第 5 章　Windows CE 6.0 数据库同步应用

本章学习目标

以往应用程序通过合并复制或者 RDA 远程数据访问进行移动设备和服务器之间的数据同步。自微软推出新一代 MicrosoftSynchronization Services for ADO.NET 数据访问技术之后，开发者可以快速方便地将数据从桌面/tablet/移动设备等和服务器端进行高效的数据同步。通过本章的学习，要求掌握以下内容:

- 了解客户端与服务器端之间数据同步的基本知识。
- 掌握 Synchronization Services for ADO.NET 的数据同步技术。
- 掌握 Synchronization Services for ADO.NET 的数据同步环境搭建。
- 掌握 Windows CE 下的数据同步编程方法。

5.1　数据同步技术简介

随着计算机技术和网络技术的快速发展，目前在很多应用场合存在着远程数据访问的需求，比如公司员工外出作业，往往携带 PDA 或者其他手持嵌入式移动设备，这些移动设备由于受处理器的运行速度和内存大小等硬件资源影响，不能处理和储存大量数据，而只能存放少量的有限数据，一旦需要访问远程数据库服务器中的数据时，只能借助于网络方式，以保持设备端和服务器端数据同步。针对这些多目标平台的数据同步应用，嵌入式数据库的数据同步技术可以有三种方式实现：RDA 远程数据访问、Merge Replication 合并复制以及 Synchronization Service for ADO.NET 数据同步。

1．RDA 远程数据访问

Remote Data Access（RDA）远程数据访问支持从远程的 SQL Server 服务器中下载数据到设备端的 SQL Server Compact 3.5 数据库中，然后在本地对数据进行浏览和修改，并将修改结果更新到 SQL Server 服务器中。RDA 使用运行于 IIS 上的 Server Agent 作为 SQL Server 数据库与 SQL Server Compact 数据之间的通信代理。Server Agent 负责监听来自 SQL Server Compact Client Agent 的 HTTP 请求。Server Agent 使用临时消息文件（*.in 和*.out）来管理 SQL Server 与 SQL Server Compact 的交换数据。运行基于 Windows CE 设备上的应用程序可以应用 RDA 完成数据下载、数据上传以及提交 SQL 语句。但 RDA 这种方式存在着一定的局限性，一方面不能处理数据同步过程中出现的数据冲突，另一方面不能进行增量数据同步，所以这种数据同步方式将被逐步淘汰。

2．Merge Replication 合并复制

合并复制在 SQL Server Compact 数据库中是一种面向 DBA 的数据同步方式，它适合嵌入式数据库与远程 SQL Server 数据库之间的数据同步，因为在客户端不需要编写很多代码，只

需在 SQL Server 服务器端进行一些较为复杂的配置操作即可完成数据同步。实现合并复制需要使用 SQL Server Compact 数据库引擎、SQL Server Compact Client Agent（客户端代理）、SQL Server 2005/2008、SQL Server Compact Server Agent（服务器端代理）、SQL Server Compact 复制提供者。当嵌入式设备被连接到网络上时，SQL Server Compact 的服务器代理（Server Agent）运行于 SQL Server 2005/2008 和 IIS 的服务器环境中，它处理来自 SQL Server Compact 客户端代理的 HTTP 请求。SQL Server Compact 客户端代理驻留在基于 Windows CE 的嵌入式设备上，并应用 HTTP 与驻留在 IIS 服务器上的 SQL Server Compact 的服务器代理进行通信，依此通过复制或远程数据访问与 SQL Server 2005/2008 进行通信，一方面能将客户端修改的数据记录发送到远程服务器端，另一方面也能将服务器端修改的数据记录通过同步反映到客户端上，实现客户端和服务器端的数据同步。

3. Synchronization Service for ADO.NET 数据同步

Synchronization Service for ADO.NET 是提供给开发人员使用的一种面向服务功能数据的同步方式，这种同步方式与合并复制同步方式存在一定的差异。合并复制是一种对数据库的数据架构和数据复制，而 Synchronization Service for ADO.NET 是同步数据，它并不要求客户端的数据库结构与服务器端的数据库结构相同，这使得开发人员通过编写代码就可以控制整个数据同步的逻辑，并有助于提高开发人员通过调试方式控制数据的能力。另外它面向 SOA 组件功能，可以使用 Web Service 及 WCF，支持任何 ADO.NET 数据提供程序，这就意味着客户端是 SQL Server Compact 3.5 数据库，而服务器端可以是 SQL Server 2005/2008 或者是 Oracle 数据库等。本章将重点介绍基于 Synchronization Service for ADO.NET 的嵌入式移动设备数据同步技术。

三种数据同步技术的对比如表 5-1 所示。

表 5-1 三种数据同步技术比较

主要对比	RDA	合并复制	Synchornization Services
使用服务进行同步	否	否	是
支持异类数据库	否	否	是
跟踪增量更改	否	是	是
冲突检测与解决	否	是	是
在客户端轻松建立视图	否	否	是
自动初始化架构和数据	是	是	是
支持大型数据集	是	是	是
可在本地使用查询分析器	是	是	是
自动传播架构更改	否	是	否
在设备上使用	是	是	是

5.2 基于 Synchronization Services for ADO.NET 的数据同步技术

目前应用程序正越来越多地应用于移动客户端，如便携式计算机和设备。由于这些移动

客户端与中央服务器没有连贯或可靠的网络连接，因此对于这些应用程序而言，能够在客户端上使用数据的一份本地副本十分重要。同等重要的一点是，在网络连接可用时，需要能够将数据的本地副本与中央服务器同步，Synchronization Services for ADO.NET 是微软推出的同步框架（Microsoft Synchronization Framework）中的一员，也是其重要的组成部分。它可以实现基于数据库的数据智能同步，即从客户端的 SSCE（SQL Server Compact Edition）数据库至服务器端数据库（SQL Server 或其他数据库，如 Oracle 等）间数据同步，另外在一些复杂的开发场景中，开发人员可以利用 Synchronization Services for ADO.NET 开发出适应复杂数据的业务逻辑。如在一个分布式场景中可以创建一个服务器端，然后使用几十或上千台 PC、手机、PDA 等移动设备作为客户端，这样可以使用 Sync Service for ADO.NET 进行数据同步。

在 Synchronization Services for ADO.NET 中包含一组 DLL，这些 DLL 提供了用于在数据服务和本地存储区之间同步数据的组件，而不是仅仅用于复制数据库及其架构。其中 Microsoft.Synchronization.Data.dll 包含同步代理、同步表和同步组；Microsoft.Synchronization.Data.SqlServerCe.dll 包含客户端同步提供程序；Microsoft.Synchronization.Data.Server.dll 包含服务器同步提供程序和同步适配器。

根据 Windows CE 移动设备应用程序的特点和体系结构的要求，Synchronization Services for ADO.NET 支持 N 层体系结构的数据同步机制，以实现 SQL Server Compact 3.5 客户端数据库和服务器数据库或任何其他数据源之间的同步。

如图 5-1 所示是一个 N 层体系结构，这需要一个代理服务器、一个服务以及一个传输机制来实现客户端数据库和服务器数据库之间的通信。对于 N 层应用程序，Microsoft.Synchronization.Data.dll 和 Microsoft.Synchronization.Data.Server.dll 驻留在提供同步服务的一台单独的计算机上。这种体系结构比双层体系结构更为常见，因为 N 层体系结构不需要在客户端数据库和服务器数据库之间建立直接连接。

