OGRAMMER TO PROGRAMMER™

Visual Basic .NET
Serialization Handbook

Visual Basic .NET
串行化参考手册

Andy Olsen
Matjaz B.Juric 等著
杨 浩 译

清华大学出版社

Visual Basic .NET 串行化参考手册

Andy Olsen
Matjaz B. Juric
等著

杨　浩　译

清华大学出版社
北　京

北京市版权局著作权合同登记号：01-2002-5389

内 容 简 介

.NET Remoting 和 Web 服务等技术在很大程度上都依赖串行化。本书专门向开发人员介绍串行化在.NET 中的重要性，论述串行化的易用性、强大功能和可定制性。

本书主要面向 VB.NET 开发人员，深入讨论.NET 中的两个主要串行化机制—— 浅串行化和深串行化，阐述如何设计用于浅串行化和深串行化的类，并介绍了高级浅串行化和深串行化技术，最后简要介绍在使用.NET 串行化机制时应重视的安全问题。

本书适合于已经编写、编译和运行过 VB.NET 代码，熟悉自己选择的开发工具的中高级 VB.NET 开发人员，也适合于对串行化技术感兴趣的开发人员。

EISBN：1-86100-800-7

Visual Basic .NET Serialization Handbook

Andy Olsen，Matjaz B. Juric et al

图书在版编目(CIP)数据

Visual Basic .NET 串行化参考手册/(美)奥尔森等著；杨浩译。—北京：清华大学出版社，2003

书名原文：Visual Basic .NET Serialization Handbook

ISBN 7-302-06496-2

Ⅰ.V… Ⅱ. ①奥…②杨… Ⅲ.BASIC 语言—程序设计—技术手册 Ⅳ.TP312-62

中国版本图书馆 CIP 数据核字(2003) 第 022912 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. com. cn

责任编辑：夏兆彦

印 刷 者：北京大中印刷厂

发 行 者：新华书店总店北京发行所

开　　本：787×1092 1/16　**印张**：18. 25　**字数**：378 千字

版　　次：2003 年 4 月第 1 版　2003 年 4 月第 1 次印刷

书　　号：ISBN 7-302-06496-2/TP・4882

印　　数：0001～4000

定　　价：36. 00 元

前　言

串行化是.NET Framework 的幕后英雄。虽然.NET Remoting 和 Web 服务声名鹊起，但人们并未意识到这些技术实际上在很大程度上依赖于串行化机制。仔细研究一下 ADO.NET，就会发现 DataSet 在广泛使用串行化，而 COM Interop 不能没有串行化，ASP.NET 也使用串行化存储会话数据。

Wrox 认为串行化为.NET 所做的许多工作，以及它能为 VB.NET 开发人员做的工作一直未得到应有的重视。Wrox 编写本书，专门向开发人员介绍串行化在整个.NET 中的重要性，更重要的是，论述串行化的易用性、强大功能和可定制性。

如果读者要在应用程序的非运行期间保存 VB.NET 对象的状态，通常串行化是最佳的选择(也常常是惟一的选择)。因此本书希望帮助 VB.NET 开发人员更好地利用这个重要而有效的机制。

本书将讨论如何快速掌握.NET 串行化技术。学习本书后，读者应很好地理解如何使用.NET 的默认串行化机制来串行化类，如何设计类，优化它们自己的串行化过程，如何定制机制以获得自己希望的串行化结果。本书将介绍一些处理棘手情况的技巧和提示，例如反串行化单个对象。更妙的是，读者将学习到需要进行串行化的场合，在这种场合下，应使用哪种类型的串行化。这将占去本书的大量篇幅，让我们快速深入主题。

本书读者对象

VB.NET 手册系列的所有书籍都是面向 VB.NET 开发人员的，他们需要学习更多的知识来完成某项特定的任务。这些开发人员已经编写、编译和运行过 VB.NET 代码，熟悉自己选择的开发工具。本书要完成的任务是介绍如何有效地控制和使用.NET 的串行化机制。

