



# **.NET 框架程序开发指南**

(下册)

张志学 等 编著

清华大学出版社

(京)新登字 158 号

## 内 容 简 介

.NET 框架是用于构建、配置、运行 Web 服务和应用程序的多语言环境,本书结合大量实例详细介绍了与.NET 框架应用程序开发相关的知识。全书共 7 章,主要内容包括:数据库操控技术、GDI+编程技术、异步调用、访问 Internet、窗体设计技术、使用控件以及使用 WMI 管理应用程序等。

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

版权所有,翻印必究。

本书封面贴有清华大学出版社激光防伪标签,无标签者不得销售。

## 图书在版编目(CIP)数据

.NET 框架程序开发指南.下册/张志学等编著.—北京:清华大学出版社,2000.8  
ISBN 7-302-05652-8

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

中国版本图书馆 CIP 数据核字(2002)第 049709 号

出 版 者:清华大学出版社(北京清华大学学研大厦,邮编 100084)

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

责任编辑:胡先福

印 刷 者:北京市清华园胶印厂

发 行 者:新华书店总店北京发行所

开 本:787×1092 1/16 印张:26 字数:646 千字

版 次:2002 年 8 月第 1 版 2002 年 8 月第 1 次印刷

书 号:ISBN 7-302-05652-8/TP·3333

印 数:0001~5000

定 价:38.00 元

# 前 言

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

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

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

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

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

作 者

2002 年 6 月

# 目 录

第 1 章 数据库操控技术.....	1
1.1 使用数据集.....	1
1.1.1 创建数据集.....	1
1.1.2 添加数据表.....	3
1.1.3 添加关系.....	4
1.1.4 导航数据表.....	4
1.1.5 合并数据集内容.....	6
1.1.6 拷贝数据集内容.....	9
1.1.7 响应数据集事件.....	10
1.2 操作数据表.....	10
1.2.1 创建表.....	11
1.2.2 定义表的纲要.....	11
1.2.3 浏览表数据.....	15
1.2.4 添加表数据.....	17
1.2.5 编辑表数据.....	17
1.2.6 管理行状态和行版本.....	19
1.2.7 删除表数据.....	20
1.2.8 处理行错误信息.....	21
1.2.9 处理行修改.....	22
1.2.10 响应数据表事件.....	22
1.3 操作数据视图.....	23
1.3.1 创建数据视图.....	24
1.3.2 排序和筛选数据.....	24
1.3.3 浏览数据.....	25
1.3.4 查询数据.....	26
1.3.5 导航关系.....	27
1.3.6 修改数据.....	28
1.3.7 响应数据视图事件.....	29
1.3.8 使用数据视图管理器.....	29
1.4 常规 ADO.NET 编程技术.....	31
1.4.1 获取 Identity 或 Autonumber.....	31
1.4.2 开放式并行处理.....	34
1.4.3 在 XML Web 服务中使用数据集.....	37
1.4.4 查询结果分页.....	42

1.4.5	实现.NET 数据提供者 .....	47
1.4.6	确保代码访问安全性 .....	65
1.4.7	访问 ADO 数据 .....	67
	本章小结 .....	69
<b>第 2 章</b>	<b>GDI+编程技术 .....</b>	<b>70</b>
2.1	GDI+概述 .....	70
2.1.1	GDI+的新特性 .....	70
2.1.2	编程模型的修改 .....	72
2.1.3	GDI+的服务 .....	75
2.2	GDI+编程基础 .....	76
2.2.1	绘制直线 .....	76
2.2.2	绘制字符串 .....	77
2.2.3	创建 Graphics 对象 .....	78
2.3	绘线 .....	79
2.3.1	矢量图概述 .....	79
2.3.2	绘制直线和矩形 .....	80
2.3.3	构造和绘制曲线 .....	85
2.3.4	抗混叠 (保真) 绘线 .....	90
2.4	创建和填充路径 .....	90
2.4.1	路径概述 .....	90
2.4.2	创建路径 .....	91
2.4.3	填充路径 .....	92
2.4.4	压平路径 .....	94
2.5	使用区域 .....	94
2.5.1	区域概述 .....	94
2.5.2	击中测试 .....	95
2.5.3	执行裁剪 .....	95
2.6	填充形状 .....	97
2.6.1	填充概述 .....	97
2.6.2	纯色填充 .....	98
2.6.3	阴影填充 .....	98
2.6.4	纹理填充 .....	98
2.6.5	图像填充 .....	99
2.6.6	梯度填充 .....	100
2.7	变换 .....	108
2.7.1	坐标系和坐标变换 .....	108
2.7.2	变换矩阵 .....	110
2.7.3	复合变换 .....	113
2.7.4	全局和局部变换 .....	114

