграции программных компонентов, созданных разными разработчиками. Технология COM (Component Object Model) воплотила в реальность программный «plug-and-play», однако разработка и развертывание COM-компонентов слишком сложны. Microsoft .NET предоставляет более простой способ построения и развертывания компонентов.

Корпоративные службы

Позволяет разрабатывать масштабируемые корпоративные приложения, не программируя управление транзакциями, средства безопасности или организацию пулов. Microsoft .NET продолжает поддерживать корпоративные службы, сокращающие время разработки и усилия, затрачиваемые на построение крупномасштабных приложений.

Смещение парадигмы Web

Внесены изменения в веб-технологии, упрощающие разработку приложений для Интернета. За последние несколько лет разработка веб-приложений сместилась от коммуникации (TCP/IP) к представлению (HTML) и к программируемости (XML и SOAP). Ключевой

задачей Microsoft .NET является обеспечение возможности продажи и распространения программного обеспечения как сервиса.

Факторы зрелости

Видно, какие уроки извлекла программная индустрия из разработки крупномасштабных корпоративных и веб-приложений. Коммерческое веб-приложение должно обеспечивать открытость, масштабируемость, доступность и управляемость. Microsoft .NET достигает всех этих целей.

Хотя здесь перечислены основные концепции Microsoft .NET, важно также отметить, что Microsoft .NET использует в своем ядре открытые стандарты Интернета (HTTP, XML и SOAP) для передачи объектов от одной машины к другой через Интернет. Фактически в .NET существует двухстороннее соответствие между XML и объектами. Например, класс может быть описан в виде определения XML-схемы (XSD); объект может быть преобразован в формат XML и обратно; метод может быть задан в XML-формате, называемом Web Services Description Language (WSDL, язык описания веб-сервисов); а вызов метода выражен в XML-формате SOAP.

Платформа .NET

Платформа .NET состоит из пяти основных компонентов, как показано на рис. 1.1. На самом нижнем уровне находится операционная система (ОС), которая может быть одной из нескольких Windows-платформ, включая Windows XP, Windows 2000, Windows Me и Windows CE. Как часть стратегии .NET, Microsoft обещает предоставить программное обеспечение для других .NET-устройств, чтобы способствовать появлению нового поколения «умных» устройств.

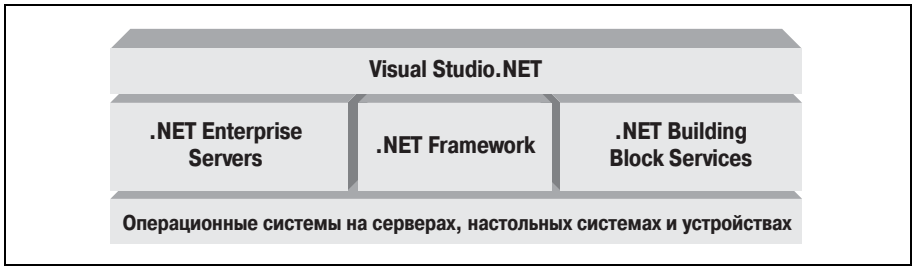

Рис. 1.1. *Платформа Microsoft .NET*

Поверх операционной системы функционирует серия продуктов .NET Enterprise Servers, сокращающих время, требуемое для разработки крупномасштабных бизнес-систем. Эти серверные продукты включают Application Center 2000, BizTalk Server 2000, Commerce Server 2000, Exchange Server 2000, Host Integration Server 2000, Internet Security Acceleration Server 2000 и SQL Server 2000.

Поскольку веб-сервисы могут активно применяться через Интернет, Microsoft планирует предоставить сервисы, являющиеся строительными блоками, которые разработчики приложений смогут использовать за плату. Примером такого сервиса является Microsoft Passport, позволяющий установить единое имя пользователя и пароль на всех сайтах, поддерживающих аутентификацию через Passport. В марте 2001 года Microsoft объявила об еще одном наборе веб-сервисов под кодовым названием HailStorm, которые теперь называются .NET My Services. В этом продукте заключен набор сервисов – строительных блоков, поддерживающих персонализацию, полностью основанную на последовательном опыте пользователя.[1] Microsoft планирует добавить и дополнительные сервисы, такие как календарь, каталог и поисковый сервис. Сторонние поставщики также создают собственные веб-сервисы.

На верхнем уровне архитектуры .NET находится совершенно новое средство разработки под названием Visual Studio.NET (VS.NET), которое делает возможным быструю разработку веб-сервисов и других приложений. Преемник Visual Studio 6.0, VS.NET, представляет собой интегрированную среду разработки (Integrated Development Environment, IDE), поддерживающую четыре различных языка и такие средства, как межъязыковая отладка и редактор XML-схем (XML Schema Editor).

А в центре .NET располагается Microsoft .NET Framework – основной предмет рассмотрения этой книги и новая инфраструктура разработки и исполнения, которая изменит создание бизнес-приложений на платформе Windows. Она включает общеязыковую среду выполнения Common Language Runtime (CLR) и общую структуру классов, которые могут использоваться всеми языками .NET.

Цели разработки .NET Framework

При разработке .NET Framework Microsoft преследовала много практических и исключительно амбициозных целей. В этом разделе мы обсудим главные задачи разработки Microsoft .NET Framework, включая улучшенную поддержку компонентов, межъязыковую интеграцию, обеспечение взаимодействия приложений через киберпространство, простую разработку и развертывание, улучшенную надежность и повышенную безопасность.