本书主要内容

本书主要面向 VB.NET 开发人员，深入探讨了串行化机制，说明何时应使用串行化，如何定制串行化，以及为什么要使用串行化。我们将介绍一些好的经验，指导读

者根据自己的需要作出正确的选择。

- 第 1 章：.NET 串行化简介

首先讨论什么是串行化，为什么需要在 VB.NET 程序中使用串行化。接着讨论.NET 中的两个主要串行化机制——浅串行化和深串行化，之后概述在默认情况下可以串行化的格式。最后介绍.NET 中串行化的一般使用场合。

- 第 2 章：基本.NET 串行化

第 1 章概述了.NET 串行化，接着就要在本章开始使用.NET 串行化技术了。本章将介绍如何把一个类浅串行化为 XML 格式，和如何把类深串行化为二进制或 SOAP 格式。然后比较这两个串行化格式，找出串行化对象的最佳方式。

- 第 3 章：设计用于浅串行化的类

第 2 章介绍了如何以一种简单、默认的方式使用.NET 串行化，实际上在许多情况下这些默认的机制并不是很好。例如，如果需要串行化为与默认的 XML、SOAP 或二进制格式不同的格式，该怎么办？本章就介绍定制默认串行化机制的方式，讨论如何给可串行化的类添加控制浅串行化格式的属性。

- 第 4 章：设计用于深串行化的类

在讨论完在可串行化的类中定制浅串行化后，就开始论述如何定制深串行化。这么做的一个原因是可以确保不使用反串行化创建几个单一的实例。

- 第 5 章：高级浅串行化技术

有时需要定制串行化，但不能访问可串行化类的源代码，所以不能进行必要的修改。或者可以访问源代码，但不能修改它们，因为该类由其他应用程序使用。在这种情况下，就需要一种方式在类的外部控制和定制串行化，即在串行化代码中进行控制和定制。本章就讨论如何对浅串行化进行这种处理。

- 第 6 章：高级深串行化技术

本章接着介绍如何在可串行化类的外部对深串行化进行定制。讨论这种技术如何帮助我们把对象旧版本中的数据反串行化为新版本，或创建一个串行化格式化器，把数据串行化为一种与 XML、SOAP 或二进制格式完全不同的格式。

- 第 7 章：安全地使用串行化

本章介绍在使用.NET 串行化机制时应理解的安全问题。这包括如何串行化和存储敏感的数据，如何加密已串行化的数据，以及如何阻止有恶意的用户误用串行化/反串行化代码。

目　　录

第 1 章　.NET 串行化简介 …… 1
1.1　什么是串行化 …… 1
1.1.1　为什么串行化非常重要 …… 3
1.1.2　串行化的基本方法 …… 3
1.2　.NET 串行化概述 …… 6
1.2.1　.NET 串行化的类型 …… 6
1.2.2　.NET 串行化的优点 …… 10
1.3　何时使用串行化 …… 13
1.3.1　状态的保存 …… 13
1.3.2　在应用程序间交换数据 …… 15
1.4　接下来要学习的内容 …… 18
1.5　小结 …… 18

第 2 章　基本.NET 串行化 …… 20
2.1　使用浅串行化 …… 20
2.1.1　System.Xml.Serialization 命名空间 …… 21
2.1.2　使用 XmlSerializer …… 21
2.1.3　对对象进行 XML 串行化 …… 36
2.1.4　使用 XML 串行化的场合 …… 44
2.1.5　使用 SOAP 编码和 XmlSerializer …… 44
2.2　深串行化 …… 46
2.2.1　System.Runtime.Serialization 命名空间 …… 47
2.2.2　使用格式化器 …… 47
2.2.3　控制基于格式化器的串行化 …… 55
2.2.4　对象图的深串行化 …… 59
2.2.5　通过深串行化克隆对象 …… 64
2.3　小结 …… 65

第 3 章　设计用于浅串行化的类 …… 66
3.1　从可串行化的类中定制 XML 串行化 …… 66
3.1.1　实现基本的学生应用程序 …… 67

3.1.2 使用 XML 属性定制串行化 …… 77
3.2 把 XML 串行化定制为 SOAP 编码格式 …… 94
3.2.1 把 Student 应用程序串行化为 SOAP 编码的 XML 格式 …… 94
3.2.2 使用 SOAP 属性定制串行化为 SOAP 编码格式的方法 …… 99
3.3 小结 …… 105

