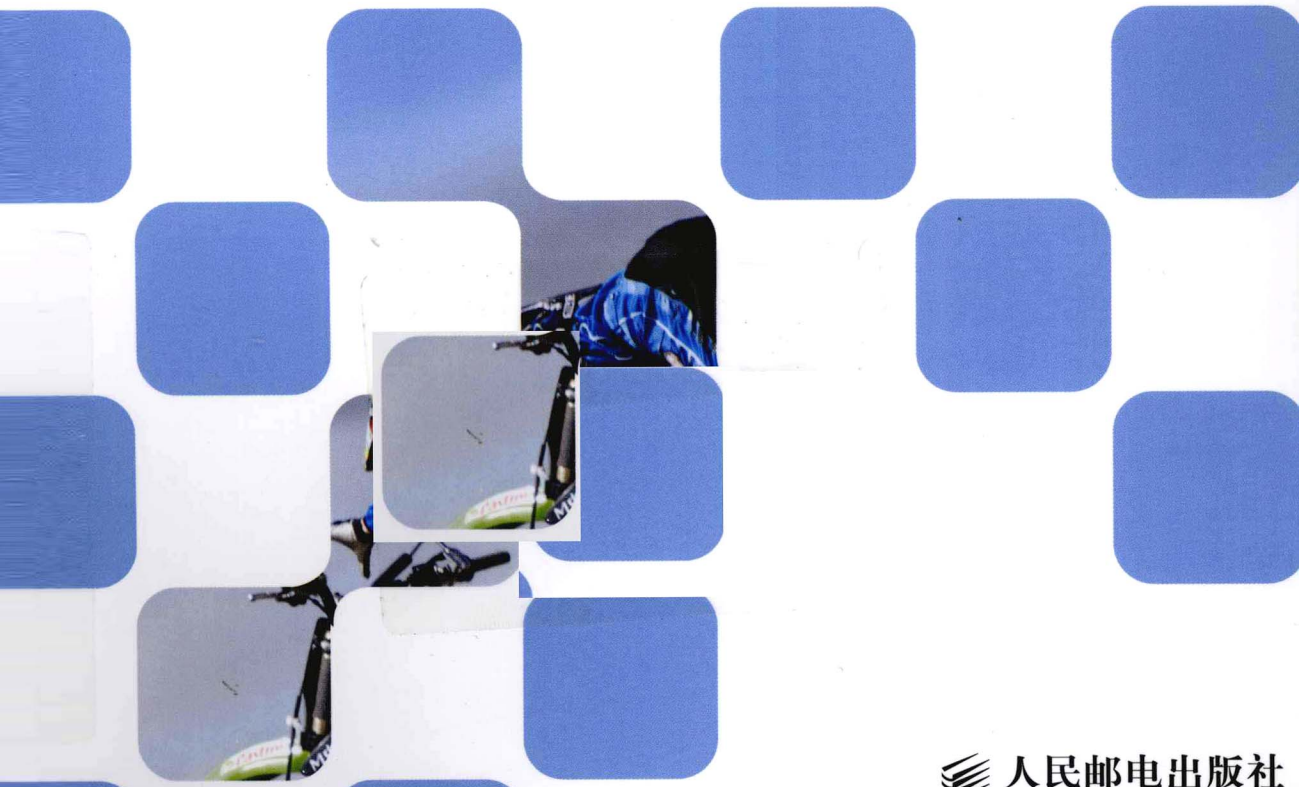
TURING 图灵程序设计丛书

Addison Wesley

Windows 7 Device Driver

Windows 7 设备驱动程序开发

[美] Ronald D. Reeves 著
张猛 纪小玲 周姝嫣 译



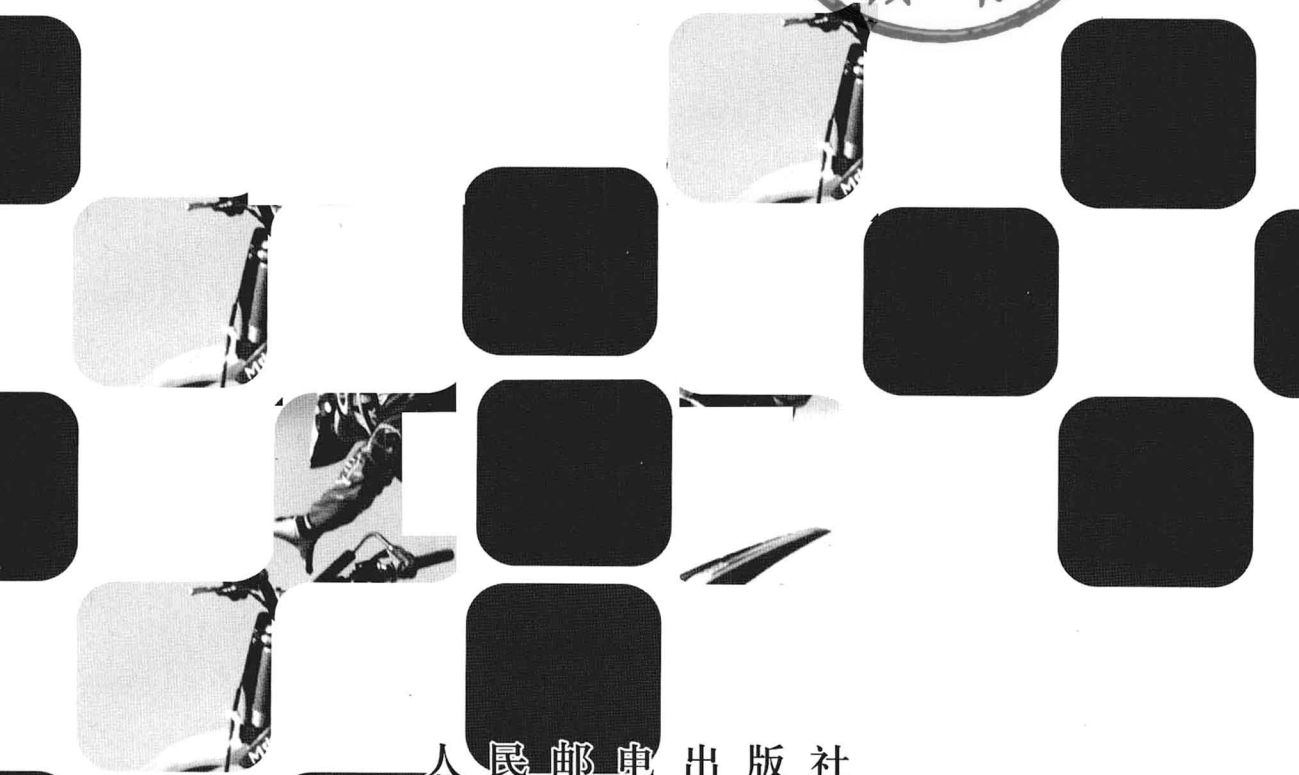
人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

Windows 7 Device Driver

Windows 7 设备驱动程序开发

[美] Ronald D. Reeves 著
张猛 纪小玲 周姝嫣 译



人民邮电出版社
北京

图书在版编目（CIP）数据

Windows 7设备驱动程序开发 / (美) 里夫斯(Reeves, R. D.) 著 ; 张猛, 纪小玲, 周姝嫣译. -- 北京 : 人民邮电出版社, 2012.1
(图灵程序设计丛书)
书名原文: Windows 7 Device Driver
ISBN 978-7-115-26579-1