Компонентная инфраструктура

До появления технологии COM разработчики, использующие средства Microsoft, не имели простого способа интеграции бинарных библиотек

[1] Дополнительную информацию о .NET My Services можно будет найти в планируемой к публикации книге «.NET My Services Essentials» Калберта и Мэрфи (Culbert and Murphy, O'Reilly).

без ссылки на их исходный код или без его изменения. С введением COM программисты смогли интегрировать бинарные компоненты в свои приложения, аналогично аппаратным компонентам plug-and-play в вашем настольном компьютере. Хотя технология COM была хороша, неприятные детали ее применения были источником массы неприятностей для разработчиков и администраторов.

Хотя COM позволяет интегрировать бинарные компоненты, разработанные с использованием любого языка, эта технология требует соблюдения правил идентификации, времени жизни и двоичной структуры COM. Необходимо также писать код «заплаток», требуемых для создания COM-компонента, таких как DllGetClassObject, CoRegisterClassObject и т. д.

Понимая, что эти требования приводят к частому повторному написанию схожего кода, .NET решается избавиться от них. В мире .NET все классы готовы к использованию на двоичном уровне. Нет необходимости в написании дополнительного «заплаточного» кода для поддержки компонентного представления объекта в .NET Framework. Вы просто пишете класс .NET, который затем становится частью сборки (мы обсудим это в главе 2) и поддерживает plug-and-play.[1]

Кроме предоставления среды, облегчающей разработку, .NET устраняет трудности создания COM-компонентов. В частности, в .NET не требуется изменять реестр для регистрации компонентов и исключается необходимость в обязательном дополнительном коде, присутствующем во всех COM-компонентах, включая код для поддержки интерфейса IUnknown, фабрик классов, времени жизни компонента, регистрации, динамического связывания и прочего.

«Компонент» – неудачное слово, так как один человек может использовать его в отношении объекта, а другой – в отношении двоичного модуля. Чтобы исключить противоречия, в этой книге термин «COM-компонент» (или просто «компонент») означает двоичный модуль, такой как DLL или EXE.

Языковая интеграция

COM поддерживает *языковую независимость* (*language independence*), означающую, что вы можете разрабатывать COM-компонент на любом

[1] COM все еще играет свою роль в .NET Framework. Фактически, если вы воспользуетесь утилитой *dumpbin.exe* для получения дампа исполняемого файла (PE), созданного компиляторами, доступными в предварительном релизе или версии Beta 1 .NET SDK, то увидите некоторые остатки COM, в частности, упоминание чего-то с именем `COM+ Header`. Дополнительную информацию можно найти в разделе «Формат портируемого исполняемого файла .NET» в главе 2.

языке. При условии, что ваш компонент соответствует правилам, изложенным в спецификации COM, можно создать его экземпляр и использовать его в своих приложениях. Поддерживая таким образом двоичное взаимодействие, технология COM не поддерживает *языковую интеграцию* (*language integration*). Другими словами, вы не можете использовать код COM-компонентов, написанных кем-то другим, не можете также расширить класс, заключенный в COM-компоненте, перехватывать исключения, вызываемые кодом COM-компонента, и т. д.

Microsoft .NET поддерживает не только языковую независимость, но и языковую интеграцию. Это означает, что невозможно наследовать от классов, перехватывать исключения и использовать преимущества полиморфизма между разными языками. .NET Framework делает это возможным с помощью системы общих типов – спецификации под названием Common Type System (CTS), которую должны поддерживать все .NET-компоненты. Например, все в .NET является объектом определенного класса, производного от корневого класса System.Object. CTS поддерживает общую концепцию классов, интерфейсов, делегатов (обеспечивающих поддержку обратных вызовов), ссылочных типов и типов значений. В базовых классах .NET представлено большинство базовых системных типов, таких как классы для поддержки целых значений, строк и манипуляций с файлами. Поскольку компилятор любого языка должен соответствовать минимальному набору правил, оговоренных в Common Language Specification (CLS), и генерировать код, согласующийся с CTS, различные языки в .NET могут общаться между собой. Мы рассмотрим CTS и CLS в главе 2.

Взаимодействие компонентов через Интернет

COM поддерживает распределенные вычисления через свой протокол взаимодействия Distributed COM (DCOM). Проблема с DCOM состоит в том, что он вставляет IP-адрес хоста в буфер Network Data Representation (NDR), таким образом, DCOM не работает через брандмауэры и программное обеспечение, осуществляющее трансляцию сетевых адресов (Network Address Translation, NAT). Кроме того, средства динамической активации, согласования протоколов и сборки мусора закрыты, сложны и дороги. Решением является открытый, простой и экономичный протокол распределенных вычислений. .NET Framework использует поддержанный компьютерной отраслью протокол SOAP, базирующийся на широко распространенных стандартах XML и HTTP.

Упрощение разработки

Те, кто разрабатывал программное обеспечение для Windows-платформ с момента их появления, видел все, начиная от Windows API и заканчивая Microsoft Foundation Classes (MFC), Active Template Library (ATL), системными COM-интерфейсами и несчетным числом обо-

лочек, таких как Visual InterDev, Visual Basic, JScript и других языков сценариев. Каждый раз тому, кто собирался разрабатывать что-нибудь при помощи другого компилятора, приходилось изучать новый API или библиотеку классов, поскольку между всеми этими библиотеками или интерфейсами не существовало никакого соответствия или унификации.

Решение .NET представляет собой набор базовых классов и позволяет работать с ними любому языку. Такая среда исключает необходимость в изучении нового API при каждом «переключении» между языками. Другими словами, наверняка проще разобраться с десятком методов определенного класса, чем изучать тысячу функций API.