第 4 章 设计用于深串行化的类 …… 106
4.1 使用 ISerializable 接口定制格式化的串行化 …… 106
4.2 使用帮助类控制反串行化 …… 122
4.2.1 串行化和反串行化已有的.NET 单一类型 …… 123
4.2.2 串行化和反串行化定制的单一类型 …… 127
4.3 正确反串行化复杂的对象图 …… 143
4.4 小结 …… 150

第 5 章 高级浅串行化技术 …… 151
5.1 在 XML 反串行化过程中处理未知的内容 …… 151
5.1.1 忽略未知的内容 …… 152
5.1.2 检测未知的内容 …… 158
5.1.3 处理未知的内容 …… 165
5.2 重写默认的 XML 串行化 …… 175
5.2.1 重写明文编码的 XML 串行化 …… 176
5.2.2 重写 SOAP 编码的 XML 串行化 …… 191
5.3 小结 …… 199

第 6 章 高级深串行化技术 …… 201
6.1 使用代理使类可串行化 …… 201
6.1.1 使用代理串行化公共状态 …… 202
6.1.2 使用带有反射的代理 …… 210
6.1.3 在代理选择器中组合代理 …… 214
6.1.4 把代理选择器串联到一起 …… 220
6.1.5 高效地使用代理类 …… 230
6.2 在反串行化时更改类型或程序集 …… 231
6.3 创建定制的格式化器 …… 240
6.4 小结 …… 245

第 7 章 安全地使用.NET 串行化 …… 246
7.1 串行化安全概述 …… 246
7.1.1 串行化的安全问题 …… 246
7.1.2 本章的主要内容 …… 248
7.2 什么是代码安全性 …… 248
7.2.1 安全性的类型 …… 249
7.2.2 .NET Framework 中的代码安全性 …… 250
7.2.3 引诱攻击 …… 252
7.2.4 .NET 代码安全系统概述 …… 258
7.3 代码安全性和串行化 …… 258
7.3.1 串行化是一个级别很高的特权 …… 259
7.3.2 重写安全系统 …… 262
7.4 安全地存储串行化状态 …… 264
7.4.1 文件存储和部分信任 …… 264
7.4.2 使用孤立的存储器 …… 268
7.5 加密和串行化 …… 271
7.5.1 加密概述 …… 272
7.5.2 实现加密的串行化 …… 275
7.5.3 加密和串行化技术展望 …… 278
7.6 小结 …… 279

第1章 .NET串行化简介

本书将回答有关串行化的如下几个问题：

- 什么是串行化？
- VB.NET 开发人员为什么要使用串行化？
- 如何在 VB.NET 程序中高效地控制和使用串行化？

本章将回答上面的第一个和第二个问题，概述.NET 串行化机制，讨论在 VB.NET 应用程序中使用串行化的场合。本章的讨论将有助于后续章节对最后一个问题的深入探讨。

1.1 什么是串行化

下面讲述串行化的含义。在应用程序中，对象通常表示一个业务实体，它包含与该业务实体相关的数据(对象的状态)和功能(通过方法实现)。在串行化一个对象时，就是把该对象的所有数据成员都转换为一个数据流，以便在应用程序的非运行期间存储或传输。

说明：

串行化就是把对象状态转换为一个字节流的过程。

对串行化的另一种解释是，串行化提供了创建对象状态快照的方式，这样就可以在应用程序运行期间之外的阶段存储它了。

例如，下面是一个在 VB.NET 中定义的业务对象：

```
Public Class Employee
  Private _EmpNumber As Integer
  Private _EmpName As String
  Private _EmpAddress As Address

  Public Property EmpNumber() As Integer
    Get
      Return _EmpNumber
    End Get
```

```
    Set (ByVal Value As Integer)
      _EmpNumber = value
    End Set
  End Property

  Public Property EmpNumber() As String
    Get
      Return _EmpName
    End Get
    Set (ByVal Value As String)
      _EmpName = value
    End Set
  End Property

  Public Property EmpAddress() As Address
    Get
      Return _EmpAddress
    End Get
    Set (ByVal Value As Address)
      _EmpAddress = value
    End Set
  End Property

  ...

  Public Sub Work()
    ' Work Method Implementation
  End Sub

End Class
```