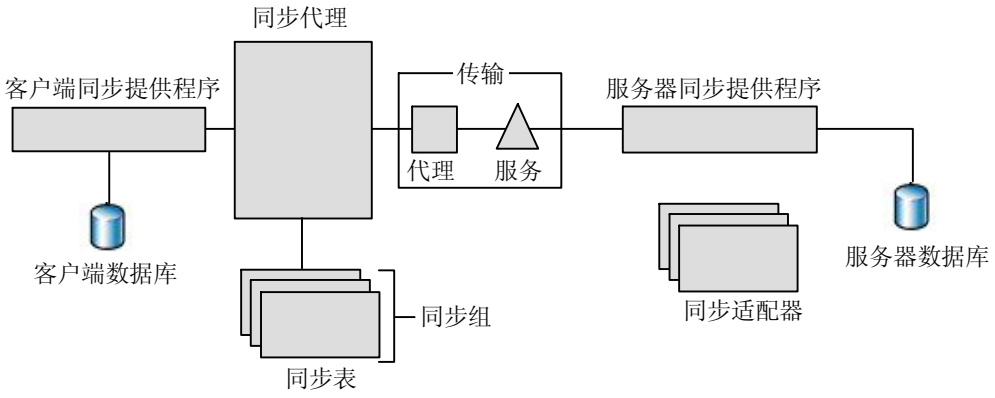

图 5-1　N 层体系结构数据同步

1．客户端数据库

用于 Sync Services 应用程序的客户端数据库使用 SQL Server Compact 3.5 SP1 版本。它内置了对 Sync Services 的支持，包含了 Microsoft.Synchronization.Data 和 Microsoft.Synchronization.Data.SqlServerCe 两个组件。Sync Services 为跟踪客户端数据库中的增量变更提供了一个基础结构。

2. 服务器数据库

服务器数据库可以是能够对其使用 ADO.NET 提供程序的任何数据库。如果希望跟踪服务器数据库中的增量变更，必须使用该数据库以达到此目的。

3. 同步代理

同步代理通过以下方式驱动同步过程：

（1）循环遍历要同步的每个表。

（2）调用客户端同步提供程序以检索和应用客户端数据库的更改。

（3）调用服务器同步提供程序以检索和应用服务器数据库的更改。

4. 客户端同步提供程序

客户端同步提供程序与客户端通信，并将同步代理与客户端数据库的特定实现屏蔽开来。Sync Services 包括一个用于 SQL Server Compact 3.5 SP1 数据库的提供程序。它的主要功能有：

（1）储存与客户端上支持同步的表的有关信息。

（2）检索自上次同步以来在客户端数据库中发生的更改。

（3）将增量更改应用于客户端数据库。

（4）检测发生冲突的更改。

5. 服务器同步提供程序

服务器同步提供程序与服务器通信，并将同步代理与服务器数据库的特定实现屏蔽开来。服务器同步提供程序的主要功能有：

（1）存储与服务器上支持同步的表的相关的信息。

（2）检索自上次同步以来在服务器中发生的更改。

（3）将增量更改应用于服务器数据库。

（4）检测发生冲突的更改。

6. 同步表和同步组

同步表是为每个进行同步的表定义的。它可以设置同步方向。通过设置不同表的同步方向（SyncDirection）可以灵活地决定哪些表只进行下载或只进行上传操作。比如，对于小数据量的并且不需要更新的基础表，可以使用快照（SnapShot）方式每次全部下载更新一遍，而对于那些不断更新的表，可以使用双向同步（Bidirectonal）方式不断地双向更新。

（1）Bidirectional 首次同步期间，客户端通常从服务器下载架构和一个初始数据集。执行后续同步时，客户端将更改上传到服务器，然后从服务器下载更改。

（2）DownloadOnly 首次同步期间，客户端通常从服务器下载架构和一个初始数据集。执行后续同步时，客户端从服务器下载更改。

（3）Snapshot 客户端将从服务器下载一个数据集。每次同步期间，这些数据都将完全刷新。

（4）UploadOnly 首次同步期间，客户端通常从服务器下载架构。执行后续同步时，客户端将更改上传到服务器。

在定义了同步表之后，多个同步表（Sync Table）可以定义到一个同步组（SyncGroup）中。在提交数据时，如果表包括在同步组中，则这些表的变更将作为一个单元进行传送，而且会在一个事务中作为一个单元提交到服务器上，为了保证数据的完整性，主从表结构的表应该定义到同一同步组中。如果组中进行的任何变更失败，则对整个组的变更都会在下一次同步时进行重试。

7. 同步适配器

同步适配器（Sync Adapter）模仿 ADO.NET 中的数据适配器，并为进行同步的每个表均定义同步适配器。同步适配器为服务器同步提供程序提供了与服务器数据库交互所需的特定命令，如 InsertCommand，此命令可从客户端数据库向服务器数据库应用插入。由于同步适配器使用 ADO.NET 的 DbCommand 对象，因此可以使用 ADO.NET 支持的任意命令结构。这包括内联 Transact-SQL、存储过程、视图、函数等。这些命令只需定义要传输和应用的结构以及数据的单个结果。

8. 代理、服务和传输

“代理”、“服务”和“传输”用在 N 层体系结构中。在 N 层应用程序中，使用的是 Microsoft.Synchronization.Data.Server.dll，但是它不驻留在客户端上。通常，DLL 驻留在直接连接到服务器数据库的一个中间层上。在这种情况下，为了实现客户端和中间层之间的通信，代理和服务是必需的。

（1）在客户端上，应用程序代码参考服务器同步提供程序的一个代理（ServerSyncProviderProxy），而不是直接参考提供程序。该代理与位于中间层的服务进行通信。

（2）在中间层，该服务 DbServerSyncProvider 继承 ServerSyncProvider 类并公开与其相同的方法。然后，通过与服务器数据库的直接连接执行服务器同步提供程序的方法，结果通过中间层发送回客户端。

5.3　Synchronization Services for ADO.NET 数据同步环境搭建

1. IIS 组件安装

在 PC 端安装 Windows XP 操作系统时，默认是不安装 IIS 组件的，要安装 IIS 可以按照以下步骤进行：

（1）打开控制面板，双击“添加或删除程序”图标，进入“添加或删除程序”对话框，如图 5-2 所示，单击左边“添加或删除 Windows 组件”图标，“进入 Windows 组件向导”对话框。

图 5-2　“添加或删除程序”对话框

（2）在如图 5-3 所示的“Windows 组件向导”对话框中，如果“Internet 信息服务(IIS)”复选框没有选中，这时选中此项，单击“下一步”按钮，开始进行 IIS 组件的安装。

图 5-3 “Windows 组件向导”对话框

2. SQL Server 2005 安装

本章重点介绍服务器端 SQL Server 2005 数据库与 SQL Server Compact 3.5 数据库进行同步。所以首先要进行 SQL Server 2005 数据库的安装。

（1）当安装文件 SQL.2005.all.chs.iso 是通过虚拟光驱安装时，双击数据库安装盘的虚拟光驱，开始安装，出现如图 5-4 所示的安装开始画面，这里安装的是 SQL Server 2005 Developer Edition 版本，单击“基于 x86 的操作系统”选项。

图 5-4 SQL Server 2005 安装开始向导界面

（2）当安装向导出现如图 5-5 所示的安装界面时，单击“服务器组件、工具、联机丛书和示例”选项。

图 5-5　SQL Server 2005 的安装选项

（3）当安装向导出现“要安装的组件”对话框时，如图 5-6 所示，这里必须选中 SQL Server Database Services 和“工作站组件、联机丛书和开发工具”，其他选项根据实际需要进行选择安装。