Упрощение развертывания приложений

Представьте себе следующий сценарий: ваше Windows-приложение, использующее три разделяемые DLL-библиотеки, прекрасно работает многие месяцы, однако перестает работать на следующий день после того, как вы установили другой программный пакет, переписавший первую DLL, ничего не сделавший со второй DLL и записавший дополнительную копию третьей в другой каталог. Если вы хоть раз встречались с такой трудной (но при этом вполне реальной) проблемой, вы попали в *ад DLL* (*DLL Hell*). И если вы спросите в группе бывалых разработчиков, знаком ли им ад DLL, вы увидите у них гримасу досады, и не из-за заданного вами вопроса, а из-за того, что они действительно испытали боль и страдание.

Чтобы избежать DLL Hell (по крайней мере для системных DLL), Windows 2000 сохраняет системные DLL в кэше. Если вы устанавливаете приложение, которое переписывает системные DLL, Windows 2000 заменит новые системные DLL исходными версиями из кэша.

Microsoft .NET еще более уменьшает проблему DLL Hell. В среде .NET ваш исполняемый файл будет использовать ту общую DLL, с которой он был собран. Это гарантируется тем, что общая DLL регистрируется в чем-то вроде кэша Windows 2000, который называется глобальным кэшем сборок (Global Assembly Cache, GAC). Кроме выполнения этого требования общая DLL должна иметь уникальные хеш-код, открытый ключ, региональный код и номер версии. Если эти требования выполнены и ваша общая DLL зарегистрирована в GAC, ее физическое имя перестает иметь значение. Другими словами, если у вас есть две версии DLL, обе под именем *MyDll.dll*, то они могут располагаться и исполнятся на одной и той же системе, не вызывая проблем. Это становится возможным потому, что исполняемый файл, использующий одну из этих DLL, жестко привязывается к DLL при компиляции.

Кроме искоренения DLL Hell, .NET также исключает необходимость настройки реестра, относящейся к компонентам. Разработчик COM расскажет вам, что половина сложностей в изучении COM состоит в

необходимости знать специфичные для COM записи реестра, за которые отвечает программист. Microsoft .NET хранит все ссылки и зависимости сборок в специальной секции, называющейся *манифестом* (*manifest*). Кроме того, сборки могут быть как закрытыми, так и общими. Закрытые сборки находятся по логическим путям или конфигурационным XML-файлам, а общие сборки регистрируются в GAC; в обоих случаях система находит требуемые вам файлы на этапе выполнения приложения. Если файлы отсутствуют, вы получите исключение, сообщающее о том, что случилось.

И наконец, в .NET возвращается концепция установки и удаления приложений без каких-либо усилий. Эта концепция является противоположностью тому, с чем вам приходилось иметь дело в мире COM. Для установки COM-приложения вы должны были зарегистрировать все компоненты после копирования их на свой компьютер. Если вы выполнили этот этап неверно, ничего не будет работать, и вам останется только рвать на себе волосы. Аналогично, чтобы удалить приложение, вам придется аннулировать регистрацию ваших компонентов (удалить записи в реестре) перед тем, как удалить файлы. И опять же, если вы неправильно выполните этот этап, в реестре останутся следы, которые будут существовать вечно.

В отличие от COM, так же как в DOS, для установки приложения в .NET вам достаточно просто выполнить команду `xcopy`, чтобы скопировать файлы из одного каталога на CD в другой каталог на вашем компьютере, и приложение сможет запуститься автоматически.[1] Аналогично, для удаления приложения с вашего компьютера достаточно просто удалить каталог.

Надежность

В индустрии коммерческого программного обеспечения существует много языков программирования и платформ, однако в немногих из них делается попытка предоставить как надежный язык программирования, так и устойчивую библиотеку времени выполнения или инфраструктуру. Наиболее успешным языком, из тех, что мы видели среди коммерческого ПО, является язык Java™ и Java Virtual Machine™, которые были с радостью приняты сообществом разработчиков ПО. Microsoft позиционирует .NET как следующий большой шаг вперед.

В Microsoft .NET безопасность типов является обязательной. В отличие от C++, любой класс в .NET является производным от прародителя всех классов, Object, поддерживающего средства идентификации типов во время выполнения, средства построения дампа содержимого и прочие возможности. CLR должна распознать и проверить типы до

[1] Это верно для закрытых, но не для общих. Дополнительные детали рассмотрены в главе 4.

того, как они смогут быть загружены и выполнены. Это сокращает возможности возникновения элементарных программистских ошибок и предотвращает переполнение буферов, ослабляющее безопасность.

Традиционные языки программирования не предоставляют общего механизма обработки ошибок. C++ и Java поддерживают обработку исключений, однако многие другие языки о вас не заботятся, вынуждая изобретать собственные средства. Microsoft .NET поддерживает исключения на уровне CLR, предоставляя согласованный механизм обработки ошибок. Другими словами, исключения работают во всех языках, совместимых с .NET.

Программируя на C++, вы должны освобождать все расположенные в куче объекты, которые были ранее созданы. Если этого не сделать, выделенные вашей системой ресурсы никогда не будут возвращены обратно, даже если они больше не требуются. И если это серверное приложение, оно не будет надежным, поскольку накопленные в памяти неиспользуемые ресурсы со временем могут привести к краху системы. Аналогично, Java во время выполнения .NET отслеживает и выполняет сборку мусора для всех созданных в памяти объектов, которые более не требуются.

