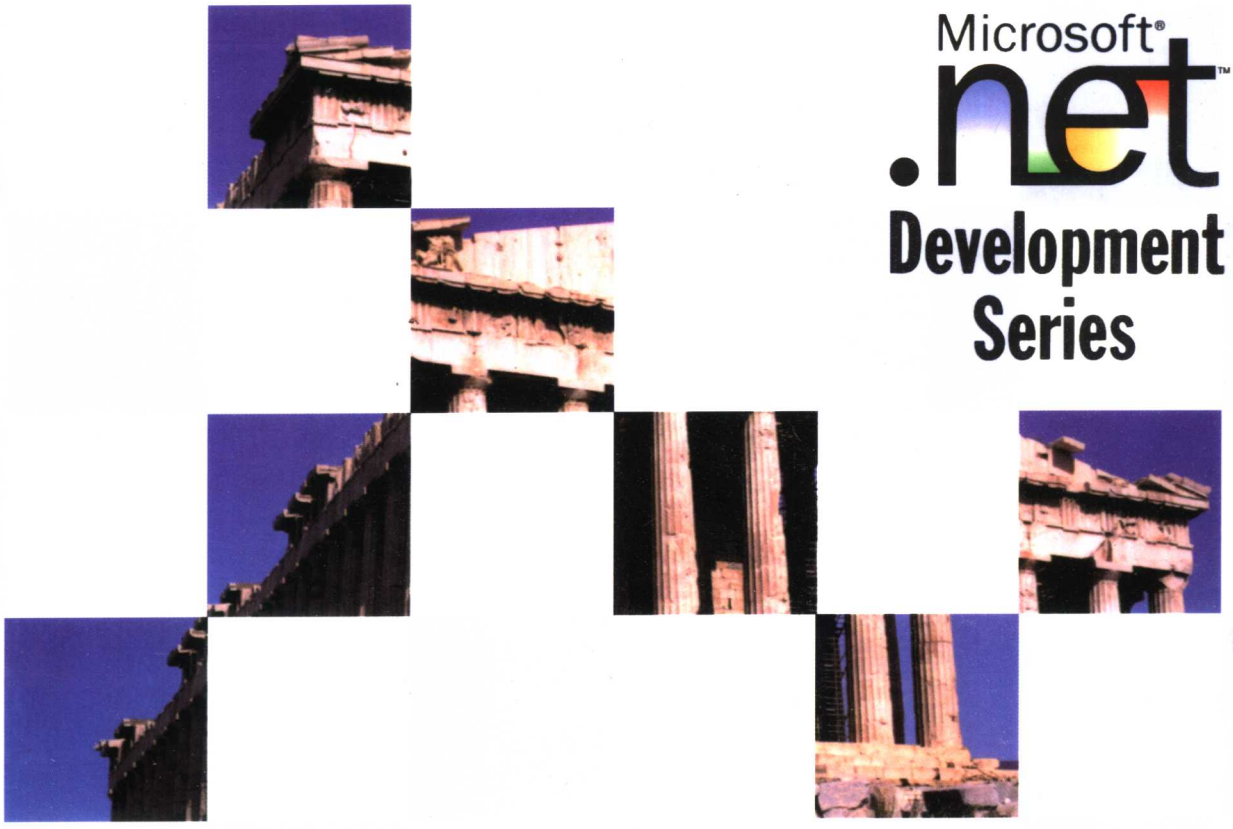
“好戏即将上演……”
——Don Box，微软公司软件架构师

WF本质论

Essential Windows Workflow Foundation

Dharma Shukla
Bob Schmidt 著　　周健 译



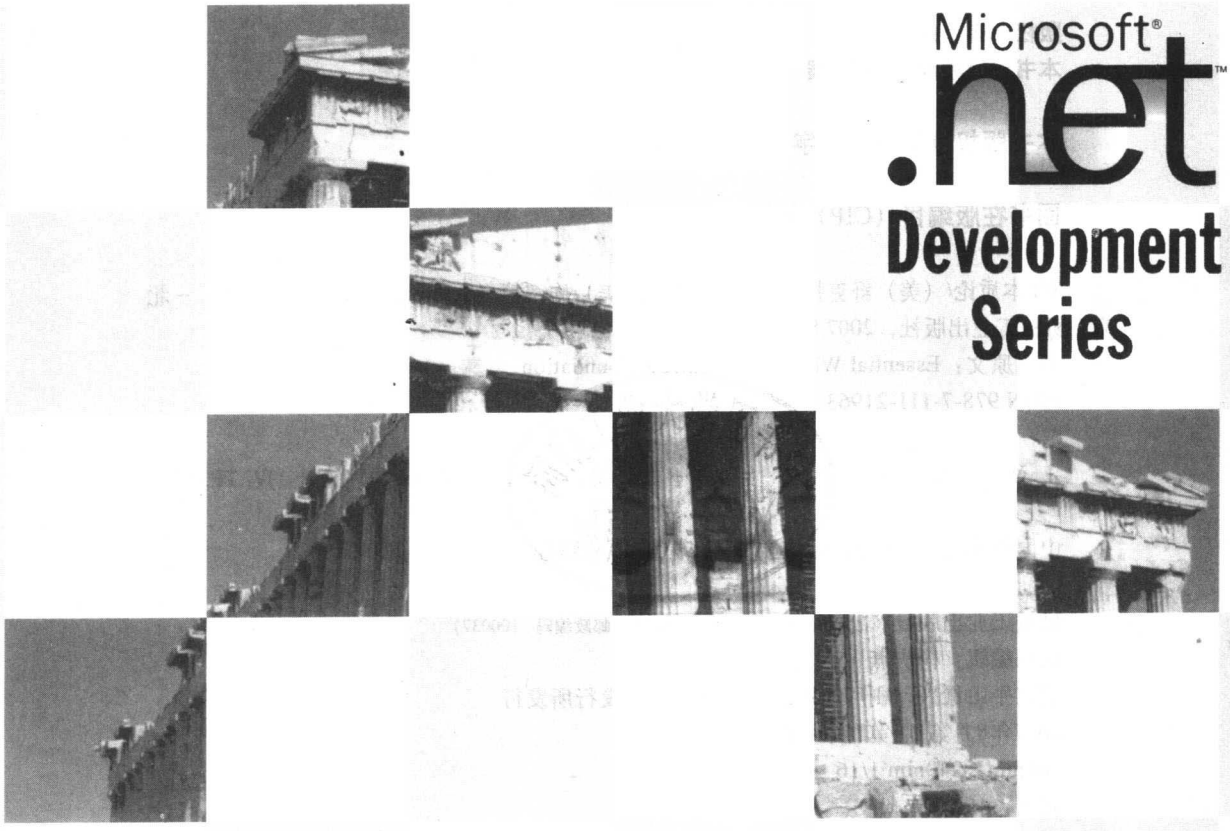
Microsoft® .net™ Development Series

机械工业出版社
China Machine Press

WF本质论

Essential Windows Workflow Foundation

Dharma Shukla
Bob Schmidt 著　周健 译



机械工业出版社
China Machine Press

本书详细讲解Windows Workflow Foundation（WF）的运作原理，并在解析原理的过程中给出WF的设计初衷。WF使用了一种开创性的方法来编写和执行程序。本书两位作者从WF项目立项开始就参与这个项目，负责规划、设计、开发了其中的大部分技术。在本书中，作者高屋建瓴地探讨了WF中的关键概念和整体架构，不仅涉及如何使用WF，还涉及为什么这样使用WF，揭开了WF技术的神秘面纱。通过简单而极具说明性的例子，本书演示了如何利用WF的可扩展性编程来构造具体领域的程序。

本书版权登记号：图字：01-2006-6895

图书在版编目（CIP）数据

WF本质论/（美）舒克拉（Shukla, D.），（美）施密德（Schmidt, B.）著；周健译. －北京：机械工业出版社，2007.8

书名原文：Essential Windows Workflow Foundation

ISBN 978-7-111-21963-7

Ⅰ.W… Ⅱ.①舒… ②施… ③周… Ⅲ.窗口软件，Windows-程序设计 Ⅳ.TP316.7

中国版本图书馆CIP数据核字（2007）第116333号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：杨庆燕

北京牛山世兴印刷厂印刷 · 新华书店北京发行所发行

2007年8月第1版第1次印刷

186mm × 240mm 1/16 · 21印张

定价：45.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

译 者 序

当看到这本书的英文名称时，你是否会联想到另一本重量级的书《Essential COM》。诚然，这两本书除了名称类似外，确实还有些渊源，为本书作序的正是《Essential COM》的作者，Don Box。既然本书能冠以Essential的名号，那必定有过人之处。