Ⅰ. ①W… Ⅱ. ①里… ②张… ③纪… ④周… Ⅲ. ①Windows操作系统－设备驱动程序 Ⅳ. ①TP316.7

中国版本图书馆CIP数据核字(2011)第209828号

内 容 提 要

本书介绍了编写全新 Windows 7 操作系统的设备驱动程序所需的全部技术。本书共包括 3 个部分：设备驱动程序体系结构概述、用户模式驱动程序和内核模式驱动程序。第一部分介绍设备处理程序软硬件开发所涉及的体系结构、UMDF 和 KMDF，以及开发 UMDF 和 KMDF 驱动程序所需的环境。第二部分介绍用户模式驱动程序的方法、设计、开发和调试。主要介绍使用 UMDF 以及 C++ 进行用户模式驱动程序开发。第三部分介绍内核模式驱动程序的方法、设计、开发和调试。主要介绍使用 KMDF 以及 C 语言开发内核模式驱动程序。

本书适合 Windows 7 驱动程序开发人员阅读。

图灵程序设计丛书

Windows 7设备驱动程序开发

◆ 著 [美] Ronald D. Reeves
译 张 猛 纪小玲 周姝嫣
责任编辑 王军花
执行编辑 李 静

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：14.75
字数：346千字 2012年1月第1版
印数：1－3 000册 2012年1月北京第1次印刷
著作权合同登记号 图字：01-2011-5239号
ISBN 978-7-115-26579-1

定价：45.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223
反盗版热线：(010)67171154

前　言

本书提供了编写全新Windows 7操作系统的设备驱动程序所需的技术指南和说明。本书介绍了这项非常复杂的编程开发工作，以及Windows驱动程序框架对这项工作的巨大简化。书中阐述了驱动程序开发人员必须了解的硬件和软件体系结构。但本书的重点是围绕开发两类驱动程序所必须采用的实际开发步骤展开的。所以，本书采用注重实效的方法，根据实际的设备处理程序开发过程，来介绍各种软件API以及计算机硬件和设备硬件。

创建和调试设备驱动程序这门艺术目前已经有了很大的进展。现在已有很多与驱动程序框架有关的面向对象设计技术，可供设备驱动程序开发人员使用。过去开发时要做的许多脏活累活，现在已经由最新推出的设备开发框架WDF（Windows Driver Foundation，Windows驱动程序基础）代为处理了。本书既介绍了用户模式设备驱动程序的开发，还介绍了内核模式设备驱动程序的开发。WDF拥有优秀的子模型，称为用户模式驱动程序框架（UMDF）和内核模式驱动程序框架（KMDF）。

Windows设备驱动程序的创建过程中用到了Windows驱动程序框架，这真是件令人高兴的事。我从1990年就开始从事Windows开发，主要使用Win32系统API与Windows操作系统通信以及控制Windows操作系统。我们用DDK（Device Driver Kit，设备驱动程序开发包），创建Windows驱动程序。因为有自己的应用软件公司，所以我非常关心开发应用软件需要的时间以及应用程序的健壮性。开发驱动程序需要使用2000多个Win32 API。

大约在1992年，微软公司推出了MFC（Microsoft框架类）。在这600多个类中封装了大多数Win32 API。当然，在此之前，大约在1988年就已经推出了C++编译器，而且面向对象编程开始成型。使用MFC框架可以更快地开发出更多软件，同时质量更好。我的投资回报率（ROI）增加了，因此赚了更多的钱。这让我极力推崇框架的使用。我一直使用MFC，直到.NET Framework出现，在过去9年中，我一直在使用这个优秀的类集合。与此同时，微软公司一直努力将同样的软件开发改进也带到设备驱动程序开发工作中来。我们从DDK发展到Windows驱动程序模型，再到Windows驱动程序基础框架。

所以，本书阐明了如何使用Windows驱动程序基础框架创建Windows 7设备驱动程序。它可以让驱动程序开发人员在完成任务时更加轻松。

本书分为如下三部分。

- 第一部分“设备驱动程序体系结构概述”。该部分介绍设备处理程序软硬件开发涉及的体系结构，还介绍通常要开发的两类驱动程序（UMD和KMD）所需的开发环境。这一部分还介绍了当前驱动程序设备开发最常用的两个Windows驱动程序框架：UMDF和KMDF。