Безопасность

В старые добрые времена DOS разработчики Microsoft, создавая приложения, мало беспокоились о безопасности, так как приложения работали на одиночном настольном компьютере с одним потоком выполнения. Как только разработчики начали создавать клиентские и серверные приложения, все усложнилось: к серверам могли обращаться несколько пользователей, а клиент и сервер обменивались друг с другом важной информацией. Проблема еще более усилилась в среде Интернета, так как существует опасность незаметно загрузить и выполнить на своей машине злонамеренный апплет.

Чтобы смягчить эти проблемы, .NET предоставляет ряд средств безопасности. Windows NT и Windows 2000 защищают ресурсы с помощью списков управления доступом и идентификаторов безопасности, однако не предоставляют инфраструктуру безопасности для проверки прав доступа к отдельным частям исполняемого кода. В отличие от традиционной поддержки безопасности, при которой защищается только доступ к исполняемому файлу, .NET идет дальше и позволяет защитить доступ к определенным частям исполняемого кода. Например, чтобы воспользоваться преимуществами декларативной проверки безопасности, вы можете указать в реализации ваших методов атрибуты безопасности и не писать дополнительный код. Чтобы воспользоваться императивными проверками, в методе пишется код для явного осуществления проверки безопасности. .NET предоставляет много других функций безопасности, чтобы усложнить несанкционированное проникновение в ваше приложение и систему.

.NET Framework

Теперь, когда вы знакомы с основными задачами, поставленными перед .NET Framework, давайте кратко рассмотрим ее архитектуру. Как видно на рис. 1.2, .NET Framework располагается поверх операционной системы, которая может быть одной из различных видов Windows[1], и состоит из нескольких компонентов. (К более подробному обсуждению каждого из этих компонентов мы приступим с главы 4, как и сказано в предисловии.) .NET по существу является системным приложением, работающим в Windows.

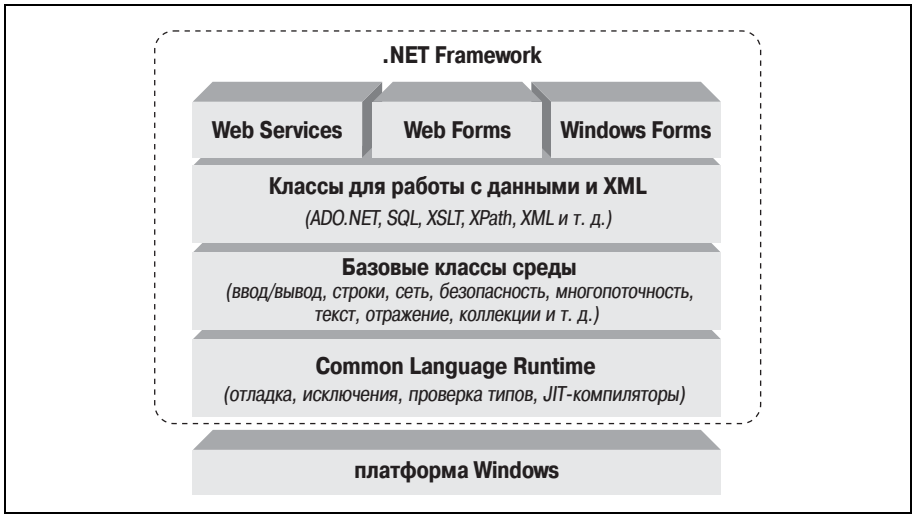

***Рис. 1.2.** .NET Framework*

Наиболее важным компонентом .NET Framework является нечто под названием Common Language Runtime (CLR), или общеязыковая среда выполнения. Если вы Java-программист, рассматривайте CLR как .NET-эквивалент виртуальной Java-машины (JVM). Если вы не знаете Java, считайте CLR сердцем и душой архитектуры .NET. На верхнем уровне CLR активизирует объекты, выполняет для них проверку безопасности, располагает их в памяти, исполняет их и выполняет сборку мусора.

Концептуально CLR и JVM схожи в том, что обе являются инфраструктурами времени выполнения, абстрагирующими различия в платформах, на которых они работают. Однако если JVM в данный момент поддерживает только язык Java, то CLR поддерживает все язы-

[1] В действительности операционной системой может быть (потенциально) любой вид Unix или иной операционной системы. Это возможно вследствие архитектуры CLR, которая обсуждается в главе 2.

ки, которые представимы на общем промежуточном языке Common Intermediate Language (CIL). JVM исполняет байт-код, поэтому технически также может поддерживать много различных языков. Однако, в отличие от байт-кода Java, IL никогда не интерпретируется. Другим концептуальным отличием между этими двумя инфраструктурами является то, что код Java с помощью JVM может работать на многих платформах, тогда как код .NET работает только на платформе Windows с CLR (на момент написания этой книги). Microsoft представила общеязыковую инфраструктуру Common Language Infrastructure (CLI), являющуюся подмножеством CLR, в ECMA, так что сторонние поставщики теоретически смогут реализовать CLR для платформы, отличной от Windows. Дополнительную информацию о сторонних поставщиках см. в приложении A.

На рис. 1.2 уровнем, расположенным над CLR, является набор базовых классов среды. Этот набор похож на наборы классов в STL, MFC, ATL или Java. Классы поддерживают элементарную функциональность ввода/вывода, манипуляции со строками, управление безопасностью, функциональность отражений и коллекций, а также другие функции.