首先，据我所知，这是市面上第一本深入剖析WF（Workflow Foundation）的书（本书的中译本更是关于WF的第一本中文资料）。本书详细解释了WF的底层运作原理，并在解析原理的过程中，交代出了WF的设计初衷。从这个角度看，可以说这不是一本写给WF初学者的书籍。它不能帮助读者学习如何使用WF设计器，也没有介绍出现在工具条中的每个活动，事实上，本书通篇没有介绍任何一个WF自带的活动。但通读本书后，读者就能完全了解WF架构者的设计目的，从而最大限度地发掘WF的潜力。可以这么说，WF是架构在CLR之上的另一个CLR，它完全改变了传统的程序开发方式。

其次，本书的两位作者见证了孵化WF的全过程。Dharma Shukla是WF的架构师，而Bob Schmidt则是WF团队的项目经理。关于WF，此二人应该是最有发言权的。也正是因为这个原因，才使得本书具备相当的深度。另一个不得不提的人就是Don Box，他的深刻洞察力以及把握全局的能力，为本书提供了强大支持。

本书以一个最简化的Hello World程序开篇，随后渐进式地改造该程序，最后该程序具有所有WF中的核心特性，可谓是麻雀虽小五脏俱全。在第1、2章中，作者从无到有地构建了一个最简化的WF程序。第3、4两章则正式切入WF，介绍了作为WF核心概念的活动的整个生命周期。随后几章则从WF的其他几个侧面入手，介绍了WF运行时的几个核心特性。当然，本书不是一本参考手册，所以不能也没有必要包罗万象。

早在大学期间，我就开始关注工作流技术了，但那时对业界来说，工作流技术始终是阳春白雪，直到WF的出现。WF第一个beta版发布后，我急急地试用了一番，由此我预感到这将是一种使工作流走向普及的技术。去年本书的原版面世，我又迫不及待地浏览了一番，这让我对WF有了新的认识：WF不仅仅是一个嵌入式工作流引擎，它更是一种新颖的编程方式。它打破了传统的以编程语句为原子单元的编程方式，取而代之以活动作为编程的基本元素。这让程序的逻辑控制流等元结构得以无限扩展。因此，当陈冀康编辑邀我翻译本书的时候，我甚感荣幸。

本书的翻译少不了周围朋友的支持。

感谢陈冀康编辑对我的信任。与陈编辑的合作十分愉快，这段经历将使我终身难忘。

我还要特别感谢我的挚友沈群超，他的文字功底和语言组织能力使得原本生硬的语句变得生动。他在新项目启动的强大工作压力下，完成了全书的审阅工作，没有他，这本书就不可能这么快和读者见面。

感谢我的导师兼上司黄纪纲。在得知我正在进行本书的翻译工作后，他一直在给予我鼓励和支持，协助我校对错误。甚至于在工作安排上也照顾到我的翻译进度，有这样老板的公司怎么能不成功。

我还得感谢我的女朋友。她虽然在工作上帮不上什么忙，但每天晚上回家后看到她的笑脸对我来说又何尝不是一种最大的安慰呢。

在这里，对所有给予过我帮助而我又没能提到的朋友们一并致谢。

由于译者水平有限，加之时间仓促，译文中有错误或不当之处，敬请读者不吝指正。我的邮箱是 zhoujian.ww@gmail.com。

周健

2007年7月

序　言

2006年7月，我写下了这篇序，当时我已经预感到这将是一个美好的开始。

起初，程序员们就已经发现，将表达其意图的程序用机器可读的方法表达具有举足轻重的作用。也正是这个念头，导致了助记码的广泛使用。

更为重要的是，不仅开发者已经逐渐意识到，相关领域的专家也已经发表了专业意见——目前尚未存在任何运行时框架或语言设计器能完全胜任数据模式的定义工作——这种观念促使了行业相关语言的诞生。

Smalltalk、Java虚拟机（JVM）以及通用语言运行时（CLR）等系统已经证实：诸如反射、序列化、产生式编程等类型定义的机器可读标识法是有其实际价值的。但是，类型的基本表示法（诸如字段、方法、类）很大程度上是一个封闭的世界，它不允许用户定义诸如控制流、并发、逻辑结构或者领域相关的概念（比如贴现政策或者四分音符），除非在将一个操作不透明[⊖]地贯穿于多个方法体。

现在，人们开始反问自己："既然类型定义能用于数据，那为什么其他设计期结构不能呢？"