可以看出，这个 Employee 类有三个公共数据成员和一个方法 Work()，在创建 Employee 的一个实例时，Microsoft .NET CLR 会提取对象类型的定义，把它加载到托管的内存中。在串行化这个对象时，要提取所有的数据成员，把它们输入一个字节流中。

1.1.1 为什么串行化非常重要

这些都很不错，但为什么要做这些工作？有几个原因。第一，大多数应用程序都需要以某种方式在其非运行期间存储有用的数据，使之在应用程序关闭，或执行一个新任务时不会丢失。毕竟，大多数业务应用程序都是以通常的方式处理数据，其结果常常会得到更多的数据，这些数据要么显示给用户，要么存储起来以备后用，要么传送给另一个应用程序。串行化提供了在非运行期间移动对象状态的机制。把这些状态转换为字节流，就可以存储到一个永久性介质上(例如文件或数据库)，另外，字节流也比较容易传送给其他应用程序。

第二，对于面向对象的应用程序，需要能以保留原对象的方式在非运行期间传送对象的状态。换言之，对象中数据之间的关系在串行化形式中应保持不变，这样才能确定哪些数据属于哪个对象。这也意味着，可以在以后反串行化数据——可以使用已串行化的数据，重新创建具有初始状态的初始对象。这是在应用程序的非运行期间简单地移动单个数据值与串行化的重要区别：串行化把对象状态作为一个单元，而不是单个的值。因此，串行化在面向对象编程中是一个非常重要的过程。

在本章的后面将讨论串行化的一般使用场合，如上所述，这些使用场合大致分为两类：

- 存储对象的表达方式，以便日后让用户或另一个应用程序提取对象的状态，或让应用程序反串行化该对象。
- 以串行化的格式传输给远程的应用程序，让远程应用程序使用对象的状态，或重新创建该对象。

但是，在深入探讨这个主题之前，先看看.NET 内置串行化机制的主要功能，学习这些内置机制为开发人员提供的好处。

1.1.2 串行化的基本方法

在介绍.NET 内置的串行化机制之前，先看看在开发平台还没有把这种机制作为它的标准功能之前，该如何串行化对象。这将有助于我们理解为什么 Microsoft .NET 把串行化看得那么重要，并提出了一种可高度定制、但非常易用的标准机制。

完成串行化任务的最简单方式是创建一个例程，把对象的一个实例作为参数，一次提取该实例的一个数据成员，把它们写入一个数据流中。但这种方式非常麻烦，必须为每个对象创建一个例程，除非使用其他像.NET Reflection 这样的技术，动态检测对象的所有数据成员和它们的值。另一个缺点是，这种方式要一个接一个地处理数据成员，如果在这个过程中出现了问题，就会得到一个无效的数据流。而串行化允许开发人员

把对象的所有数据成员作为整体来处理，而不是分别单独处理。

为了解决这个问题，可以在每个对象上实现一个特殊的方法，以连接字符串的形式返回所有的对象状态。例如，下面的代码就定义了一个对象，并实现了一个方法，以连接字符串的形式返回实例数据：

```
Public Class BankAccount
  Public AccountNumber As Integer
  Public AccountTitle As String
  Public Balance As Double
  ' Class implementation...

  Public Function GetStringData() As String
    Return AccountNumber.ToString() + ", " + AccountTitle + ", " + _
        Balance.ToString()
  End Function
End Class
```

注意：

上面的代码把类的所有数据成员都声明为公共字段，而不是通过公共属性来访问的私有字段。在面向对象编程中，以这种方式直接访问对象状态并不好。另外，上面的代码省略了异常的处理。本章将采用这种方式缩短代码，使之易读懂，以便集中精力来介绍与串行化相关的代码。