Над базовыми классами среды находится набор классов, расширяющих базовые поддержкой управления данными и XML. Классы работы с данными поддерживают работу с постоянными данными – информацией, хранящейся в базах данных. Эти классы включают классы работы с SQL (Structured Query Language), позволяя вам манипулировать хранилищами данных через стандартный SQL-интерфейс. Аналогично SQL-классам, набор классов под названием ADO.NET позволяет вам манипулировать постоянными данными. Наряду с классами работы с данными, .NET Framework поддерживает несколько классов, позволяющих вам манипулировать XML-данными, выполнять XML-поиск и XML-преобразования.

Классы в трех различных технологиях (Web Services, Web Forms и Windows Forms) расширяют базовые классы и классы работы с данными и XML. Web Services включают ряд классов, поддерживающих разработку облегченных распределенных компонентов, которые будут работать даже через брандмауэры и программное обеспечение NAT. Эти компоненты поддерживают plug-and-play через киберпространство, так как веб-сервисы применяют стандартные протоколы HTTP и SOAP.

Web Forms включает набор классов, позволяющих вам быстро разрабатывать веб-приложения с графическим интерфейсом пользователя (GUI). Если сейчас вы разрабатываете веб-приложения с помощью Visual Interdev, можете рассматривать Web Forms как средство, позволяющее использовать для создания графических веб-интерфейсов подход «drag-n-drop», аналогичный используемому для разработки пользовательских интерфейсов в Visual Basic. Просто поместите управляющие элементы на вашу веб-форму, два раза щелкните мышью на элементе управления и пишите код реакции на соответствующее событие.

Windows Forms поддерживают набор классов, позволяющий вам разрабатывать обычные графические Windows-приложения. Можно рассматривать все эти классы как улучшенную версию MFC, т. к. они поддерживают более простую разработку графических интерфейсов и предоставляют общий, совместимый интерфейс, который можно использовать во многих языках.

В следующей главе мы рассмотрим внутреннее устройство CLR и то, как в ней поддерживаются и исполняются .NET-компоненты, официально называемые в .NET *сборками* (*assemblies*).

2

Common Language Runtime

Наиболее важный компонент .NET Framework – это общеязыковая среда выполнения Common Language Runtime (CLR). CLR управляет кодом и исполняет код, написанный на языках .NET, и является фундаментом всей архитектуры .NET, подобно виртуальной машине Java. CLR активизирует объекты, выполняет для них проверку безопасности, располагает в памяти, исполняет их и выполняет сборку мусора.

В этой главе мы приведем описание среды CLR, исполняемых файлов (с примерами на нескольких языках), метаданных, сборок, манифестов, CTS и CLS.

Среда CLR

CLR – это инфраструктура, лежащая в основе .NET. Ее возможности охватывают все задачи, о которых мы говорили в главе 1. В отличие от таких библиотек программного обеспечения, как MFC или ATL, CLR построена с чистого листа. CLR управляет исполнением кода в .NET Framework.

Сборка (*assembly*) – это основная единица для развертывания и контроля версий, состоящая из манифеста, набора из одного или более модулей и необязательного набора ресурсов.

На рис. 2.1 показаны две части среды .NET: нижняя – это CLR, а верхняя – исполняемые файлы CLR или переносимые исполняемые файлы (Portable Executable, PE), являющиеся сборками .NET или единицами развертывания. CLR – это механизм времени выполнения, загружающий требуемые классы, выполняющий оперативную (just-in-time)

компиляцию необходимых методов, проверку безопасности и много других функций времени выполнения. Исполняемые файлы CLR, показанные на рис. 2.1, являются EXE- или DLL-файлами, состоящими в основном из метаданных и кода.

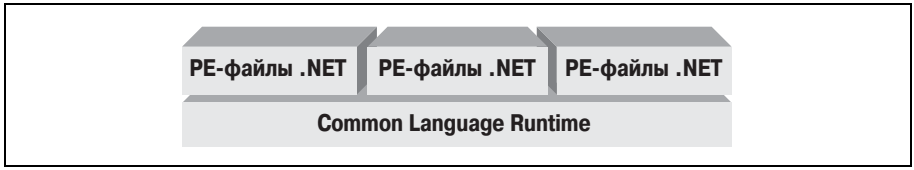

Рис. 2.1. *Среда CLR*

Исполняемые файлы CLR

Исполняемые файлы Microsoft .NET отличаются от типичных исполняемых файлов Windows тем, что они содержат не только код и данные, но еще и метаданные (см. далее в этой главе разделы «Метаданные» и «Промежуточный язык IL»). В данном разделе мы начнем с кода нескольких .NET-приложений и обсудим PE-формат, используемый в .NET.

Hello, World: Managed C++

Начнем с простого приложения *Hello, World*, написанного на Managed C++ – расширении языка C++, используемом в Microsoft .NET. Managed C++ содержит несколько новых ключевых слов, специфических для .NET, которые позволяют программам, написанным на C++, воспользоваться новыми возможностями .NET, включая сборку мусора. Вот версия нашей программы, написанная на Managed C++:

```
#using <mscorlib.dll>
using namespace System;

void main()
{
  Console::WriteLine(L"C++ Hello, World!");
}
```

Как видите, это обычная программа на C++ с дополнительной директивой `#using` (выделенной жирным шрифтом). Если вы работали с компилятором Microsoft Visual C++, поддерживающим средства для технологии COM, вам может быть знакома директива `#import`. Если `#import` конвертирует информацию о типах для генерации классов-оболочек для COM-интерфейсов, то директива `#using` делает доступными все типы из указанной DLL, аналогично директиве `#include` в C или C++. Однако в отличие от `#include`, импортирующей типы C или C++, `#using` импортирует типы любой сборки .NET, написанной на любом языке .NET.