幸运的是，有一个人早在2003年就提出了这个问题，他就是我的好友，曾就职于Biztalk Server团队的Dharma Shukla。

当Biztalk Server团队的成员开始将XLANG orchestration engine封装为一个可嵌入的平台组件时，他们已经可以把语言结构从XLANG移植到另一个XML语法的语言上了（事实上，这项提案确实已经提上了日程）。对于构建Windows Orchestration Engine（WinOE）来说，这是迄今最为符合设计目的的方法。

幸运的是，Dharma先生早已极具预见性地洞察到：对所有的程序定义"一种实模式"并非可行之计，相反，他认为应该将它"元化"，用户可以为他们所工作的领域定义自己的操作码，于是WF项目朝着构建一个可扩展运行时的方法进行。这个重要的决策使你能使用XML语法来定义和组合这些操作码，因此你有了一个系统，这个系统可以让你亲自决定用于描述某个领域中的应用程序的词汇表和文句结构。

在本书中，Dharma Shukla和Bob Schmidt向我们展示了Windows Workflow Foundation（WF）是一个极好的元运行时的例子，它使程序员将程序的编写与执行置于其管理之下。开发者为程序定义模式，并提供一个该模式的解释器来使其能被部署和运行。这个简单的概念会有大量衍生体。

是的，我说过，好戏即将上演，我们坚信，并执着期待。

Don Box

2006年7月，华盛顿州 Yarrow Point

⊖ 不透明的含义为类型使用者不得不关注将一个操作分布于多个方法体这个事实。——译者注

前　言

Windows Workflow Foundation（WF）是一个通用的编程框架，它可用于创建需要对外部实体的信号作出响应的交互式程序。交互式程序的基本特征是它会在执行期间暂停某一长短未知的时段，以等待输入。

当然，交互式程序并不是什么新现象。自从计算机网络出现以来，交互式程序就广泛应用于两台计算机之间的通信。这些技术还适用于运行在单台计算机上的数据交换。这种技术的变体涵盖了从套接字到Web服务等各种技术，这些技术在程序通信领域已应用了多年。尽管在这几年中，编程模型的互操作性、可伸缩性以及易编程性等方面有了长足的发展，但开发人员在程序交互方面的投入依然匮乏。比如，时下流行的Web编程模型，这类模型通常为开发人员强加控制流模型，且非常生硬。因此，WF希望改变这一切。

WF编程模型的核心概念就是活动——WF程序中的程序语句。活动的执行具有固有的可恢复性，同时以片段式的方式执行，它根据活动与外部实体的交互来暂停和恢复执行。当你在深夜合卷睡觉前，一般都会在书本当前阅读的位置放置一个书签，对WF中的活动来说也是这样，当活动等待外部输入时，它会在当前执行的位置做一个书签，当收到外部输入打算恢复执行时，则从书签的位置继续执行。

WF是一个框架，而不是一组由语法定义的编程构造。WF中活动的概念是可扩展的，这使得WF程序中的表达式以及控制流相对于C#或Visual Basic这样的语言要丰富得多。你可以使用领域专有的活动以及特定的控制流结构来表示WF程序，因此，它能通过捕捉复杂的人机交互来高度拟真特定的场景。

WF运行时是一个元运行时，它建立在通用语言运行时（CLR）之上，并管理着WF程序的执行。在一个分布式的环境中，WF程序可以在不需要做任何额外工作的前提下，相当自然地暂停或恢复，也可以执行任意长一段时间。在WF程序空闲时，它既不会执行失败，也不会过度消耗系统资源。在WF的上下文中，CLR的工作就是管理那些在内存中代表WF程序的对象。而WF运行时的工作就是管理WF程序的完整生命周期，WF程序的生命周期是可以跨越CLR线程、应用程序域、操作系统进程甚至机器的。

总之，WF提供了一个编写和执行交互式程序的编程模型。WF程序是由被称为活动的领域相关程序语句构建的。这允许领域专家可以使用该领域的概念来描述。

关于本书

我们撰写这本书有一个简单的理由，那就是我们都相信在WF的核心有一个令人瞩目的概念。在当今主流平台上，这些概念还是第一次出现。WF的面向活动编程方法采用了与时下流

行的编程范式完全不同的原则，因此，学习WF最好的方式就是首先关注WF编程模型赖以建立的基础概念。仅仅熟悉System.Workflow命名空间下的一系列类型，对于WF开发者来说是远远不够的。