- 第二部分“用户模式驱动程序”。该部分介绍用户模式驱动程序的方法、设计、开发和调试。该部分将引导驱动程序开发人员从头开始完成用户模式驱动程序的开发，主要使用UMDF完成开发工作。代码是用C++编写的，因为C++是开发此类驱动程序的最佳方式。其中将围绕用UMDF开发的一个USB用户模式驱动程序展开讨论。我们使用来自OSR（Open Systems Resources）公司的USB硬件学习工具包。这为我们提供了测试用户模式驱动程序所需的硬件模拟。这一部分内容自成体系，与本书其他部分的知识无联系。但阅读第一部分将对本部分内容的学习有所帮助。
- 第三部分“内核模式驱动程序”。该部分介绍内核模式驱动程序的方法、设计、开发和调试。该部分同样从头开始全面介绍了内核模式驱动程序的开发，主要使用KMDF完成开发工作。代码用C语言编写，因为这是开发此类驱动程序的最佳方式。其中将围绕用KMDF开发的一个内核模式驱动程序展开讨论。我们使用来自OSR公司的一个PCI（外部组件互联）硬件学习工具包。这为我们提供了测试内核模式驱动程序所需的硬件模拟。这一部分内容自成体系，与本书其他部分的知识无联系。但阅读第一部分将对本部分内容的学习有所帮助。

致谢

首先感谢Pearson Education出版社的编辑Bernard Goodwin，给了我编写本书的机会。他在本书的准备阶段给予了我大量的支持。还要感谢他的助理Michelle Housley及时给我提供参考图书和参考资料。另外还要感谢Pearson Education出版社的视频项目经理John Herrin在为本书创建视频方面提供的支持和帮助。感谢开发编辑Michael Thurston对本书的润色。

引　言

设备驱动程序是非常特殊的软件部分，就相当于轮胎和路面接触的地方，应用程序通过它才能与外界沟通。Windows 7与外界的所有沟通都必须有设备驱动程序的参与。这些设备包括鼠标、显示器、键盘、光驱、数据采集、数据网络通信、打印机等。微软公司编写并随Windows 7操作系统一同提供了很多驱动程序。这些驱动程序支持绝大多数标准设备，本书不打算介绍这些驱动程序。

本书介绍的是如何为非标准设备创建设备驱动程序，即那些标准PC上通常找不到的设备。很多时候，由于市场太小，微软公司没有为此类设备创建标准设备驱动程序，例如数据采集板、实验设备、特殊的测试设备以及通信板等。

这类讨论主要介绍设备驱动程序开发人员感兴趣的重要功能。图0-1显示了Windows 7的一般结构图。在本书的各个部分中，我们还会在此基础上绘制更详细的示意图。

在图0-1中，用户应用程序不直接调用Windows 7操作系统服务。它们通过Win32子系统的动态链接库（DLL）运行。用户模式设备驱动程序（后面讨论）也使用相同的通信通道。