---

2.7.5 图形容器 .....	116
2.8 使用文本和字体 .....	120
2.8.1 构造字体族和字体 .....	120
2.8.2 绘制文本 .....	120
2.8.3 格式化文本 .....	121
2.8.4 枚举已安装的字体 .....	123
2.8.5 创建私有字体集合 .....	124
2.8.6 获取字体规格 .....	129
2.8.7 文本抗混叠 .....	131
2.9 处理不同类型的图像 .....	131
2.9.1 载入和显示位图 .....	132
2.9.2 载入和显示图元文件 .....	134
2.9.3 绘制、定位和克隆图像 .....	134
2.9.4 裁剪和缩放图像 .....	135
2.9.5 控制图像缩放质量 .....	136
2.9.6 旋转、反射和弯曲图像 .....	138
2.9.7 创建缩略图 .....	139
2.9.8 修改图像颜色 .....	139
2.9.9 访问图像元数据 .....	147
2.10 $\alpha$ 混合和填充 .....	149
2.10.1 绘制不透明和半透明线 .....	150
2.10.2 使用不透明和半透明画刷 .....	150
2.10.3 控制 $\alpha$ 混合 .....	151
2.10.4 设置图像 $\alpha$ 值 .....	152
本章小结 .....	154
<b>第3章 异步调用 .....</b>	<b>155</b>
3.1 异步编程概述 .....	155
3.1.1 异步调用方式 .....	155
3.1.2 取消异步操作 .....	156
3.2 异步编程设计模式 .....	156
3.2.1 异步设计模式概述 .....	156
3.2.2 异步方法的数字签名 .....	159
3.2.3 IAsyncResult 接口 .....	159
3.2.4 异步操作的异步回调 Delegate .....	160
3.3 异步 Delegate 编程 .....	160
3.3.1 使用异步 Delegate .....	160
3.3.2 编译器和公用语言运行库支持 .....	163
3.3.3 异步 Delegate 编程示例 .....	163
3.4 多语言源码生成和编译 .....	166

3.4.1	使用 CodeDOM.....	166
3.4.2	构造 CodeDOM 图.....	167
3.4.3	根据 CodeDOM 图生成源代码和编译程序.....	168
	本章小结.....	169
<b>第 4 章</b>	<b>访问 Internet .....</b>	<b>170</b>
4.1	可插式协议.....	170
4.1.1	Internet 应用程序.....	170
4.1.2	资源标识.....	170
4.1.3	.NET 框架中的请求/响应.....	171
4.1.4	WebClient.....	172
4.1.5	可插式协议编程.....	172
4.1.6	System.Net 类的使用建议.....	175
4.2	请求数据.....	176
4.2.1	数据请求概述.....	176
4.2.2	创建 Internet 请求.....	177
4.2.3	使用 Internet 请求和响应类.....	177
4.2.4	在网络中使用流.....	181
4.2.5	执行异步请求.....	182
4.2.6	处理错误.....	186
4.3	使用 HTTP 服务.....	188
4.3.1	HttpWebRequest 和 HttpWebResponse.....	188
4.3.2	管理 HTTP 连接.....	189
4.3.3	使用连接组.....	190
4.3.4	通过代理访问 Internet.....	190
4.4	使用 TCP 服务.....	191
4.4.1	设计 TCP 客户.....	191
4.4.2	设计 TCP 服务器.....	192
4.5	使用 UDP 服务.....	194
4.5.1	广播概述.....	194
4.5.2	发送广播.....	195
4.5.3	接收广播.....	196
4.6	套接字编程.....	197
4.6.1	System.Net.Sockets 名称空间.....	197
4.6.2	创建套接字.....	218
4.6.3	使用客户套接字.....	219
4.6.4	使用服务器(监听)套接字.....	226
4.7	Internet 安全.....	241
4.7.1	使用安全套接字层.....	241
4.7.2	Internet 认证.....	241