总而言之，这本书不会介绍WF的三个程序集中350个以上的类。WF有意地回避了WF的外围设施，而将注意力集中于WF的精髓——核心编程模式和运行时特性集。大量经验告诉我们，从基本原理着手来学习一个框架是成为资深WF开发者的最可靠路径。

撰写这本书是对我们信念的考验，也是对交互程序所做假设以及这类程序开发方式的检验。如果这本书以及WF中的新概念能帮助读者更好地构建应用，那么我们就成功了。

本书的组织如下。第1章介绍支撑WF编程模式的一些关键概念：书签、延续、线程与进程灵活性、钝化、可恢复程序语句以及可恢复程序的运行时。对这些概念的讨论都游离于WF之外，以最简洁的形式出现，以求使讨论最明确化。

第2章将第1章中介绍的概念映射到了WF编程模型中，因此，第2章可以说是接下来几章的桥段。我们从这里开始开发活动，将这些活动组合为简单程序，从而运行这些WF程序。

第3、4章讨论活动执行的一些细节方面，包括书签机制、错误处理、取消操作以及补偿逻辑。而描述活动生命周期的活动自动机则是这两章一致的主题。第5章探讨构建WF运行时宿主程序的方法，同时展现WF的可扩展性。第6章讨论WF程序执行中事务的临界区功能。第7章讲解许多与活动和WF程序创作相关的进阶话题，其中包括验证和编译。第8章介绍WF核心概念的若干外围特性。

附录A是活动自动机的一个参考。附录B为我们展现若干复合活动的代码，这些复合活动比正文中所开发的复合活动复杂得多。这些例子强调了WF编程模型的那些扩展点，同时演示了复合活动构建复杂控制流模型的能力。

在本书中，我们完全把重点放在Windows Workflow Foundation上。我们假定读者已经掌握C# 2.0和CLR的基本知识。如果读者还不了解这些，可以参考《The C# Programming Language, Second Edition》，Anders Hejlsberg等（Addison-Wesley，ISBN：0321334434）和《Essential .NET, Volume I: The Common Language Runtime》Don Box和Chris Sells（Addison-Wesley，ISBN：0201734117），以上列出的书都是这一领域的权威资料。本书中的示例都十分简单——我们有意识地降低场景对示例的干扰，将注意力集中在概念本身。在你掌握了窍门之后，我们期望你能尝试将本书中的概念和技术应用到你所从事的行业中。

致谢

WF项目开始于三年前。当时我们这个小型工程师团队已经强烈地预感到：改变陈旧的程序构造方式，开创一种崭新的程序开发方法已经势在必行。我们的首次尝试发生在与Microsoft Office团队合作的时候，当时，我们思忖着该项崭新的技术是否能成为他们计划中的工作流特性的基础结构。我们的工作进展十分顺利，WF成为了Microsoft Office 2007中工作流特性的基础，同时更让我们引以为荣的是，WF的第一个公开版本是作为.NET Framework 3.0的一部分发布的（众所周知，.NET Framework 3.0是和Windows Vista一起发布的）。许多人为WF的设计出

谋划策，没有他们的意见和建议，WF就没有今天的公开发行版。在这里，我们不可能对每个为WF作出贡献的人们致谢，但是我们还是向他们所做的努力致敬。

我们的审校也做了相当多的工作，他们告诉我们什么该着重强调，哪里不能着墨过多，他们还帮我们指出了各章节的错误和遗漏。他们是：Angel Azcarraga、Don Box、Krzysztof Cwalina、Joe Duffy、 Omri Gazitt、Ian Griffiths、Mark Michaelis、Dennis Pilarinos、Jeffrey Richter、Andrei Romanenko、Akash Sagar、Chris Sells、Clemens Szyperski、Nate Talbert、Eric Zinda，正是有了以上各位的反馈意见，才能使本书变得更好。Ian提供了极具针对性和洞察力的意见；Aditya Bhandarkar在设计期间一节中提供了额外的帮助；Don不仅提供建设性的意见，还慷慨地捉笔为此书撰写序言。

对我们来说，在宝贵的业余时间撰写这样一本书确实是一个挑战。我们要感谢Addison-Wesley公司的Karen Gettman和Curt Johnson，所有耐心指导我们的出版界专家们，以及所有将我们的文字最终变为书本并呈现在您面前的人们。

如果本书还有什么纰漏的话，那一定是我们的缘故了。

我是幸运的，因为我从WF诞生那一刻起就参与其中。WF是我迄今为止在微软做过的最有趣的项目。对我来说，撰写本书是构建WF技术的一种延续。现在这本书完成了，我也可以好好休息一下了。

多年来，Don Box一直是我的灵感源泉，他一直鼓励我撰写这本书，对此我十分感激。Don帮助我构思本书的内容，他废寝忘食的精神以及和蔼可亲的态度，我将铭记在心。再次深表感谢，Don!