图 5-6　选择 SQL Server 2005 要安装的组件

（4）当安装向导出现“服务账户”对话框时，如图 5-7 所示，选中“使用内置系统账户”选项，下方复选中 SQL Server Agent，单击“下一步”按钮。

图 5-7　选择服务账户

（5）当安装向导出现“身份验证模式”对话框时，如图 5-8 所示，选中“Windows 身份验证模式”选项，单击“下一步”按钮。

图 5-8　选择身份验证模式

（6）当 SQL Server 2005 所有组件都安装成功之后，出现如图 5-9 所示的安装向导界面，单击“下一步”按钮。然后在出现的界面上单击“完成”按钮，完成 SQL Server 2005 的安装。

图 5-9　SQL Server 2005 安装成功的界面

3．Visual Studio 2008 SP1 安装

（1）从 SQL Server Compact 3.5 SP1 版本开始，SQL Server Compact 支持 Microsoft Synchronization Services for ADO.NET 同步技术，并同时可用于台式设备和移动设备，SQL Server Compact 3.5 Service Pack 1 (SP1) 随 Visual Studio 2008 SP1 提供，如图 5-10 所示。

图 5-10　Visual Studio 2008 SP1 安装向导界面

（2）当出现如图 5-10 所示的安装向导界面之后，单击“下一步”按钮，开始进入 Visual Studio 2008 SP1 的安装，安装过程时间较长，需要等待一段时间才能完成，如图 5-11 所示。

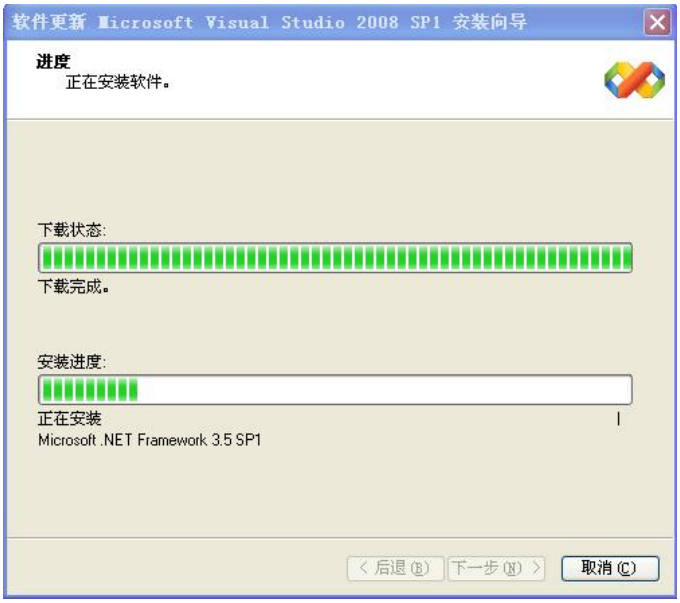

图 5-11　Visual Studio 2008 SP1 的安装过程

4. 用于移动设备的 SQL Server Compact 3.5 SP1 (SSCEDeviceRuntime-CHS.msi)安装

对于在 Visual Studio 2008 中开发 Windows Mobile 和基于 Windows CE 设备的应用程序，并在移动设备上部署这些应用程序，这些运行时组件是必需的。

（1）双击 SSCEDeviceRuntime-CHS.msi 安装程序，进入如图 5-12 所示的安装开始界面，单击“下一步”按钮。

图 5-12　安装开始界面

（2）如图 5-13 所示，进入 SQL Server Compact 3.5 SP1 for Devices 安装过程。

（3）安装完成之后，SQL Server Compact 设备的运行时组件可以在 %ProgramFiles%\Microsoft SQL Server Compact Edition\v3.5\Devices 目录中查找到，如图 5-14 所示。

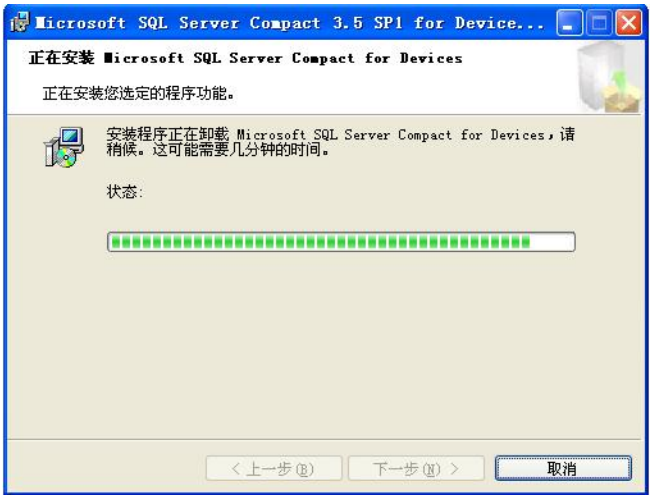

图 5-13　SQL Server Compact 3.5 SP1 for Devices 安装过程

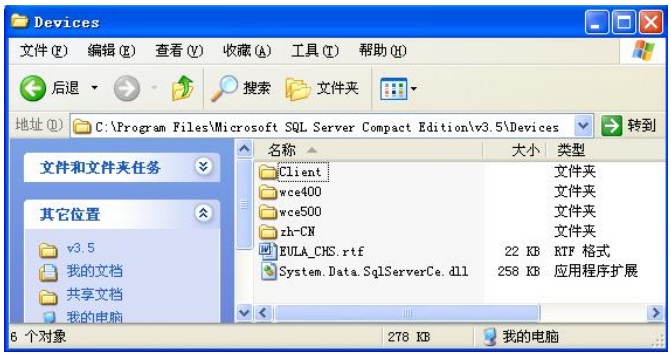

图 5-14　SQL Server Compact 设备的运行时组件

5. 用于移动设备的 Synchronization Services for ADO.NET 1.0 SP1 (Devices)安装

（1）双击 SyncServices.msi 安装程序，进入如图 5-15 所示的安装向导界面，单击 Next 按钮。

图 5-15　Synchronization Services for ADO.NET 1.0 SP1 (Devices)安装向导界面

（2）当安装完成之后，可以在%ProgramFiles%\Microsoft Synchronization Services\ADO.NET\v1.0\Devices 目录下查找到支持移动设备数据同步的组件，如图 5-16 所示。

图 5-16　移动设备数据同步的组件

5.4　创建 Synchronization Services for ADO.NET 的数据同步应用

5.4.1　SQL Server2005 数据库创建及安全性设置

（1）打开 SQL Server 2005 的 Management Studio 数据库管理工具，创建一个 SQL Server 2005 的 NoteDB 用户数据库，并新建一张表，表名为 NoteInfo，如图 5-17 所示。

图 5-17　在 SQL Server 2005 中创建 NoteDB 数据库和 NoteInfo 表

（2）在对象资源管理器中选择“安全性”→“登录名”，右击“登录名”，选择“新建登录名”，弹出“新建登录”对话框。选择“Windows 身份验证”选项，单击“搜索”按钮，在“选择用户或组”对话框中添加 PC-WHTC\ASPNET 对象，如图 5-18 所示，单击“确定”按钮。