---

4.7.3 Web 和套接字许可 .....	242
本章小结 .....	243
<b>第 5 章 窗体设计技术 .....</b>	<b>244</b>
5.1 窗体概述 .....	244
5.1.1 Windows 窗体 .....	244
5.1.2 Web 窗体 .....	245
5.1.3 Windows 窗体和 Web 窗体的比较 .....	245
5.2 创建和使用窗体 .....	246
5.2.1 窗体编程基础 .....	247
5.2.2 创建窗体 .....	264
5.2.3 模态和非模态窗体 .....	265
5.2.4 对话框 .....	266
5.3 为窗体添加菜单 .....	271
5.3.1 基础菜单管理类 .....	271
5.3.2 标准菜单管理类 .....	273
5.3.3 菜单项管理类 .....	274
5.3.4 使用标准菜单 .....	280
5.3.5 使用快捷菜单 .....	285
5.4 为窗体添加工具栏 .....	289
5.4.1 工具栏管理类 .....	289
5.4.2 工具栏按钮管理类 .....	292
5.4.3 使用工具栏 .....	294
5.5 为窗体添加状态栏 .....	296
5.5.1 状态栏管理类 .....	296
5.5.2 状态栏窗格管理类 .....	299
5.5.3 使用状态栏 .....	300
本章小结 .....	302
<b>第 6 章 使用控件 .....</b>	<b>303</b>
6.1 控件基础功能支持 .....	303
6.1.1 基础控件属性 .....	304
6.1.2 基础控件方法 .....	309
6.1.3 基础控件事件 .....	325
6.2 标签控件 .....	331
6.2.1 标签管理类 .....	331
6.2.2 使用标签 .....	335
6.3 图片框控件 .....	336
6.3.1 图片框管理类 .....	336

6.3.2 使用图片框 .....	337
6.4 按钮控件 .....	337
6.4.1 按钮管理类 .....	338
6.4.2 使用按钮控件 .....	340
6.5 复选框控件 .....	341
6.5.1 复选框管理类 .....	341
6.5.2 使用复选框控件 .....	343
6.6 单选按钮控件 .....	345
6.6.1 单选按钮管理类 .....	345
6.6.2 使用单选按钮 .....	345
6.7 文本框控件 .....	346
6.7.1 文本框管理类 .....	347
6.7.2 使用文本框 .....	352
6.8 列表框控件 .....	354
6.8.1 列表框管理类 .....	354
6.8.2 使用列表框 .....	362
6.9 组合框控件 .....	365
6.9.1 组合框管理类 .....	365
6.9.2 使用组合框 .....	365
6.10 编组框控件 .....	370
6.10.1 编组框管理类 .....	370
6.10.2 使用编组框 .....	370
6.11 定时器控件 .....	371
6.11.1 定时器管理类 .....	371
6.11.2 使用定时器控件 .....	372
6.12 管理窗体控件集合 .....	374
本章小结 .....	375
<b>第7章 使用 WMI 管理应用程序 .....</b>	<b>376</b>
7.1 WMI 概述 .....	376
7.1.1 WMI 的组成结构 .....	376
7.1.2 纲要 .....	377
7.1.3 查询 .....	377
7.1.4 管理事件 .....	377
7.1.5 WMI 名称空间 .....	378
7.2 访问管理信息 .....	378
7.2.1 获取管理对象集合 .....	379
7.2.2 查询管理信息 .....	381
7.2.3 预订和处理管理事件 .....	382

---

7.2.4	执行管理对象的方法.....	384
7.2.5	远程和连接选项 .....	386
7.2.6	使用强类型对象 .....	387
7.2.7	浏览 WMI 纲要 .....	388
7.3	实现可管理性能 .....	390
7.3.1	可管理性能概述 .....	390
7.3.2	CLI 和 WMI 中的类和映射.....	392
7.3.3	提供管理事件 .....	394
7.3.4	提供管理数据 .....	396
7.3.5	继承 .....	397
7.3.6	注册纲要 .....	403
	本章小结 .....	404

# 第 1 章 数据库操控技术

虽然数据集是独立于数据源的、存在于内存中的数据表示，但是，在使用 .NET 数据提供者时，数据集依然可以与数据源中的现存数据一起使用。 .NET 数据提供者使用数据适配器为数据集填充数据和/或纲要信息，并将对数据集的修改写回到数据源中。也就是说，可以通过对数据集的操作实现操控数据库的目的。

可对数据集表（本书简称为数据表或表）执行的操作与可对数据库表执行的操作相同：可以添加、查看、编辑和删除表中数据；可以监测错误和事件；可以查询表中数据。当修改（数据集表）中的数据时，还可以检查此修改是否合法，并通过代码确定应接受还是拒绝修改。通过数据视图，可以为数据集表中的数据创建多个不同视图。例如，可以为不同视图指定不同的排列顺序，或在不同视图中包含根据不同行状态或筛选表达式选出的数据。这种特性经常由数据绑定应用程序使用。