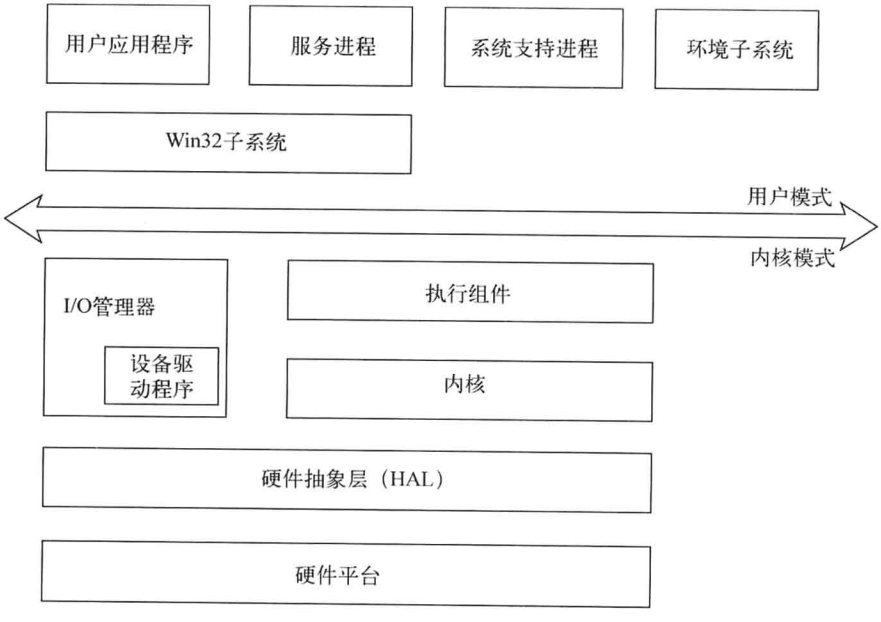


图0-1　Windows 7系统概述

各种独立运行的Windows 7服务由服务进程处理。它们通常由服务控制管理器启动。

各种Windows 7系统支持进程都不被当做Windows 7服务，所以不由服务控制管理器启动。

Windows 7 I/O管理器实际上由多个执行子系统组成，负责管理硬件设备、系统和应用程序的优先级接口。我们将在第二部分和第三部分详细介绍Windows 7 I/O管理器。

I/O管理器部分中的设备驱动程序部分是本书的重点，即设计、开发、测试Windows 7设备驱动程序。驱动程序将用户的I/O函数调用转变成硬件设备的I/O请求。

HAL是将特定平台的硬件差异与Windows 7操作系统隔离开的一个代码层。这就允许Windows 7操作系统在不同的硬件主板上运行。一般来说，将设备驱动程序代码移植到新平台时，只需进行重新编译。设备驱动程序代码依赖HAL中的代码（宏）来引用硬件总线和寄存器。通常使用HAL是为了实现内联的性能。

Windows 7的性能目标通常会给设备驱动程序作者带来很大影响。系统线程和用户向设备请求服务时，重要的是驱动程序代码不要阻塞执行。在这种情况下，驱动程序请求不能被立即处理，请求必须加入队列，等待后续处理。在后面的内容中将会看到，I/O管理器例程支持这种做法。

Windows 7为应用程序提供了可以利用的丰富的体系结构。但是，设备驱动程序的作者不得不为这种丰富付出代价。微软公司早在大约14年前就认识到了这点，因此开始开发驱动程序开发模型和框架，为设备驱动程序作者提供帮助。最早的Windows驱动程序模型（WDM）很难掌握，但这是一大进步。微软公司接下来又开发了WDF，它使得开发健壮的Windows 7驱动程序变得更容易实现和学习。本书将介绍如何使用WDF开发Windows 7设备驱动程序。

目　录

Part 1 第一部分

设备驱动程序体系结构概述

本部分内容

- 第1章 对象
- 第2章 WDF体系结构

第1章 对 象

在开始讨论驱动程序之前，我们需要简要地回顾一下对象，因为这是贯穿本书始末的概念。

1.1 对象的本质

基于组件的软件工程的一个基本概念是对象的使用。对象到底是什么呢？关于对象是什么，似乎还没有一个能被普遍接受的概念。计算机科学家Grady Booch（1991）的看法是，一个对象主要由3种特征定义：状态、行为和标识。在大多数认知理论中，分析的基本单位是信息处理组件。组件用于完成最基本的信息处理，这种处理针对的是对象或符号的内部表示（Newell和Simon，1972；Sternberg，1977）。如果我们了解一下这些组件的工作方式，就会发现它们可以将感知输入转换成概念表示，将一种概念表示转变成另一种概念表示，或者将一种概念表示转换成电机输出。

软件的面向对象编程（OOP）技术已经诞生差不多25年了。但它并不是新生事物。古代哲学家（如柏拉图和亚里士多德）及现代哲学家（如伊曼努尔·康德）都已解释了存在的一般意义，并确定了概念和对象的本质特征（Rand，1990）。最近，Minsky提出了一种对象理论，即对象的行为非常类似于人脑中发生的处理（Minsky，1986）。Novak和Gowin展示了对象在教育和认知科学中如何发挥重要作用（Novak和Gowin，1984）。在他们的方法中，通过在由某个名称指定的对象中寻找模式而发现了概念。不过请等一下，我们准备谈论对象，而现在却在谈论概念。这是因为，概念反映了将世界分类的方式，而我们学习、沟通和思考的大多数内容都涉及这些类之间的关系。概念是类的思维表示，它们的主要功能是提高认知效率。而且，类还可以看做是一个模板，可用于生成具有类似结构和行为的对象。