（3）在如图 5-19 所示的“新建登录”对话框中，显示登录名为 PC-WHTC\ASPNET 对象。

图 5-18　添加 PC-WHTC\ASPNET 对象

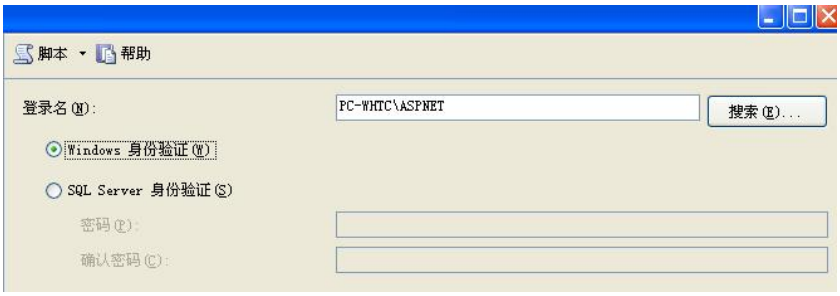

图 5-19　“新建登录”对话框显示 PC-WHTC\ASPNET 对象

（4）单击“用户映射”选项，接着在右面选择要映射的数据库 NoteDB，并在 db-owner 和 public 一栏前面打钩，如图 5-20 所示。然后单击“状态”选项，在右面栏中选中“授予”、“启用”，一般这两项是默认的，但如果默认的不是此两项必须改过来，不然是连不上的，最后单击“确定”按钮。

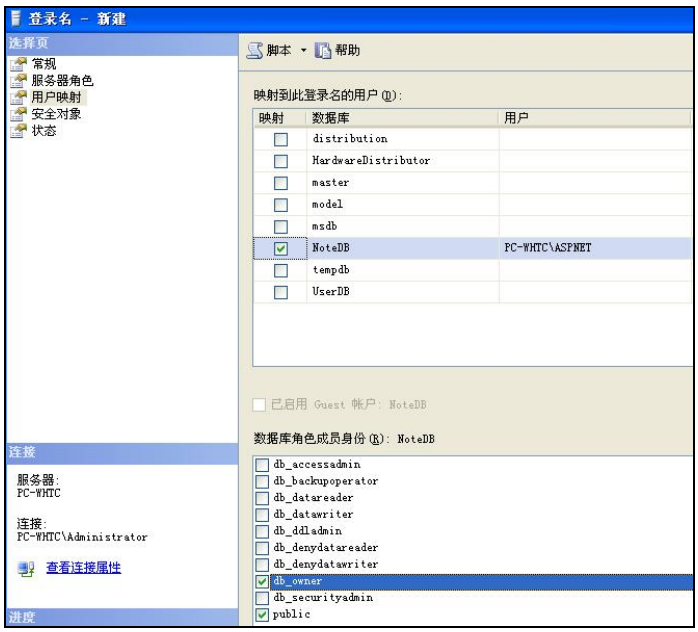

图 5-20　在用户映射中设置数据库权限

5.4.2 创建基于 Windows CE 的数据同步应用程序工程

（1）运行 VS.NET 2008 平台，选择“文件”→“新建”→“项目”选项，在“新建项目”对话框中选择“其他项目类型”→“Visual Studio 解决方案”，然后模板选择空白解决方案，并且框架选择.NET Framework 3.5 版本，最后名称栏输入 SynWebServiceNote，单击“确定”按钮，如图 5-21 所示。

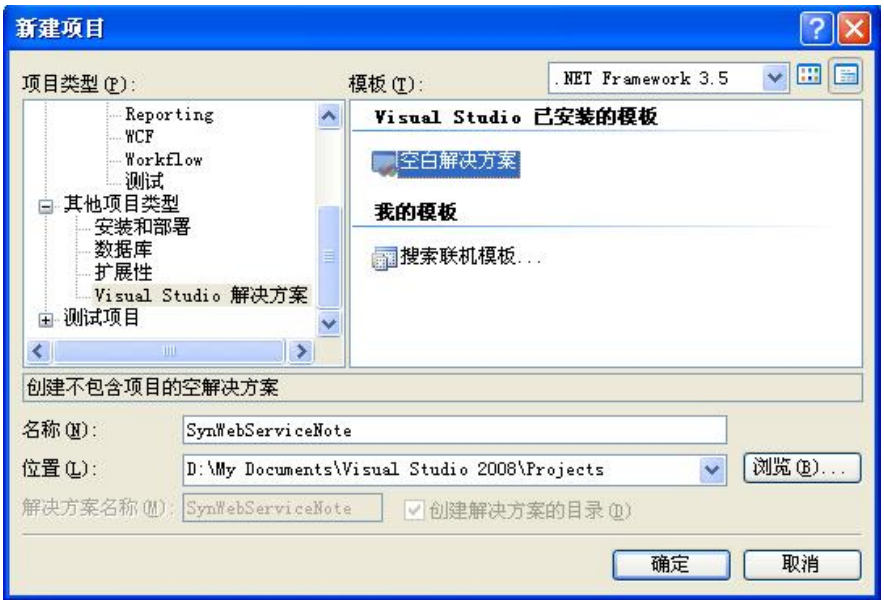

图 5-21 新建空白解决方案

（2）在解决方案资源管理器中，如图 5-22 所示，右击 SynWebServiceNote 解决方案，选择“添加”→“新建项目”选项。

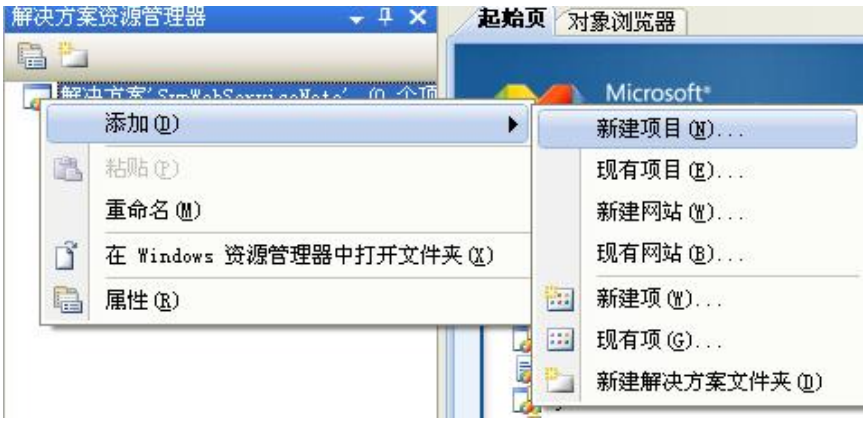

图 5-22 选择解决方案资源管理器新建项目

（3）在“添加新项目”对话框中，项目类型选择“Visual C#”→“智能设备”，模板选择“智能设备”项目，名称栏中输入 SynWinCEDevice，单击“确定”按钮，如图 5-23 所示。

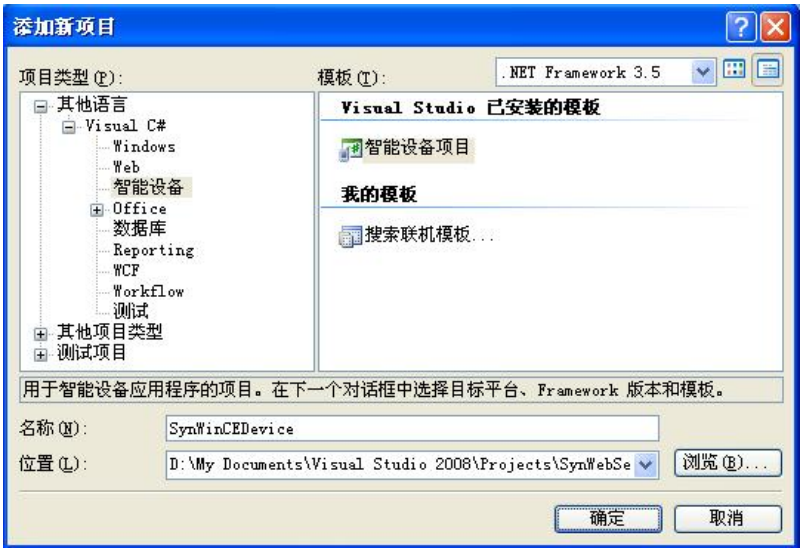

图 5-23　新建 C#智能设备项目

（4）在“添加新智能设备项目”对话框中，目标平台选择 Windows CE，.NET Compact Framework 版本选择.NET Compact Framework Version 3.5，模板选择“设备应用程序”，单击“确定”按钮，创建完成基于 Windows CE 的 C#设备应用程序，如图 5-24 所示。