本章要点：

- ❖ 使用数据集、数据表和数据视图
- ❖ ADO.NET 常用编程技术

## 1.1 使用数据集

ADO.NET 数据集代表驻留于内存中的数据，它提供了独立于数据源的关系编程模型。数据集代表一套完整的数据，包括表、约束和关系。

使用数据集的方法主要包括以下 3 种，这 3 种方法可以独立使用，也可以组合使用：

- 通过程序在数据集中创建数据表、数据关系和约束，并将数据填充到表中。
- 使用数据适配器填充数据集。
- 使用载入和保存数据集内容。

使用 XML Web 服务还能够传输强类型数据集。实际上，数据集的设计特点使得它非常适于通过 XML Web 服务传输。

### 1.1.1 创建数据集

调用 DataSet 的构造函数，可以创建数据集实例。在构造函数中，如果没有指定数据集名称，则新实例的名称将被设置为 NewDataSet。

用户还可以基于现存数据集创建新数据集，此新数据集可以为：现存数据集的拷贝；现存数据集的“克隆”，也就是说只包含现存数据集的纲要，而不包括其中的数据；现存数据集的子集，例如只包含被修改的行。

下面给出创建数据集实例的示例代码：

```
[C#]
```

```
DataSet custDS = new DataSet("CustomerOrders");
```

除了对值的后绑定访问外（通过弱类型化变量），数据集还提供了强类型化数据访问。表和列为数据集的一部分，并通过用户友元名和强类型化变量访问。

类型数据集是派生自 `DataSet` 的类，它继承了数据集的所有方法、事件和属性。此外，类型数据集提供了强类型方法、事件和属性。这意味着可以通过名称访问表和列，而不只是使用基于集合的方法。除了增强的代码可读性外，类型数据集还允许 Visual Studio .NET 代码编辑器自动完成输入行。

强类型数据集还能在编译时以正确的类型访问值。也就是说，类型不匹配错误将在编译而不是运行时被捕捉。

通过由 .NET 框架 SDK 提供的 `XSD.exe` 工具，可以根据由 XSD (XML Schema Definition Language, XML 纲要定义语言) 标准编译的 XML 纲要生成强类型数据集。下面的代码给出了使用 `XSD.exe` 生成数据集的语法：

```
xsd.exe /d /l:C# mySchema.xsd /n:XSDSchema.Namespace
```

在上面的语法形式中，`/d` 指令通知工具生成数据集；`/l` 通知工具应使用何种语言（例如 C# 或 Visual Basic.NET）；可选的 `/n` 指令通知工具还应为数据集生成一个名为 `XSDSchema.Namespace` 的名称空间。上述命令的输出为 `XSDSchemaFileName.cs`，该文件可以被编译并在 ADO.NET 应用程序中使用。生成的代码可以被编译为库或模块。

下面给出使用 C# 编译器（`csc.exe`）将生成的代码编译为库的命令：

```
csc.exe /t:library XSDSchemaFileName.cs /r:System.dll /r:System.Data.dll
```

`/t` 指令告诉工具将文件编译为库；`/r` 指令说明需要编译独立库。命令的输出为 `XSDSchemaFileName.dll`，当使用 `/r` 指令编译 ADO.NET 应用程序时，该输出可被传递给编译器。

下面给出了在 ADO.NET 应用程序中，访问传递给 `XSD.exe` 的名称空间的代码：

```
[C#]
```

```
using XSDSchema.Namespace;
```

下面的示例代码使用名为 `CustomerDataSet` 的类型化数据集从数据库中载入客户。一旦使用 `Fill` 方法载入数据后，示例将使用类型化 `CustomersRow` 对象循环每个客户。这提供了对 `CustomerID` 列的直接访问，而不是通过 `DataCloumnCollection` 访问：

```
[C#]
```

```
CustomerDataSet custDS = new CustomerDataSet();
```

```
SqlDataAdapter custCMD = new SqlDataAdapter("SELECT * FROM Customers", "Data  
Source=localhost;Integrated Security=SSPI;Initial Catalog=northwind");
```

```
custCMD.Fill(custDS, "Customers");
```

```
foreach(CustomerDataSet.CustomersRow custRow in custDS.Customers)  
    Console.WriteLine(custRow.CustomerID);
```

下面给出了示例中使用的 XML 纲要:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="CustomerDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="CustomerDataSet" msdata:IsDataSet="true">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="Customers">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="CustomerID" type="xs:string" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