这段代码允许“串行化”对象的状态。但还需要考虑如何从已串行化的状态中“反串行化”对象。

为此，可以在类上实现一个定制的构造函数，提取这个连接字符串，把所有的数据字段填充回其原来的值。让我们看看下面的代码：

```
Public Class BankAccount
  Public AccountNumber As Integer
  Public AccountTitle As String
  Public Balance As Double
  ' Class implementation...
  Public Sub New(ByVal strState As String)
    Dim arrData() As String =  strState.Split(",")
    AccountNumber = CType(arrData(0), Integer)
    AccountTitle = arrData(1)
    Balance = CType(arrData(2), Double)
  End Sub
```

```
    Public Function GetStringData() As String
        Return AccountNumber.ToString() + ", " + AccountTitle + ", " + _
            Balance.ToString()
    End Function
End Class
```

上面突出显示的代码是一个定制的构造函数，它提取用逗号分隔开的字符串，并设置所有的数据成员。

下面看看如何使用前面的方式串行化和反串行化 BankAccount 实例：

```
Public Class MyApp
    ...
    Public Sub SerializeAccount()
        Dim myAccount As BankAccount = New BankAccount()
        ...
        Dim strState As String = myAccount.GetStringData()
        ' Save this string to a persistent storage
    End Sub

    Public Sub DeSerializeAccount()
        ' Get data from persistent storage into strState
        Dim myAccount As BankAccount = New BankAccount(strState)
        ' Your account is now deserialized
    End Sub
    ...
End Class
```

从这段代码中可以看出这种串行化方法的优点。尽管这种方法很简单，但也有几个潜在的问题：

首先，开发人员必须为每个需要在应用程序中串行化的类添加定制的串行化代码。如果只有几个对象要编写定制的串行化代码，那还好；如果要为复杂应用程序中的每个需要串行化的对象编写这些代码，就非常麻烦了。

第二，串行化对象数据的原因常常是要使这些数据方便地传送到另一个应用程序中，以在该应用程序使用。在前面的例子中，我们需要确保使用这些串行化数据的应用程序的开发人员，能理解我们所使用的串行化格式，否则串行化的数据就是无意义的。这还会引发另一个问题：串行化数据的格式是在要串行化的类的方法中定义的，所以不修改该类的源代码，重新编译，再重新发布应用程序，是很难改变这种格式的。

这种方法的另一个重要问题是，它是基于字符串的，如果在不同环境的网络中传送

这个字符串，还会遇到编码问题。这种方式还涉及到其他一些问题。

因此，Microsoft 提出了一种标准方式在.NET Framework 中进行串行化。.NET 串行化为解决这些问题提供了一种标准的、顺畅的机制。它提供了一组处理大多数串行化问题的类，还为串行化采用了一种通用的模式，使之具有相当的可扩展性。下面就开始探讨.NET 的串行化机制。

1.2 .NET 串行化概述

串行化是开发框架的一个组成部分。.NET Framework 也不例外，它为串行化提供了优异的支持。这种机制包含一组功能丰富的库，可支持不同格式的串行化。其功能全面的串行化机制是高度可定制和可扩展的，也很容易使用和采纳。.NET 在可编程性和数据格式方面解决了串行化问题。

.NET Framework 提供了一种独特而统一的方式来串行化应用程序中的对象。这缩短了业务组件的开发时间，降低了应用程序在开发期间的出错机率，提高了可维护性，因为在整个应用程序中采用标准的方式完成任务，会得到易于维护的基本代码。这样就可以使用标准的框架，以自己喜欢的方式定制对象，更好地维护应用程序。

在设计串行化时，Microsoft 花了很多精力来考虑串行化的大多数使用场合，因此允许用户在串行化后采用许多数据格式。

本节介绍.NET 内置串行化机制的强大功能和灵活性，并与其他常见技术所使用的串行化机制进行比较。

1.2.1 .NET 串行化的类型

在.NET(和其他面向对象的平台)上，有两种不同的串行化机制：浅串行化和深串行化。选择哪种串行化机制来串行化类取决于使用串行化的场合。

1. 浅串行化

在浅串行化中，只有对象公共接口中的数据(公共可读写字段或属性的值)才能串行化。例如，在 BankAccount 例子的改写版本中，公共字段 AccountName 的值可以串行化，私有字段_AccountNumber 的值也能串行化，因为该字段可通过 AccountNumber 属性来访问。但是，私有字段_PIN 的值不能通过属性来访问，所以不能串行化。

```
Public Class BankAccount
  Private _AccountNumber As Integer
  Private _PIN As Integer
```

```
    Public AccountName As String

    Public Property AccountNumber() As Integer
      Get
        Return _AccountNumber
      End Get
      Set (ByVal Value As Integer)
        _AccountNumber = value
      End Set
    End Property
  End Class
```