Первый и единственный оператор в методе main() не требует объяснений – он обозначает вызов статического метода (или метода уровня класса) WriteLine(), класса Console. Символ `L`, предшествующий строке символов, сообщает компилятору C++ о необходимости преобразовать строку символов в строку Unicode. Как вы могли догадаться, класс Console – это тип, расположенный в *mscorlib.dll*, и его метод принимает один строковый параметр.

Вы также можете заметить, что код сообщает компилятору об использовании типов из пространства имен System, на что указывает оператор `using namespace`. Это позволяет нам применять идентификатор Console вместо полного обозначения класса System::Console.

Скомпилируем эту простую программу с помощью нового компилятора C++ с интерфейсом командной строки, поставляемого с .NET SDK:

```
cl hello.cpp /CLR /link /entry:main
```

Параметр командной строки `/CLR` исключительно важен, т. к. он сообщает компилятору C++ о необходимости сгенерировать PE-файл для среды .NET, а не обыкновенный PE-файл Windows.

После выполнения этой команды компилятор C++ сгенерирует исполняемый файл *hello.exe*. Когда вы запустите *hello.exe*, CLR загрузит, проверит и исполнит его.

Hello, World: C#

Поскольку .NET серьезно относится к языковой интеграции, мы проиллюстрируем ту же программу, используя новый язык C#, разработанный в Microsoft специально для .NET. Заимствовавший синтаксис у Java и C++, C# – это простой и объектно-ориентированный язык, который Microsoft использовала для написания большей части базовых классов и утилит .NET. У тех, кто программирует на Java (или C++), не будет проблем с пониманием кода C#. Вот *Hello, World* на C#:

```
using System;

class MainApp
{
  public static void Main()
  {
    Console.WriteLine("C# Hello, World!");
  }
}
```

C# схож с Java в том, что в нем нет концепции заголовочного файла: определения классов и их реализации хранятся в одном файле *.cs*. Другое сходство с Java состоит в том, что Main() – это открытая статическая функция определенного класса, что видно из кода. Этим он отличается от C++, в котором метод main() является глобальной функцией.

Ключевое слово `using` работает здесь аналогично `using namespace` в предыдущем примере, сообщая компилятору C#, что мы хотим использовать типы из пространства имен System. Вот как компилируется эта программа C#:

```
csc hello.cs
```

В приведенной выше команде `csc` – это имя компилятора C#, поставляемого в составе .NET SDK. Результатом выполнения этой команды также будет исполняемый файл *hello.exe*, который можно запустить как обычный EXE, однако он работает под управлением CLR.

Hello, World: VB.NET

И наконец, для полноты картины, вот та же самая программа на языке Visual Basic.NET (VB.NET):

```
Imports System

Public Module modmain
  Sub Main()
    Console.WriteLine ("VB Hello, World!")
  End Sub
End Module
```

Если вы VB-программист, вас ждет сюрприз. Синтаксис языка немного изменился, однако, к счастью, в этих изменениях воспроизведены возможности других объектно-ориентированных языков, таких как C# и C++. Посмотрите внимательно на этот фрагмент кода, и вы увидите, что каждая его строка может быть транслирована в эквивалентную строку C#. В то время как в C# применяются ключевые слова `using` и `class`, в VB.NET аналогичную роль играют, соответственно, ключевые слова `Import` и `Module`. Вот как нужно компилировать эту программу:

```
vbc /t:exe /out:Hello.exe Hello.vb
```

Microsoft также предоставляет для VB.NET компилятор с интерфейсом командной строки, *vbc*. Параметр `/t` указывает на тип создаваемого PE-файла. В этом случае, поскольку мы указали имя EXE-файла, результатом выполнения этой команды будет файл *hello.exe*.

Во всех трех версиях этой программы *Hello, World*, класс Console и метод WriteLine() оставались неизменными. Таким образом, не имеет значения, на каком языке вы программируете, и если вы знаете, как сделать что-либо с помощью одного языка, то вы знаете, как это делать и в другом. Это большое изменение по сравнению с традиционным программированием в Windows, в котором, если вы знаете, как осуществляется запись в файл на C++, вы не обязательно знаете, как это сделать в VB, Java или Cobol.

Переносимый исполняемый файл .NET

Исполняемый файл Windows, EXE или DLL, должен соответствовать формату, который называется форматом *PE* и является производным от формата Microsoft Common Object File Format (COFF). Для обоих этих форматов существуют и свободно доступны полные спецификации. ОС Windows знает, как загружать и исполнять DLL- и EXE-файлы, т. к. понимает формат PE-файла. Поэтому любой компилятор должен генерировать исполняемый файл Windows в соответствии со спецификацией PE/COFF.

Стандартные PE-файлы Windows делятся на две основных секции. Первая включает в себя заголовки PE/COFF со ссылками на содержимое внутри PE-файла. Кроме секции заголовков в PE-файле имеется несколько секций двоичных образов, включающих секции `.data`, `.rdata`, `.rsrc` и `.text`. Это стандартные секции типичного исполняемого файла Windows, однако компилятор Microsoft C и C++ позволяет добавлять в PE-файл собственные секции с помощью директивы компилятора `pragma`. Например, можно создать собственные секции, содержащие зашифрованные данные, которые сможете читать только вы сами. Пользуясь этой возможностью, Microsoft добавила несколько новых секций в обычный PE-файл специально для поддержки функциональности CLR. Среда CLR понимает и управляет новыми секциями. Например, она читает эти секции и определяет, каким образом загружать классы и выполнять ваш код.