OMG（Object Management Group，对象管理组）将类定义如下。

类是一种可以被初始化以创建具有相同行为的多个对象的实现。对象是类的一个实例。

从软件的角度来看，通过将软件分类，我们将减少必须感知、学习、记忆、沟通和推理的信息量。

1.2 软件对象是什么

软件对象是什么？1976年，Niklaus Wirth出版了他的著作*Algorithms + Data Structures =*

Programs（《算法+数据结构＝程序》）。书中提到的算法和数据结构两方面的关系，加强了我们对程序主要构成部分的认识。1986年，J. Craig Cleaveland出版了*Data Types*（《数据类型》）一书。1979年，Bjarne Stroustrup已经在开发带类的C。到1985年，这就演化成了C++编程语言。1990年，Bjarne Stronustrup出版了*The Annotated C++ Reference Manual*一书。本书将只讨论与对象相关的.NET Framework基类和.NET Framework类库，因为这是我们现在关注的重点。

当Bjarne Stroustrup的上述有关C++或者说带类的C的著作出版时，我们开始将类和对象与术语*抽象数据类型*关联起来。但是，数据类型与抽象数据类型到底有什么区别呢？数据类型是一组值。用某个算法对它进行操作，即可管理和更改这组值。抽象数据类型不仅有一组值，而且还有一组可以对这些值执行的操作。抽象数据类型背后的主要思想是将数据类型的使用与其实现分离开来。图1-1展示了一个抽象数据类型的4个主要部分。语法和语义定义了应用程序将如何使用抽象数据类型。表示和算法给出了可能的实现。

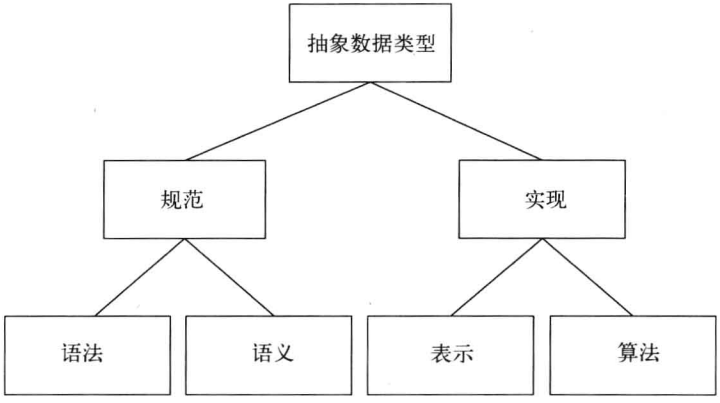


图1-1 抽象数据类型

对于抽象数据类型，我们定义了一组行为以及一系列抽象数据类型可能使用的值。使用数据类型不需要知道实现细节。我们用表示来定义值在内存中的表现方式。在VB.NET或C#中，这些表示称为*类成员变量*。算法或程序指定了这些操作如何实现。在VB.NET或C#中，这些程序为*成员函数*。语义指定了当输入任何可能值后，成员函数将会返回什么结果。语法指定了VB.NET或C#操作符或函数名称、所有操作数的数量和类型，以及成员函数的返回值。因此，我们要创建自己的数据对象（抽象数据类型），以便软件使用。这与仅使用编译器预定义的数据类型（如整型、字符型等）有所不同。在Grady Booch的*Object-Oriented Analysis and Design with Applications, Third Edition*（2007）一书中，这些抽象数据类型或对象定义如下：对象表示独立的、可识别的条目、单元或实体，它可以是实际的或抽象的，它在特定问题域中具有明确定义的角色。