在反串行化过程中，通过设置公共数据成员(公共字段和公共属性)的值，就可以重新创建对象。但是，如果对没有任何公共状态的对象进行浅串行化，浅反串行化就不能把对象重新创建为其串行化之前的原始状态。所以，在 BankAccount 例子中，由于私有数据成员_PIN 的值不能在浅串行化过程中进行串行化，如果反串行化该对象，就不能保证这个私有字段的值与串行化之前的值相同。

.NET 默认的浅串行化机制将把串行化的数据格式化为 XML。

注意：

本书将进一步讨论 XML，包括 SOAP 和 XML 模式的问题。如果读者需要更多地了解 XML，可参阅 Wrox 出版社出版的《Professional XML》一书(ISBN 1-86100-505-9)。

下面就开始深入讨论 XML 串行化器。

(1) 在.NET 中浅串行化为 XML 格式

.NET 的 XML 串行化器提供了一种快捷而方便的方式来浅串行化对象的状态。它把对象的状态串行化为 XML 格式的流，然后就可以使用这个流完成希望完成的任务，例如把它们保存为 XML 文件，或写入一个网络套接字。公共对象状态保存到一个格式完备的 XML 文档中，如果要把串行化的数据传送给另一个应用程序或用户，该 XML 文档就非常有用。XML 的自我描述本质意味着，把数据保存为标准的 XML 格式通常便于其他程序理解。XML 串行化以 XSD 有效格式生成 XML 数据，并把大多数 CLR 类型映射为 XSD 类型。因此，生成的 XML 建立在 XML 类型的基础之上，而不是与平台相关的类型，并可以根据 XML 模式定义进行有效性检查。所以，XML 串行化非常适合于不同类型的应用程序集成和交互操作，例如 Web 服务。在调用 Web 服务时，XML 串行化也可以用于编组所有的参数和返回值。这也提供了一种根据标准来与各公司建立连接的简单方式。

设计跨多网络的应用程序时，一个主要问题是安全而可靠地在防火墙之间传送数据。RPC 连接在防火墙的问题上臭名昭著。而使用 XML 串行化，就很容易解决这个问

题，安全地在防火墙之间来回传送数据，而不会有大的问题。

XML 串行化也允许控制生成数据的编码。可以把对象串行化为任何可在平台上使用的标准编码。使用 XML 串行化，可以把对象数据编码为 Unicode、UTF8 等，使应用程序能处理多种语言。

XML 实质上是一种基于文本的数据格式化技术，很容易读取和理解。这对于调试应用程序非常有帮助，使用适当的工具查看 XML 数据，能很容易在串行化过程中看出幕后发生了什么。

而其缺点是，XML 串行化器只能执行浅串行化，如果要在实现交互操作或集成时使用它，就需要确保所有需要传送的对象状态都能公开访问。而且，为了使属性可以串行化为 XML，还必须把该属性设置成可读写的。也就是说，不能串行化一个只读的公共属性。但是，浅串行化非常适合于 Web 服务，因为 Web 服务主要用于只公开访问在数据交换过程中涉及到的状态。

由于 XML 串行化只能执行浅串行化，所以不适合于把状态传送给应用程序，在该应用程序中创建对象的副本(但只要所有的对象状态都是公共的，就可以这么做)。

作为一种面向 XSD 类型的机制，XML 串行化试图把 CLR 类型映射为 XSD 类型。但是，它并不总是能把所有的 CLR 类型正确地转换为 XSD 类型，或把 XSD 类型转换为 CLR 类型。例如，CLR 类型 System.Int64 通常映射为整数 XSD 类型。但在反串行化过程中，XSD 类型 integer、long、nonPositiveInteger 和 unsignedLong 都映射为 System.Int64。