图 5-24　创建完成基于 Windows CE 的设备应用项目

（5）在解决方案资源管理器中，右击 SynWebServiceNote 解决方案，选择“添加”→“新建项目”。在“添加新项目”对话框中，项目类型选择 Visual C#，模板选择类库，名称栏中输入 GBAServerSyncProvider，单击“确定”按钮，如图 5-25 所示。

（6）当智能设备应用项目与类库项目创建完成之后，可以在 SynWebServiceNote 解决方案资源管理器中查看到如图 5-26 所示的项目文件显示。

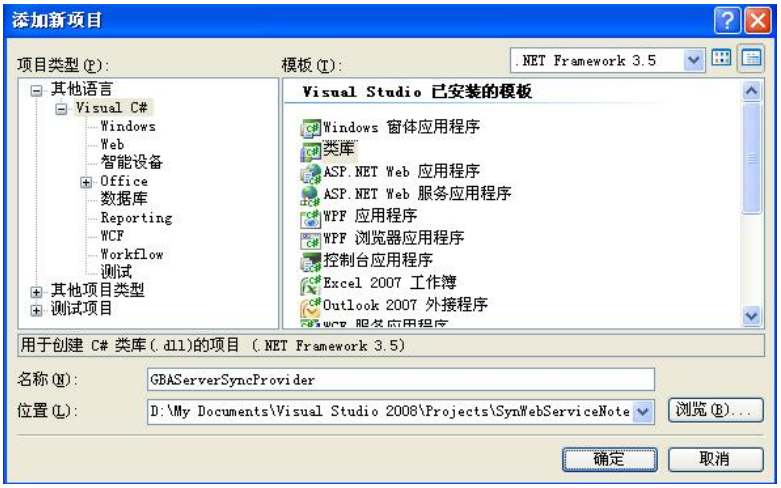

图 5-25　创建完成一个类库项目

图 5-26　智能设备与类库项目

5.4.3　工程项目中数据同步设置

（1）在解决方案资源管理器中，右击 GBAServerSyncProvider 类库项目，选择“添加”→“新建”选项，进入如图 5-27 所示的“添加新项”对话框，选择右边模板中的“本地数据库缓存”项，名称栏输入 ServerDataCache.sync，单击“添加”按钮。

图 5-27　添加本地数据库缓存项

（2）在如图 5-28 所示的“配置数据同步”对话框中，需要设置数据库的服务器连接和客户端连接，单击“新建”按钮。

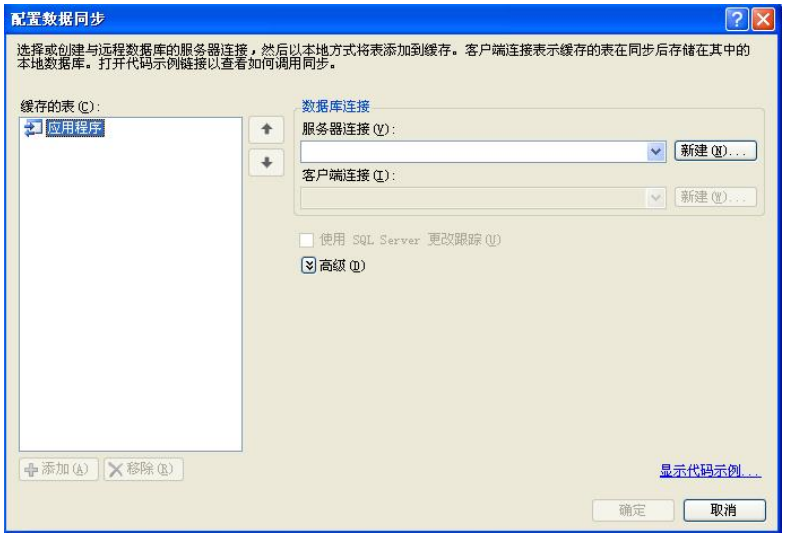

图 5-28　新建数据库连接

（3）在“添加连接”对话框中，如图 5-29 所示，如果数据源种类不是 Microsoft SQL Server 数据源，单击“更改”按钮。

图 5-29　“添加连接”对话框

（4）在“更改数据源”对话框，如图 5-30 所示，选择 Microsoft SQL Server 数据源选项，单击“确定”按钮。

（5）返回到“添加连接”对话框中，服务器名选择所在计算机的 SQL Server 2005 数据库服务器名称（PC-WHTC），选择使用 Windows 身份验证登录到服务器，数据库选择前面所创建的 NoteDB 数据库，然后单击下方的“测试”按钮，如果测试成功，表示数据库连接已成功创建，单击“确定”按钮，如图 5-31 所示。

图 5-30 “更改数据源”对话框

图 5-31 成功创建数据库连接

（6）在“配置数据同步”对话框中，单击“高级”按钮展开数据同步选项配置，其中服务器项目位置选择 GBAServerSyncProvider 类库项目，客户端项目选择 SynWinCEDevice 智能设备项目，如图 5-32 所示。

（7）在如图 5-32 所示的“配置数据同步”对话框中，单击左下方的“添加”按钮，进入如图 5-33 所示的同步表对话框中，这里将在 NoteInfo 表自动创建 LastEditDate、CreationDate 列以及新增 NoteInfo_Tombstone 表。“使用下列项比较更新”下拉列表框中的 LastEditDate 是要在远程服务器的 NoteInfo 表中建立的一个字段，该字段是日期型，用于记录当前数据行的最后一次修改变时间。“使用下列项比较插入”下拉列表框中的 CreationDate 是该行数据的插入时间。“将已删除的项移至”下拉列表框中的 NoteInfo_Tombstone 是当删除 NoteInfo 表中的某一行数据时，将该行数据的主键 NoteID 和删除时间 DetetionDate 记录到 NoteInfo_Tombstone 表中，单击“确定”按钮。

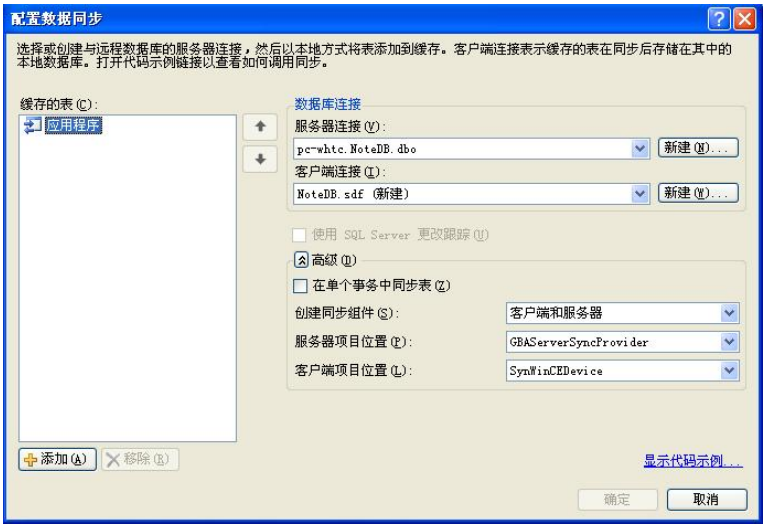

图 5-32　服务器和客户端数据同步选项配置

图 5-33　添加和创建同步的字段和表

（8）当出现“生成 SQL 脚本”对话框时，单击“确认”按钮，生成 SQL 脚本，如图 5-34 所示。这样通过以上的设置，系统就会拥有了关于进行数据记录（创建、修改、删除）的时间戳，作为进行数据同步时的依据。