有关对象的另一本经典书籍是*Design Patterns*（Gamma，1995）。本书指出了可重用的面向对象软件的元素。

1.3　预备知识

我们已经逐渐认识到程序应该有什么特性，这样它才能用于解决复杂的现实问题。仅靠像VB.NET或C#这样的新语言及其相关的功能来创建类和对象是不够的。而且我们还认识到仅使用抽象数据类型或类也是不够的。随着技术的不断发展，面向对象技术方法学逐步演化成对象模型。软件工程基础的元素统称为对象模型，它包括抽象原则、模块化、封装、层次结构、类型、并发性和持久性。对象模型定义了这些元素的使用，这些元素又形成了协同关联。

像所有学科一样（如数学中的微积分），我们需要符号或表示法来表达对象的设计。例如，C++语言为编写面向对象的程序提供了一种语言表示法。但我们还需要用一种设计方法学的符号来表达软件开发的总体方法。1991年，Grady Booch的*Object-Oriented Analysis and Design with Applications*一书的第1版面市，书中定义了一组表示法。这些表示法现在已经成为面向对象设计的事实标准。他在第2版中更好地描述了所有面向对象设计的表示法和对象模型。他用C++语言表达了所有例子，从而使C++语言曾一度成为面向对象软件开发的主流语言。甚至有基于这种表示法的Windows GUI工具来辅助我们进行构思。由Rational Corporation和Grady Booch开发的这款工具称为ROSE。这与微积分及其表示法最初如何得到广泛使用有点不同。我们几乎马上就有了编程所需的相同引擎来帮助我们。这个工具还在持续发展，它也就是现在我们所说的UML（Universal Modeling Language，统一建模语言）。

对象（或组件）是一个基于抽象数据类型理论的实体，在VB.NET或C#语言中，抽象数据类型是作为一个类来实现的，而且类结合了对象模型的特性。但是，我们所描述的有关对象的内容只是其冰山一角。到现在为止的描述已经说明了静态定义，尚未提及对象之间的通信。接下来我们看看其中一个对象模型特性：继承。继承对于软件领域来说，就相当于大规模集成电路（LSI）领域中的集成电路（IC）制造技术，是IC技术实现了电子系统制造的巨大进步。当然，目前使用继承的软件还很少，但这个方向已经确定。继承支持在软件中创建小规模集成（SSI）块。这个SSI将对象封装成一个软件群集（software cluster），形成应用程序所需的某个功能的解决方案。这样就可以将大量的复杂性抽象掉，程序员只需要针对这个软件群集的接口编程即可。程序员可以在这些软件群集之间发送消息，就好像设计的电子逻辑在多个IC之间设有电线一样，通过这些电线可以发送信号。

1.4　软件组件

虽然我们将软件组件比做硬件芯片，但只有从最一般的意义上讲，这种类比才成立。软件组件是用具有丰富词汇的编程语言创建的，以程序员头脑中的构思为基础，因此软件组件所具有的灵活性和解决问题的能力，要远比硬件芯片强大得多。当然，强大的背后则是软件程序极高的复杂性本质。但是，运行在硬件芯片之上的软件组件增加了另一级抽象。复杂的LSI芯片中涉及的主控逻辑实际是非常惊人的。但在与人类的思维对象建立起协同关系方面，LSI芯片的能力就极为有限了。

随着在.NET Framework的各项技术上对它的对象模型研究得越来越深入，我们越能感觉到自己正在将思维对象的行为机制外化出来。当然，在软件对象之间通过接口建立起来的链接和嵌入关系与影响神经元集群的树突的分布没有多少相似之处。但是从现在起，软件对象正开始为了实现自己的目标而影响一个或多个其他的软件对象。