Bob Schmidt和我共同完成了本书的撰写工作。和Bob一起工作是一段十分愉快的经历。我们先是一起构建WF技术，随后共同完成本书。谢谢你，Bob，祝旅途愉快。

还要感谢那些一线的编程人员，感谢你们将不可能变为可能。

我一直十分感激Abhay Parasnis，他创建了一套独特的环境，不仅为孕育技术革新提供了温床，并且奇迹般地在短时间内让实现新技术成为可能。良好的开端是成功的一半，在WinOE期间的工作给我留下了美妙的回忆。

感激我的父母为我不计回报地付出，还有Pushpa坚定的支持。

我还要感谢我的妻子Bina，她总与我形影不离，还有我的女儿Anya，使一切都变得这么美好。

Dharma Shukla

2006年7月

作为首个研发经理加入WF团队，我确信这将是个研究改变程序开发方法的契机。能够为这种新技术的开发贡献自己的一份力量，我深感荣幸，与此同时，我也十分荣幸地参与了撰写这本关于WF的书。

Dharma Shukla诚邀我与他合著这部书籍，我深表感激，他的洞察力、热忱以及对真理的追求，使得我们彼此的讨论总是熱情洋溢，而他的宽阔眼界与丰富的想象力，为这部书疏理出清晰的脉络。

我要对Don Box表示感谢，这期间，我们从他那里得到了很大的支持和指导，Don Box有着敏锐的目光，设计的初始思路中着重于基础的理念正是来源于他，这个理念使得我们能够凝聚力量。

Abhay Parasnis不仅在WF项目中是绝对的主力，同时还起到了模范带头作用，他的领导能力在无形中激励着我们。Abhay Parasnis理应得到褒奖，这无可厚非。

由衷地感谢我的妻子Elaine，这么多年来她一直为我默默地付出，她是我坚强的后盾。我的两个儿子，Thomas 和Andrew，作为父亲，因为著书而频繁地奔波于各项会议而无暇照顾他们的时候，他们总是在耐心等待而没有无理取闹，他们甚至知道什么时候才能进入我的办公室以提供给我些许乐趣与灵感——我向你们保证：这个美好的夏天将充斥着冰淇淋和你们喜爱的壁球运动，一同期待吧。还有我的父母，你们让我领悟到，全身心投入的团队能创造多么伟大的奇迹，你们对我的帮助是无价的，感谢你们。

Bob Schmidt

2006年7月

目　录

第 1 章

剖 析 WF

关于编程技术的书籍往往都是以介绍一个名为“Hello, World”的小程序开始，这个程序用于在标准输出设备上打印一条简单的信息。在此，我们也未能免俗，以下是这个程序的C#版：

```
using System;

class Program
{
  static void Main()
  {
    Console.WriteLine("hello, world");
  }
}
```

“Hello, World”之所以能够成为一个较为流行的编程起点，归功于它的简单，因为它避免了许多实际程序所需要关注的问题。软件开发从业者肯定知道，扩展像“Hollo, World”这样的程序，会很快碰到棘手的问题。我们来看看这个名为“Open, Sesame”的例子，它在打印传统的问候语之前，要求用户键入一个口令：

```
using System;

class Program
{
  static void Main()
  {
    // Print the key
    string key = DateTime.Now.Millisecond.ToString();
    Console.WriteLine("here is your key: " + key);

    string s = Console.ReadLine();

    // Print the greeting if the key is provided
    if (key.Equals(s))
      Console.WriteLine("hello, world");
```

```
    }
  }
```

“Open, Sesame”这个程序各个方面几乎都不怎么出彩，不过我们依然看到了它值得一看的地方：由于依赖于用户在控制台上所输入的一行内容，所以，从程序执行开始到结束，在执行时间上的花费可以是任意长的。你可以编译并运行这个程序，并且把它搁置几周时间，然后再键入口令并最终打印出欢迎信息。这一切都是设计使然。