Как показано на рис. 2.2, разделы, добавленные Microsoft к обычному формату PE, – это заголовок CLR и раздел данных CLR. Заголовок CLR содержит информацию, указывающую, что PE-файл является исполня-

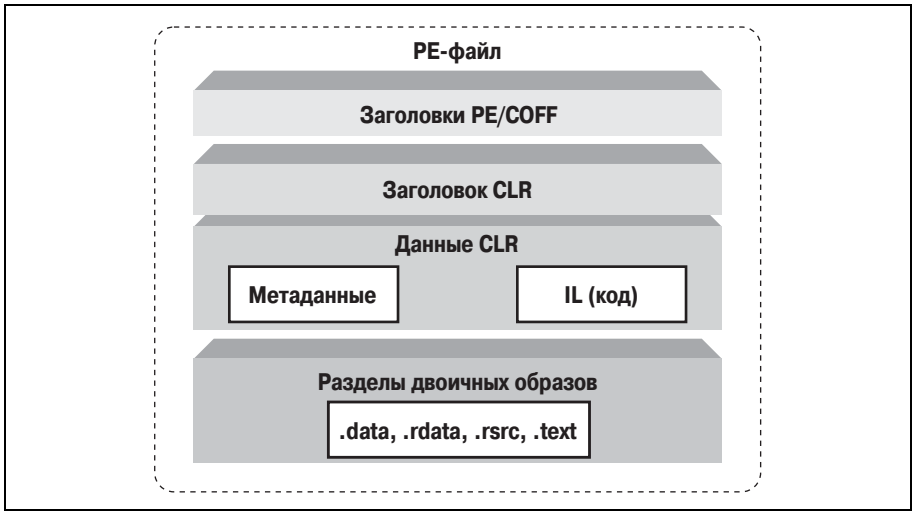

Рис. 2.2. *Формат PE-файла .NET*

емым файлом .NET, а раздел данных CLR содержит метаданные и IL-код, которые вместе определяют, как будет выполняться программа.

Если вы хотите убедиться, что исполняемый файл .NET содержит обе эти секции, воспользуйтесь утилитой *dumpbin.exe*, которая показывает содержимое исполняемого файла Windows в читабельном текстовом виде. Например, выполнив в командной строке команду:

```
dumpbin.exe hello.exe /all
```

вы получите следующие данные (для краткости мы показали лишь основные элементы, которые хотели проиллюстрировать):

```
Microsoft (R) COFF/PE Dumper Version 7.00.9344.1
Copyright (C) Microsoft Corporation.  All rights reserved.

Dump of file hello.exe

PE signature found

File Type: EXECUTABLE IMAGE

FILE HEADER VALUES  [MS-DOS/COFF HEADERS]
  14C machine (x86)
   3 number of sections
   . . .

OPTIONAL HEADER VALUES  [PE HEADER]
  10B magic # (PE32)
  . . .

SECTION HEADER #1  [SECTION DATA]
  . . .
  Code
  Execute Read

RAW DATA #1
  . . .

  clr Header:
    . . .

  Section contains the following imports:
    mscoree.dll
      402000 Import Address Table
      4022B8 Import Name Table
        . . .

          0 _CorExeMain
```

Взглянув на этот текстовый дамп PE-файла .NET, можно увидеть, что PE-файл начинается с заголовков MS-DOS и COFF, которые должны содержаться в любой программе. За этими заголовками следует PE-заголовок для поддержки 32-разрядных программ Windows. Сразу за PE-заголовком находится первый раздел данных исполняемого файла. В PE-файле среды .NET это раздел (SECTION HEADER #1, как пока-

зано выше), в котором хранится заголовок и данные CLR. Заметьте, что он помечен как `Core` и `Execute Read`, благодаря чему загрузчик ОС и CLR знают, что эта секция содержит код, который должен исполняться CLR во время запуска программы.

Можно заметить, что в CLR-заголовке импортируется функция с именем `_CorExeMain`, реализованная в *mscoree.dll*, ядре механизма исполнения CLR.[1] На время написания этих строк Windows 98, 2000 и Me имели загрузчик, который знал, как загружать только стандартные PE-файлы. Чтобы избежать внесения значительных изменений в эти операционные системы и при этом обеспечить в них запуск .NET-приложений, Microsoft обновила загрузчики для всех этих платформ. Новые загрузчики знают, как проверить наличие заголовка CLR и, если такой заголовок существует, исполняют `_CorExeMain`, таким образом, не только передавая управление CLR, но и отдаваясь в ее руки. Как вы можете догадаться, ваша функция Main() будет в конечном итоге вызвана из CLR.

Рассмотрев содержимое заголовка CLR, перейдем к содержимому данных CLR, включающих метаданные и код, которые, бесспорно, являются наиболее важными элементами в .NET.

Метаданные

Метаданные (*metadata*) – это данные о ресурсах, или «данные о данных», предназначенные для чтения машиной. Это могут быть подробные сведения о содержимом, формате, размере или других характеристиках источника данных. В .NET метаданные включают определения типов, сведения о версии, ссылки на внешние сборки и другую стандартизованную информацию.