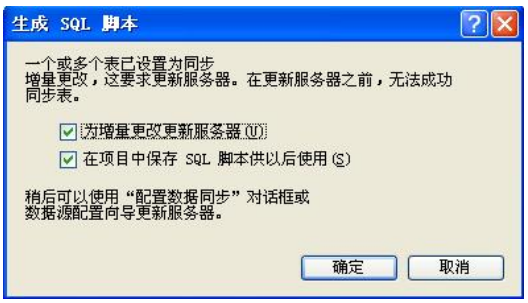

图 5-34　生成 SQL 脚本

（9）由于工程项目中对 SQL Server Compact 3.5 数据库的各项操作是通过编写代码实现的，不需要生成强类型数据集，所以这里单击“取消”按钮不生成强类型数据集 DataSet，如图 5-35 所示。

图 5-35　取消生成强类型数据集

（10）在 SQL Server 2005 的 Management Studio 数据库管理工具中，打开 SQL Server 2005 的 NoteDB 数据库中的 NoteInfo 表，可以查看到，通过项目工程的数据同步配置之后，NoteInfo 表中增加了 LastEditDate、CreationDate 两列以及新增了 NoteInfo_Tombstone 表，这表示项目工程的数据同步配置成功，如图 5-36 所示。

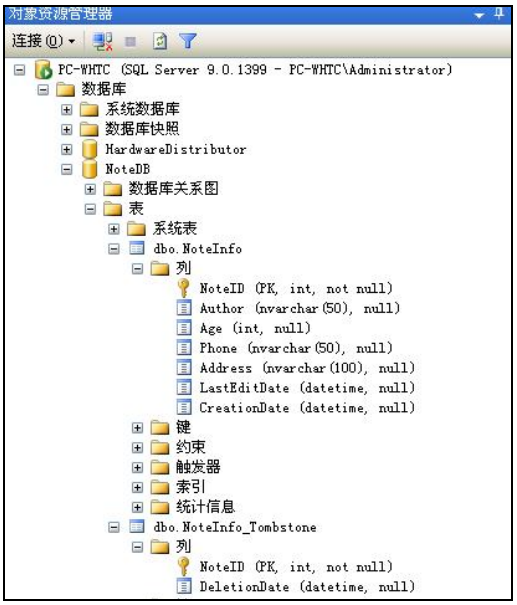

图 5-36　NoteDB 数据库的新增列以及新增表

5.4.4　类库项目的功能实现

（1）在解决方案资源管理器中，右击选择 Class.cs 文件的属性。将 Class.cs 文件名改为 GBABuilderServerSyncProvider.cs，如图 5-37 所示。

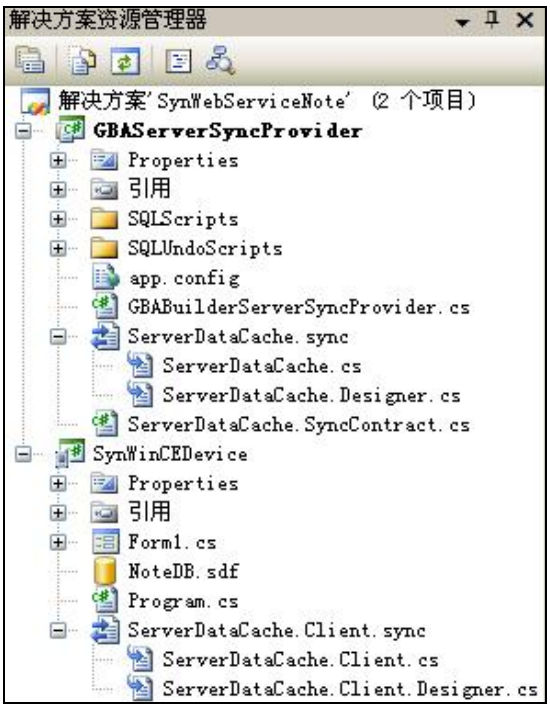

图 5-37　修改类库项目文件属性

（2）在 GBABuilderServerSyncProvider.cs 文件中添加 GBABuilderServerSyncProvider 类的构造方法，这里要注意的是"SELECT@"+SyncSession.SyncNewReceivedAnchor=GETUTCDATE()"这一句，因为这里使用的是 GETUTCDATE()，即以 UTC 时间（通用协调时间或格林尼治标准时间）表示的系统当前日期。另外同步适配器生成器 SqlSyncAdapterBuilder 允许指定要同步的表、表中的跟踪列、同步方向以及要包括的行和列。

具体代码实现如下：

```
public class GBABuilderServerSyncProvider: DbServerSyncProvider
{
    public GBABuilderServerSyncProvider()
    {
    this.Connection=new    SqlConnection(Properties.Settings.Default.
    ServerNoteDBConnectionString);
    SqlCommand    selectNewAnchorCmd=new SqlCommand();
    selectNewAnchorCmd.CommandText="SELECT@"+
    SyncSession.SyncNewReceivedAnchor +
    " = GETUTCDATE()";
      SqlParameter newRecAnchor = new SqlParameter();
      newRecAnchor.ParameterName = "@" + SyncSession.SyncNewReceivedAnchor;
```

```
            newRecAnchor.DbType = System.Data.DbType.DateTime;
            newRecAnchor.Direction = System.Data.ParameterDirection.Output;
            selectNewAnchorCmd.Parameters.Add(newRecAnchor);
            this.SelectNewAnchorCommand = selectNewAnchorCmd;
            SqlSyncAdapterBuilder   builder=new SqlSyncAdapterBuilder((SqlConnection)this.Connection);
            builder.SyncDirection = SyncDirection.Bidirectional;
            builder.CreationTrackingColumn = "CreationDate";
            builder.UpdateTrackingColumn = "LastEditDate";
            builder.DeletionTrackingColumn = "DeletionDate";
            builder.TableName = "NoteInfo";
            builder.TombstoneTableName = builder.TableName + "_Tombstone";
            this.SyncAdapters.Add(builder.ToSyncAdapter(true, true, false, false));
        }
    }
```

（3）右击 GBAServerSyncProvider 类库项目，选择“生成”选项，这时将在 D:\My Documents\Visual Studio 2008\Projects\SynWebServiceNote\GBAServerSyncProvider\bin\Debug 目录下生成 GBAServerSyncProvider.dll 动态链接库，后面 Web 服务要引用这个 GBAServerSyncProvider.dll，如图 5-38 所示。