可是，.NET 中使用 XML 串行化器的浅串行化还有另一个优点：与稍后讨论的.NET 深串行化不同，要串行化的对象不需要在源代码中标记为可串行化。使用 XML 串行化器就可以浅串行化任何类，但不能处理包含循环引用(两个对象互相引用)的对象的串行化(对象有其成员，这些成员又有它们自己的对象)。这是因为在 XSD 类型格式中，不能描述循环类型引用。为了实现这种类型的串行化，通常使用.NET 的深串行化机制，它能处理循环对象的引用。

最后，如第 3 章和第 5 章所述，XML 串行化也是可以定制的。可以使用属性来控制 XML 文档的格式化。例如，可以控制 XML 根元素、XML 节点、命名空间和 XML 属性。还可以定制 XML 串行化，使之以 SOAP Section 5 编码格式生成串行化数据，该格式可以表示循环引用的对象，但需要许多定制编码。

2. 深串行化

在深串行化中，对象的所有状态(不包括字段访问修饰符)都串行化为一个流(这是使用.NET 反射技术获得的)。如果串行化 BankAccount 类型的对象(在上一节定义)，串行化的数据就会包含_PIN、_Account 和 AccountName 数据成员的值，即包含对象所有字段的值。

深串行化机制可以确保串行化所有的对象状态，所以也可以确保在数据上进行深反串行化时，会重新创建与原来一模一样的对象。如上一节所述，这是深串行化比浅串行化优越的一个重要方面。前面还提到深串行化的另一个优点：它还可以串行化包含循环引用的对象。

.NET提供了对深串行化为SOAP或二进制格式的优异支持，还可以定制这些串行化格式，甚至可以创建新的格式化器，串行化为一个不同的格式。但是，为了直接使用这些默认或定制的深串行化，要串行化的类必须用一个特殊的属性在源代码中进行标记(实际上并不是这样，因为我们可以使用一些技术来深串行化未标记为串行化的类，详见本书后面的内容。但是，这些技术会使串行化代码非常复杂)。

还要注意，在.NET中对类进行深串行化时，不仅会串行化对象的状态，还会在得到的串行化数据中添加程序集信息、类名和其他相关的信息，深串行化机制将使用这些信息正确地反串行化对象。

(1) 在.NET中深串行化为二进制格式

与XML和SOAP串行化相比，二进制格式的数据要紧凑得多，因为它们不需要把所有的数据成员封装到标记对中。这种紧凑的串行化数据使二进制串行化成为一种快速、高效的串行化机制，特别是在网络上传送串行化的状态时，更是如此。

当需要在同一个平台上的应用程序之间传送对象时，二进制串行化非常有效。这有几个原因：首先，显然我们需要使用深串行化，来确保在反串行化阶段正确地重新创建对象。第二，两个应用程序运行在一个平台上，就可以确保它们都使用相同的二进制格式标准，而不同的平台常常使用不同的二进制格式。如果知道两个应用程序使用相同的二进制格式，通常使用二进制串行化为SOAP格式要比使用深串行化更好，因为该格式的紧凑性可提高数据交换的速度，因此二进制串行化非常适合于.NET Remoting。

注意：

要学习更多有关使用VB.NET进行.NET远程通信的知识，可参阅清华大学出版社引进并出版的《VB.NET Remoting技术手册》一书。

从二进制串行化中生成的对象串行化数据建立在.NET CLR类型的基础之上。也就是说，数据使用.NET类型，例如int或double来格式化。这将易于把数据反串行化为合适的数据类型。但是，在其他平台上使用.NET二进制数据会比较困难，除非在这些平台上使用正确的类型映射。这就限制了串行化二进制数据的可移植性，使之很难用于Web服务。因为Web服务常常交换SOAP消息，所以使用XML或SOAP格式化器要比二进制更合适。

另一个缺点是由于二进制数据的二进制本质，串行化的对象实例数据很难读懂(与XML不同)，因此调试应用程序也比较困难，需要一个二进制数据查看器如HexEdit来