Для того чтобы две системы, два компонента или два объекта могли взаимодействовать друг с другом, по крайней мере один из них должен что-то знать о другом. В COM это «что-то» – спецификация интерфейса, реализуемая поставщиком компонента и используемая его клиентами. Спецификация интерфейса содержит прототипы методов с полными сигнатурами, включая определения типов для всех параметров и возвращаемых значений.

Только разработчики C/C++ могли легко модифицировать и использовать определения типов на языке Interface Definition Language (IDL, язык описания интерфейса), что было недоступно для разработчиков

[1] Мы советуем вам запустить *dumpbin.exe* и просмотреть данные экспорта *mscoree.dll*. Вы обнаружите также `_CorDllMain`, `_CorExeMain`, `_CorImageUnloading` и другие интересные экспортируемые функции. Интересно отметить, что эта DLL представляет собою внутрипроцессный COM-сервер, свидетельствуя о том, что .NET создана с привлечением технологии COM.

на VB и других языках и, что более важно, для утилит и связующего программного обеспечения (middleware). Поэтому в Microsoft придумали нечто отличное от IDL, что мог бы использовать каждый, под названием *библиотека типов* (*type library*). В COM библиотеки типов позволяют утилите или среде разработки читать, разбирать и создавать классы-оболочки, наиболее подходящие и удобные для данного разработчика. Библиотеки типов также позволяют механизмам времени выполнения, таким как библиотеки времени выполнения VB, COM, MTS или COM+, определять типы во время выполнения приложений и предоставлять необходимые заплатки или промежуточную поддержку для их использования приложениями. Например, библиотеки типов поддерживают динамические вызовы и позволяют исполняющей системе COM предоставлять универсальный маршалинг[1] для межконтекстных вызовов.

Библиотеки типов в COM исключительно богаты возможностями, однако многие разработчики критикуют их за отсутствие стандартизации. Команда .NET создала новый механизм получения информации о типах. Вместо термина «библиотека типов» в .NET для названия такой информации используется термин «метаданные».

Библиотеки типов «на стероидах»

Если считать, что библиотеки типов – это заголовочные файлы C++ «на стероидах», то метаданные – это библиотеки типов «на стероидах». В .NET метаданные – это общий механизм, применяемый и средой времени выполнения .NET, и компиляторами, и утилитами, или «диалект», на котором они «общаются». В Microsoft .NET метаданные служат для описания всех типов, используемых и предоставляемых данной сборкой .NET. В этом смысле метаданные описывают сборку в деталях, включая ее идентификатор (комбинация имени сборки, версии, культуры и открытого ключа), типы, на которые она ссылается, экспортируемые типы и требования безопасности для исполнения. Предоставляя намного больше информации, чем библиотеки типов, метаданные включают описания сборки и модулей, классов, интерфейсов, методов, свойств, полей, событий, глобальных методов и т. д.

Метаданные предоставляют достаточно информации для любой среды выполнения, утилиты или программы, чтобы выяснить буквально все, что требуется для интеграции компонентов. Давайте посмотрим на краткий список компонентов, использующих метаданные в среде .NET, просто чтобы доказать, что метаданные действительно подобны библиотекам типов «на стероидах».

---

[1] В COM *универсальный маршалинг* (*universal marshaling*) – это обычный способ преобразования всех типов данных. Универсальный маршалер может использоваться для преобразования любых типов данных, поэтому вам не требуется писать собственный код посредников и заглушек.

CLR

CLR использует метаданные для верификации, обеспечения безопасности, межконтекстного маршалинга, распределения памяти и исполнения. CLR активно полагается на метаданные для поддержки этих средств времени выполнения, которые мы обсудим в свое время.

Загрузчик классов

Компонент CLR, загрузчик классов, использует метаданные для обнаружения и загрузки классов .NET. Это становится возможным потому, что в метаданных содержится подробная информация по конкретному классу, – о том, где этот класс находится, в той же ли он сборке, внутри или вне определенного пространства имен или в подчиненной сборке где-то в сети.

JIT-компиляторы

JIT-компиляторы применяют метаданные для компиляции кода Microsoft Intermediate Language (IL). IL – это промежуточное представление, вносящее значительный вклад в поддержку языковой интеграции, но не являющееся ни кодом VB, ни байт-кодом, который должен интерпретироваться. В .NET JIT компилирует IL перед исполнением в двоичный код и делает это с использованием метаданных.

Инструментальные средства

Инструментальным средствам метаданные нужны для поддержки интеграции. Например, инструментальные средства разработчика могут, основываясь на метаданных, генерировать процедуры-оболочки, обеспечивающие взаимодействие компонентов .NET и COM. Такие средства, как отладчики, профайлеры, браузеры объектов, могут при помощи метаданных обеспечивать более обширную поддержку разработки. Один из примеров – средства IntelliSense, поддерживаемые в Microsoft Visual Studio.NET. Как только вы набрали имя объекта и точку, данное средство выводит список методов и свойств, из которого можно выбрать требуемое. Благодаря этому не приходится искать имена методов или свойств и синтаксис их вызова в заголовочных файлах или документации.

Как и CLR, любое приложение, инструментальное средство или утилита, способные читать метаданные из сборки .NET, могут использовать эту сборку. Классы-отражения в Microsoft .NET Framework помогут исследовать PE-файл .NET и узнать все о типах данных, которые используются и предоставляются сборкой. В CLR применяется тот же набор классов-отражений для просмотра и предоставления функций времени выполнения, включая управление памятью, средства безопасности, проверки типов, отладки, удаленного управления и т. д.

Метаданные гарантируют возможность межъязыкового взаимодействия, основного элемента .NET, т. к. все языки должны использовать