接下来从日常实用的角度分析其他工程领域中使用的一个或一组控制对象。我们早期的一大爱好是汽车，所以迫不及待地要学习开车。请注意，我们说开车的时候，指的是任意一辆车。在车的封装和接口公开方面，我们做得非常棒，所以才能在学会驾驶一类车之后，就能驾驶其他任何类型的车。我们与之交互的汽车对象有三个主要的接口控件：方向盘、油门和刹车。我们知道在汽车对象中封装了许多内部功能。我们确信，在不同的汽车对象之间，这些控件接口不会发生变化。换句话说，如果从一辆通用公司的车换到一辆福特公司的车上，可以依靠这些控件接口的相同功能继续开车。

软件对象的另一个特征是持久性。孩子们很小的时候就学会了对象的持久性。我们拿一个玩具给小孩看，然后把它藏在背后的时候，小孩就会知道玩具仍然在那里。孩子已经将玩具这个对象概念化成为自己思维对象集合的一部分。在程序员对各种软件对象进行思维概念化的时候，就会在程序员的头脑中形成对象的高级持久性。因为可重用性是标准软件对象的主要特性之一，所以随着程序员在思维模型中概念化出标准对象，程序员的效率就会持续提高。

多态行为是在软件对象中可以实现的另一个特征。有些形态可具有数种不同的行为，孩子们可能很早就意识到椅子就是这样一种形态。椅子这种对象的多态性在于它的行为取决于它的形式。我们有摇椅、餐椅、躺椅等。形式与相关行为的思想已经形成了一个完整的研究领域，即形态学。当然，这对于在认知上将各种对象关联在一起来说是一个关键的思想。不仅对象之间有形态上的联系，而且对象的内部构造之间也存在形态上的联系。在程序的逻辑流程和各个有意义的程序块的布局之间存在着确定的关系。这与函数纯粹的多态性略有不同，但的确指出了我们应该关注程序中对象、对象的组成部分及其布局之间的形态关系。

第2章 WDF体系结构

Windows操作系统系列的下一代驱动程序模型是WDF。这个新模型可以减少驱动程序的开发时间，实现更好的系统稳定性，提高驱动程序的适用性。本章将介绍WDF的整体驱动程序模型以及它的各种功能。在第二部分和第三部分的几章中，我们将深入介绍开发这两类驱动程序的编程细节。所以本章的目的就是让读者对WDF驱动程序模型的体系结构有一个总体的感觉。注意，一般来说，文中如果出现编程结构或者变量，会用代码体标识。其中当然包括可用于开发驱动程序的各种WDF API。

2.1 WDF组件的功能

WDF包含一整套支持开发、部署、维护内核模式驱动程序与用户模式驱动程序的组件。WDF组件与现有的驱动程序开发工具配合使用，能够满足下面整个驱动程序开发周期的需求。

- 规划和设计：驱动程序模型——WDF驱动程序模型支持创建面向对象的、事件驱动的驱动程序。通过WDF，驱动程序开发人员可以把注意力集中在设备的硬件而不是操作系统上。可以针对内核模式编写WDF驱动程序，也可以针对用户模式编写。
- 开发：框架和Windows驱动程序工具包（WDK）——WDF定义了一个驱动程序模型，提供了开发内核模式驱动程序和用户模式驱动程序的框架。WDF的框架提供了支持WDF模型的基础设施。它们实现了公共功能，提供了智能默认设置，负责管理与操作系统的大多数交互。

  KMDF（Kernel Mode Driver Framework，内核模式驱动程序框架）实现了Windows必需的基本内核模式驱动程序支持功能，这些功能是所有内核模式驱动程序的公共功能。

  UMDF（User Mode Driver Framework，用户模式驱动程序框架）提供了与KMDF类似的功能支持，但是允许某类设备的驱动程序不在内核模式中运行，而在用户模式中运行。
- 测试：跟踪和静态分析工具——KMDF和UMDF都有内置的验证代码，并凭借Windows事件跟踪（ETW）支持集成的跟踪功能。生成的跟踪信息有助于在开发期间对驱动程序进行调试，诊断已发布驱动程序的问题。WDF驱动程序还可利用现有的驱动程序验证工具。另外，有些编译时驱动程序验证工具也是WDF的一部分。例如PREfast和SDV（Static Driver Verifier）。