“Open, Sesame”是一个交互式程序（reactive program）的例子。所谓交互式程序，就是程序在执行期间依赖外部实体的刺激，并对此做出响应。有时，这个外部实体是一个人，有时，这个外部实体是另一个程序。但无论是哪种方式，交互式程序都将花费大量的时间在等待这些外部刺激，所以，现在我们面临的挑战不是等同于编写诸如“hello, world”程序这般轻松了。

大多数的计算机程序是交互式的。现实世界中，你会发现各种各样的处理过程几乎都有软件的身影：文档协作编辑、客户订单管理、原始资料提取、纳税申报单的预备、供应、产品开发管理、在线商店、客户关系管理、车间和仓库操作的协调等。这个列表的内容还会不停地增加。在这些处理过程中，交互式程序需要对人或者其他程序所提供的输入信息做出适当的反应。

一些交互式程序的开发使用了某些架构，比如ASP.NET和Java Servlets。而另一些原生（homegrown）解决方案则直接构建在执行环境之上，这些执行环境包括通用语言运行时（CLR）和Java虚拟机（JVM）。还有一些程序则是用C或者（非托管）C++之类的语言编写的。

然而，如果我们观察这些交互式程序的编写过程，就会发现大部分的程序跟前面提到的“Open, Sesame”程序没什么共同点。现在就来看一个Web Service程序（Web应用程序也不失为一个极具启迪性的选择），这个程序完成了和“Open, Sesame”程序一样的功能。

现在，我们把“Open, Sesame”程序移植到ASP.NET Web service上来：

```
using System;
using System.Web.Services;

[WebService]
public class Service : WebService
{
  [WebMethod(EnableSession = true)]
  public string PrintKey()
  {
    string key = DateTime.Now.Millisecond.ToString();
    Session["key"] = key;
    return "here is your key: " + key;
  }

  [WebMethod(EnableSession = true)]
  public string PrintGreeting(string s)
  {
    if (Session["key"].Equals(s))
      return "hello, world";
```

```
    return null;
   }
  }
```

不难看出，这个Web service有两个操作。但是在这个程序里已经完全找不到“Open, Sesame”中的控制流了，实际上，PrintKey必须在PrintGreeting之前被调用，并且，每一个步骤都必须严格地执行且仅可以执行一遍，只有这样，程序才是成功地完成了执行。为了建立操作顺序的约束，我们添加以下黑体字标出的代码来修改Web service：

```
using System;
using System.Web.Services;

[WebService]
public class Service : WebService
{
  [WebMethod(EnableSession = true)]
  public string PrintKey()
  {
    bool alreadyDidStep1 = (Session["key"] != null);
    if (alreadyDidStep1)
      throw new InvalidOperationException();
   string key = DateTime.Now.Millisecond.ToString();
   Session["key"] = key;
   return "here is your key: " + key;
  }

  [WebMethod(EnableSession = true)]
  public string PrintGreeting(string s)
  {
    bool didNotDoStep1Yet = (Session["key"] == null);
    if (didNotDoStep1Yet)
      throw new InvalidOperationException();

    bool alreadyDidStep2 = (Session["programDone"] != null);
    if (alreadyDidStep2)
      throw new InvalidOperationException();

    Session["programDone"] = true;

    if (Session["key"].Equals(s))
      return "hello, world";

    return null;
  }
}
```

我们现在使用一组运行时检查以确保程序有正确的Web service控制流，但那些程序逻辑既分散又不明显，而且还很容易导致错误。“Open, Sesame”控制台程序中那些原本一目了然的顺序变成了难以理解的逻辑被分散到了Web service的各个操作中。设想一下，仅仅提供Web Service的源码，也不用关心数据流，以求达到理顺程序控制流的目的，对于仅仅包括两步操作

的简单例子，我们也许只要花数秒钟时间就能理顺其中的关系，但倘若是一个规模十倍于此的Web Service程序，并且该程序中还包括分支和循环的控制流，我们又将如何自处？

为什么我们不用更自然的C#控制流结构来指定Web service操作之间的关系，以达到约束何时发生什么事件的目的？毕竟我们是在用C#编程。“Open, Sesame”控制台程序有着精确的控制流，并操纵着程序所需要的局部变量。为什么不能用这种方式编写Web Service、类似的Web程序，或者任何实际的程序呢？