### 1.1.2 添加数据表

ADO.NET 允许用户创建新数据表, 并将其加入到现存数据集中。通过表列集中的 DataColumn 对象的 PrimaryKey 和 Unique 属性, 可以为数据表设置约束信息。

下面的示例代码构造了一个数据集, 然后向其中添加了一个数据表, 向数据表中添加了 3 个 DataColumn 对象, 最后将其中一列设置为表的主键列:

```
[C#]
DataSet custDS = new DataSet("CustomerOrders");

DataTable ordersTable = custDS.Tables.Add("Orders");

DataColumn pkCol = ordersTable.Columns.Add("OrderID", typeof(Int32));
ordersTable.Columns.Add("OrderQuantity", typeof(Int32));
ordersTable.Columns.Add("CompanyName", typeof(string));

ordersTable.PrimaryKey = new DataColumn[] {pkCol};
```

数据集中的表名和关系名是大小写敏感的。因此, 数据集中的表或关系名可以相同, 只要它们的大小写不完全相同即可。例如, 数据集 ds 中可以同时包含两个表: Table1 和 table1。在这种情况下, 引用表名时大小写也必须完全匹配, 否则就可能产生异常。例如, 如果以 ds.Tables["TABLE1"] 的形式引用表, 则将导致异常产生。

如果数据集中的表或关系名称间不只是存在大小写的不同, 则不会施用大小写敏感性规则。也就是说, 可以以任意形式的拼写引用同一表或关系。例如, 使用

ds.Tables["TABLE1"], 可以引用 Table1。

数据集的 CaseSensitive 属性并不影响上述行为, 而只影响排序、检索、筛选或约束等。也就是说, 使用名称引用表或关系的行为并不会受到 CaseSensitive 属性的影响。

### 1.1.3 添加关系

在包含多个表的数据集中, 可以通过 DataRelation 对象关联表、在表间导航, 或从相关表中返回父/子行。

创建 DataRelation 所需的参数包括关系名和 DataColumn 数组, 该数组中包含关系中的父列和子列。在创建了关系后, 就可以通过它在表间移动和获取值了。

向数据集添加关系时, 也会默认为父表添加一个 UniqueConstraint, 为子表添加一个 ForeignKeyConstraint。

下面的示例代码在数据集的两个表间创建了一个关系:

```
[C#]
custDS.Relations.Add("CustOrders", custDS.Tables["Customers"].Columns["CustID"], custDS.
Tables["Orders"].Columns["CustID"]);
```

如果 DataRelation 的 Nested 属性被设置为 true, 则当使用 WriteXml 方法将表写为 XML 时, 将导致子表行被嵌套于相关父表行中。

### 1.1.4 导航数据表

数据关系的主要功能之一就是在数据集表间导航。当给定了相关表中的单个 DataRow 时, 允许获取所有相关 DataRow 对象。例如, 在建立了订单表和客户表间的关系后, 可以使用 DataRow.GetChildRows 方法, 获取特定客户的所有订单记录。

下面的示例代码在 Orders 和 Customers 表间建立了关系, 并返回每个客户的所有订单:

```
[C#]
DataRelation custOrderRel = custDS.Relations.Add("CustOrders",
    custDS.Tables["Customers"].Columns["CustomerID"],
    custDS.Tables["Orders"].Columns["CustomerID"]);
foreach (DataRow custRow in custDS.Tables["Customers"].Rows)
{
    Console.WriteLine(custRow["CustomerID"]);
    foreach (DataRow orderRow in custRow.GetChildRows(custOrderRel))
        Console.WriteLine(orderRow["OrderID"]);
}
```

下一个示例构建在上一个示例的基础上, 它将 4 个表关联在一起, 并浏览这些关系。与上一个示例一样, CustomerID 将 Customers 和 Orders 表关联起来。对于 Customers 表中的每个客户, 都会确定 Orders 表中的所有相关子行, 以便能返回特定客户的订单数。

本示例还会从 OrderDetails 和 Products 表中返回值。Orders 表通过 OrderId 与 OrderDetail

表相关。OrderDetails 表通过 ProductID 与 Product 表相关，从而能够从 OrderDetails 表中直接返回 ProductName（产品名）。在这个关系中，Products 表是父表，而 OrderDetails 为子表。因此，当枚举 OrderDetails 表时，将调用 GetParentRow 方法，以获取相关的 ProductName 值。