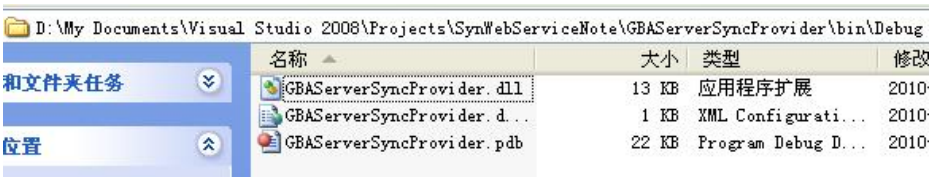

图 5-38　生成 GBAServerSyncProvider.dll 动态链接库

5.4.5　Web 服务项目功能实现

在基于 N 层体系结构的数据同步中，客户端不是直接与服务器数据库进行通信，而是通过与中间层进行通信以达到和服务器进行通信。而中间层包含了对外提供 WebService 的数据同步服务方法，这就需要建立 Web Service 工程项目，并将前面 GBAServerSyncProvider 类库项目中编译生成的 DLL 引入 Web Service 工程项目中，其中编译生成的 DLL 包含了用于数据同步操作的方法。然后在 Web Service 工程项目中利用这个 DLL 建立 Web 服务的方法。

（1）在解决方案资源管理器中，右击 SynWebServiceNote 解决方案，选择“添加”→“新建项目”。在“添加新项目”对话框中，项目类型选择 Visual C#，模板选择“ASP.NET Web 服务应用程序”，名称栏输入 SynWebService，单击“确定”按钮，如图 5-39 所示。

（2）在如图 5-40 所示的解决方案资源管理器中，右击 SynWebService 工程项目，选择“引用”→“添加引用”。

（3）在“添加引用”对话框中，选择“浏览选”项卡，查找 GBAServerSyncProvider 类库项目中编译生成的 GBAServerSyncProvider.dll，单击“确定”按钮，如图 5-41 所示。

（4）按照前面 SynWebService 工程项目中添加 GBAServerSyncProvider.dll 引用的步骤，再将 Microsoft.Synchronization.Data 和 Microsoft.Synchronization.Data.Server 添加到 SynWebService 工程项目中，如图 5-42 所示。

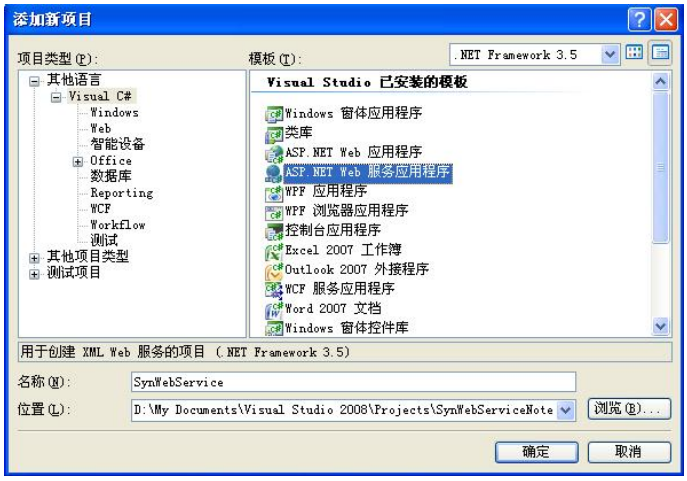

图 5-39　新建 Web Service 工程项目

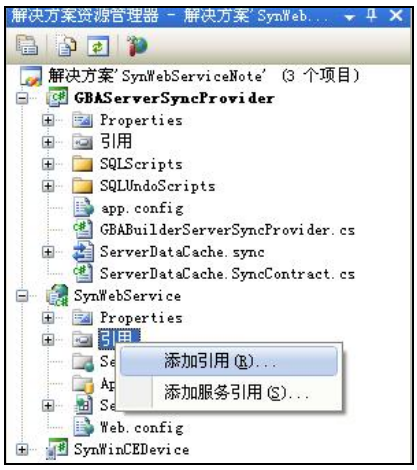

图 5-40　SynWebService 工程项目添加引用

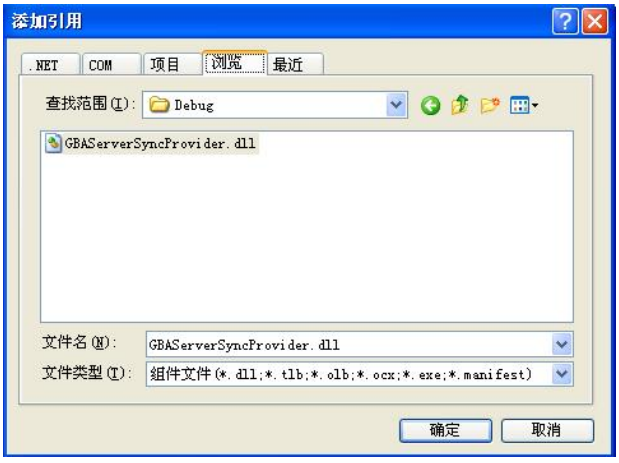

图 5-41　引用 GBAServerSyncProvider 类库项目中生成的 DLL

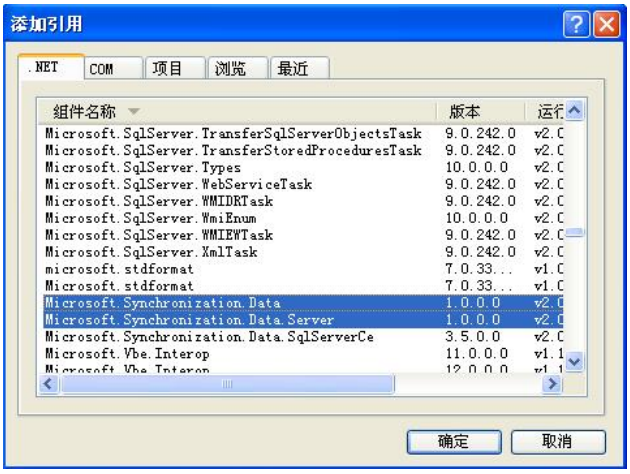

图 5-42　添加 Microsoft.Synchronization.Data 和 Microsoft.Synchronization.Data.Server 引用

（5）添加完成上述所有引用之后，在解决方案资源管理器中单击引用文件夹，可以查看到如图 5-43 所示添加完成的三个 DLL 引用。

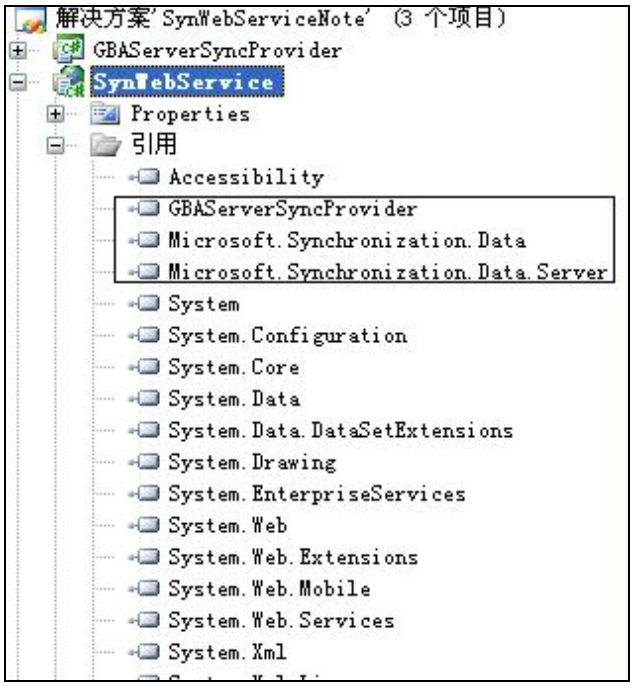

图 5-43　添加完成的三个 DLL 引用

（6）在 GBAServerSyncProvider 类库项目中，打开 ServerDataCache.SyncContract.cs 文件，可以找到服务器同步提供的程序对象 ServerSyncProvider 以及四个对外发布的数据同步服务方法：ApplyChanges 方法、GetChanges 方法、GetSchema 方法以及 GetServerInfo 方法。然后这些对象和方法复制拷贝到 SynWebService 工程项目的 Service1.asmx 代码文件中，并进行修改。WebService 必须实现这四个方法才可以正常地去同步客户端与服务器端的数据。