我们有两点解释：

- 在“Open, Sesame”控制台程序中，Console.ReadLine会阻塞调用线程的执行。程序也许会花费数天时间在等待输入上。如果有几个这样的程序在同时运行，他们都在等待输入，系统就会慢慢地停下来。这种线程处理方式并不适用于现实世界的程序，尤其是那些部署在多用户环境的程序。
- 现实世界的处理过程往往都要持续相当长的一段时间——几天，几个星期甚至几个月。我们当然希望在此期间操作系统进程（或者CLR应用程序域），能保持正常运行。

对于一个只具备教学意义的控制程序来说，上述这些问题很少会困扰我们。所以，我们能够用“Hello, World”这种非常自然的方式来编写“Open, Sesame”程序。阅读它的代码就能精确地知道程序做了些什么，但是对于我们的Web Service程序，这恰恰相反。开发Web Service和Web应用程序时，伸缩性和健壮性往往更重要。

ASP.NET运行时设计成可以有效地管理多个服务和应用程序，并能有效地维护独立会话的状态（在正确的配置后，会话能够跨越主机）。然而，代码是不会说谎的，程序的伸缩性和健壮性是需要极大代价的。在上述Web Service中，两个操作之间共享的key变量利用了一个弱类型的名称值对（name-value pair）。此外，为了确保操作的执行顺序而需要在每个操作的开始处检查变量（比如key）是否已经赋过值，这种逻辑十分别扭。

以此看来，更好地建立程序伸缩性、健壮性和更自然地表达交互式程序的状态和控制流，仿佛是鱼和熊掌不可兼得了。世界上成千上万的Web程序和Web service证实了，为了完成工作，开发者心甘情愿地受限于今天的编程模型。尽管如此，Web编程范例仅仅是交互式程序的一部分合适的解决方案。当务之急便是找到一种更好的、多用途的方式来开发交互式的、基于网络的程序。这种方法必须具备以下几个特点：

1. 存在一种编写交互式程序的方式，它并不牺牲自然的控制流结构，相反，事实上这种编写方式恰恰增强自然的控制流结构。

2. 存在一种既具可伸缩性又能健壮地运行交互式程序的方式。

1.1 线程进程灵活性

我们为“Open, Sesame”所做的ASP.NET Web Service解决方案具有可伸缩性和健壮性，但控制台程序却没有。让我们来仔细看一下下面这条语句，它看起来似乎是问题的根源。

```
string s = Console.ReadLine();
```

问题的重点在于：当“Open, Sesame”控制台程序调用Console.ReadLine后，线程就会阻

塞（忽略错误的场景），直到有相应的信号到达后，线程才会继续执行。有些场景要求相当多的“Open, Sesame”实例（或者其他类似的程序）同时运行，让每个程序实例拥有高昂的线程资源（的做法）很难构建出伸缩性强的解决方案。

解决这个问题的一个常见的方法是采用异步方法调用的方式分出一个线程去执行。举个例子，按照标准的.NET Framework模式的做法，ReadLine方法可以通过一对Begin, End方法来达到异步调用的目的。

```
public static System.IAsyncResult BeginReadLine(
  System.AsyncCallback asyncCallback,
  object state
);

public static string EndReadLine(System.IAsyncResult ar);
```

在BeginReadLine方法内部，建立了一个工作请求，并把它加入到CLR的工作队列中。CLR线程池中的某个线程会异步地处理这个请求，但在此期间，BeginReadLine方法将会直接返回到调用者，而不是等待请求处理完成。

调用BeginReadLine的线程可以通过轮询System.IAsyncResult对象的IsCompleted属性来获取当前的工作状态，或者也可以使用AsyncWaitHandle属性来等待工作完成，这种方式比轮询高效。以下为后者的示例代码：

```
using System;

class Program
{
  static void Main()
  {
    // Print the key
    string key = DateTime.Now.Millisecond.ToString();
    Console.WriteLine("here is your key: " + key);

    IAsyncResult result = BeginReadLine(null, null);
    result.AsyncWaitHandle.WaitOne();
    string s = EndReadLine(result);

    // Print the greeting if the key is provided
    if (key.Equals(s))
      Console.WriteLine("hello, world");
  }
}
```

尽管这个程序使用了异步调用，但它仍然占用着一个线程，用来调用IAsyncResult对象的WaitOne。

注意到BeginReadLine方法的两个参数，就可以使我们跳出上面的两难境地。.NET Framework中的异步方法允许调用Begin时传递一个System.AsyncCallback类型的委托，Begin会在异步方法完成后调用这个委托。Begin的另一个参数是一个对象，这个对象在回调和调用代