需要注意的是，当为 Customers 和 Orders 表创建 DataRelation 时，没有为 createConstraints 标志指定值（默认为 true）。这意味着 Orders 表中所有行的 CustomerID 值，都存在于父表 Customers 中。如果 Orders 表中的 CustomerID 不存在于 Customers 表中，那么将违反 ForeignKeyConstraint，从而抛出异常。

对于子行中可能包含父行中不存在的值的情况，应在添加关系时将 createConstraints 标志设置为 false。在本示例中，Orders 和 OrderDetails 表间关系的 createConstarints 标志被设置为 false。这允许应用程序返回 OrderDetails 表的所有记录，以及 Orders 表中的一个记录子集，而不会导致异常。本示例的输出如下：

```
Customer ID: ZZXUE
  Order ID: 320
    Order Date: 4/15/2002 12:00:00 AM
      Product: 猪肉罐头
      Quantity: 6
      Product: 鸡肉罐头
      Quantity: 4
      Product: 牛肉罐头
      Quantity: 6
  Order ID: 341
    Order Date: 3/23/2002 12:00:00 AM
      Product: 羊肉罐头
      Quantity: 3
```

示例代码如下：

```
[C#]
DataRelation custOrderRel = custDS.Relations.Add("CustOrders",
    custDS.Tables["Customers"].Columns["CustomerID"],
    custDS.Tables["Orders"].Columns["CustomerID"]);

DataRelation orderDetailRel = custDS.Relations.Add("OrderDetail",
    custDS.Tables["Orders"].Columns["OrderID"],
    custDS.Tables["OrderDetails"].Columns["OrderID"], false);

DataRelation orderProductRel = custDS.Relations.Add("OrderProducts",
    custDS.Tables["Products"].Columns["ProductID"],
    custDS.Tables["OrderDetails"].Columns["ProductID"]);

foreach (DataRow custRow in custDS.Tables["Customers"].Rows)
{
    Console.WriteLine("Customer ID: " + custRow["CustomerID"]);
```

```
foreach (DataRow orderRow in custRow.GetChildRows(custOrderRel))
{
    Console.WriteLine("Order ID: " + orderRow["OrderID"]);
    Console.WriteLine("\tOrder Date: " + orderRow["OrderDate"]);

    foreach (DataRow detailRow in orderRow.GetChildRows(orderDetailRel))
    {
        Console.WriteLine("\t Product " + detailRow.GetParentRow(orderProductRel)["ProductName"]);
        Console.WriteLine("\t Quantity: " + detailRow["Quantity"]);
    }
}
}
```

### 1.1.5 合并数据集内容

使用 `DataSet.Merge` 方法，可以将 `DataSet`、`DataTable` 或 `DataRow` 数组合并到现存数据集中。影响新数据与现存数据集合并的因素和选项如下：

#### 1. 如何处理主键

合并数据集时，如果接受新数据和纲要的表具有主键，则新行将匹配具有相同 `Original` 主键值的现存行。如果新纲要中的列与现存纲要匹配，则现存行中的数据将被修改；与现存纲要不匹配的行或被忽略或被添加，这取决于 `MissingSchemaAction` 参数的设置。与现存行主键不匹配的新行将被追加到现存表中。

如果新行或现存行的状态为 `Added`，则将使用其 `Current` 主键值进行匹配，因为它们没有 `Original` 行状态。如果新表和现存表包含同名但不同类型的列，则将抛出异常，并且将触发数据集的 `MergeFailed` 事件。如果新表和现存表都定义了主键，但是主键列不同，则将抛出异常，并且将触发数据集的 `MergeFailed` 事件。

如果接受新数据的表没有主键，则新行不能匹配现存表中的行，因而将被追加到现存表。

#### 2. 是否保留修改

当将 `DataSet`、`DataTable` 或 `DataRow` 数组传递给 `Merge` 方法时，可以通过可选参数指定是否保留现存数据集中的修改，以及如何处理新数据中的新纲要元素。布尔型标志 `preserveChanges` 用于指定是否保留现存数据集中的修改。如果 `preserveChanges` 标志被设置为 `true`，则新值将不会覆盖现存行的 `Current` 版本值。如果没有指定 `preserveChanges` 标志，则它将被默认设置为 `false`。

当 `preserveChanges` 标志为 `true` 时，现存行的数据将由行的 `Current` 版本维护；而现存行的 `Original` 版本将被新行的 `Original` 版本覆盖；现存行的 `RowState` 被设置为 `Modified`，但也会存在如下例外情况：

- 如果现存行的 `RowState` 为 `Deleted`，则合并后它的状态将依然为 `Deleted`，而不会