1）ApplyChanges：将同步表中的数据插入、更新和删除应用到服务器数据库。

2）GetChanges：从服务器数据存储区中选择要在客户端存储区应用的同步组中每个表的数据增量进行插入、更新和删除。

3）GetSchema：从服务器数据库中检索一个或多个表的架构。

4）GetServerInfo：获取服务器同步信息。

修改完成之后如以下代码所示：

```
public class Service1 : System.Web.Services.WebService
{
    private ServerDataCacheServerSyncProvider _serverSyncProvider;
    public Service1()
    {
        this._serverSyncProvider = new ServerDataCacheServerSyncProvider();
    }
    [WebMethod]
    public    SyncContext ApplyChanges(SyncGroupMetadata groupMetadata, DataSet dataSet,
```

```
        SyncSession syncSession)
        {
            return this._serverSyncProvider.ApplyChanges(groupMetadata, dataSet, syncSession);
        }
            [WebMethod]
            public   SyncContext GetChanges(SyncGroupMetadata groupMetadata, SyncSession syncSession)
        {
            return this._serverSyncProvider.GetChanges(groupMetadata, syncSession);
        }
            [WebMethod]
            public   SyncSchema GetSchema(Collection<string> tableNames, SyncSession syncSession)
        {
            return this._serverSyncProvider.GetSchema(tableNames, syncSession);
        }
            [WebMethod]
            public   SyncServerInfo GetServerInfo(SyncSession syncSession)
        {
            return this._serverSyncProvider.GetServerInfo(syncSession);
        }
    }
```

5.4.6　Web 服务项目的发布

SynWebService 工程项目中有关数据同步的 Web 方法实现完成之后，就可以将其通过 IIS 服务器对外进行发布，以提供给客户端数据同步的 Web 服务功能。以下将完成 SynWebService 工程项目的对外发布。

（1）找到 SynWebService 工程项目所在的文件夹之后，右击 SynWebService 文件夹，选择“共享和安全”选项，如图 5-44 所示。

图 5-44　选择“共享和安全”选项

（2）选择“Web 共享”选项卡，单击“共享文件夹”选项，如图 5-45 所示。

图 5-45　选择“共享文件夹”选项

（3）当出现“编辑别名”对话框时，可以按照如图 5-46 所示的默认选项选择其中的各项属性值，然后单击“确定”按钮，完成 Web 共享文件夹的设置，如图 5-47 所示。

图 5-46　别名对话框编辑属性

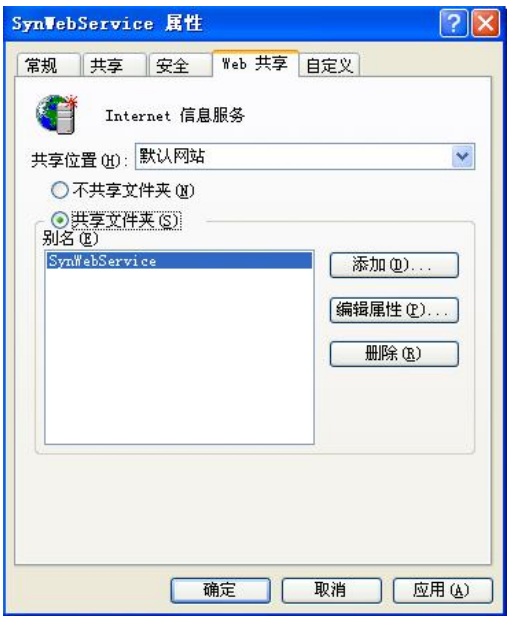

图 5-47　完成 Web 共享文件夹的设置

（4）打开“控制面板”→“管理工具”→“Internet 信息服务”，如图 5-48 所示，可以查看到 SynWebService 项目通过前面的操作步骤，完成了对外发布数据同步的 Web 服务功能。

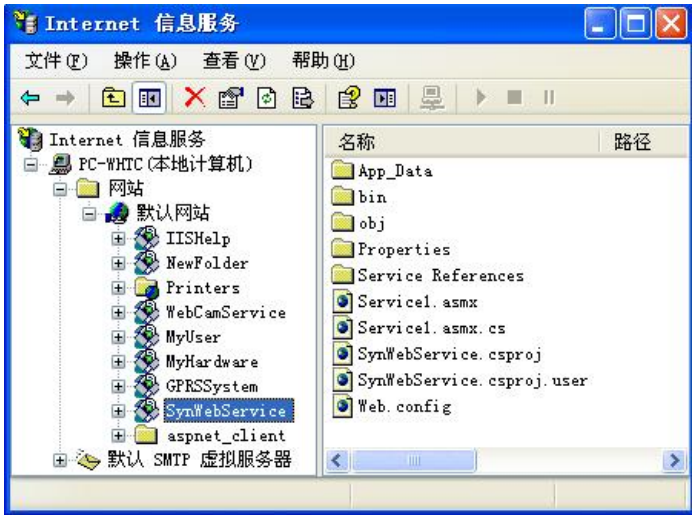

图 5-48　IIS 显示 SynWebService 服务项目的 Web 服务发布

（5）在 IIS 中完成了 SynWebService 项目的 Web 发布之后，可以通过在 IE 浏览器的地址栏上输入 http://192.168.1.101/SynWebService/Service1.asmx 测试一下 Web 服务发布是否成功。其中 192.168.1.101 是所在计算机的 IP 地址，如图 5-49 所示表示 Web 服务已成功发布。

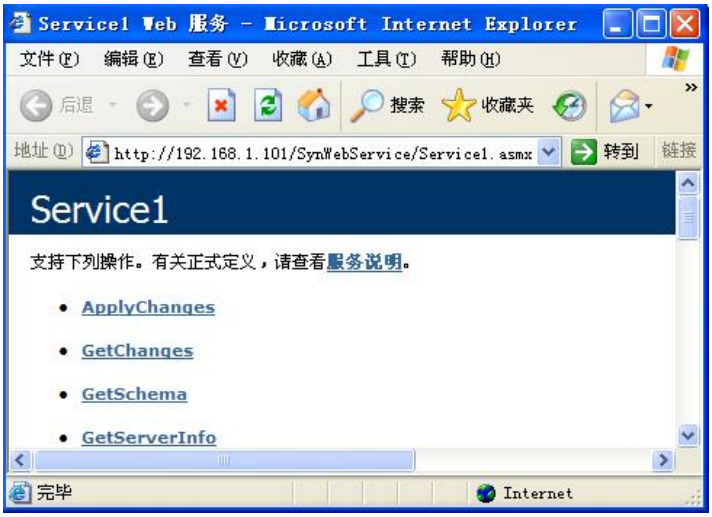

图 5-49　SynWebService 项目的 Web 服务发布成功

5.4.7　智能设备项目数据访问及业务逻辑功能实现

1. 相关类的设计

在项目解决方案资源管理器中右击 SynWinCEDevice 项目，选择“添加”→“新建文件夹”，增加 BLL、DAL、Sync 三个文件夹，然后选择类模板，分别在三个文件夹中添加类文件，其中用于对数据库访问的类为 CommonDB.cs，用于业务操作的类为 NoteBLL.cs 和 Notes.cs，实现与远程数据库服务器同步操作的类为 ClientSyncAgent.cs 和 ServerSyncProviderProxy.cs，如图 5-50 所示。

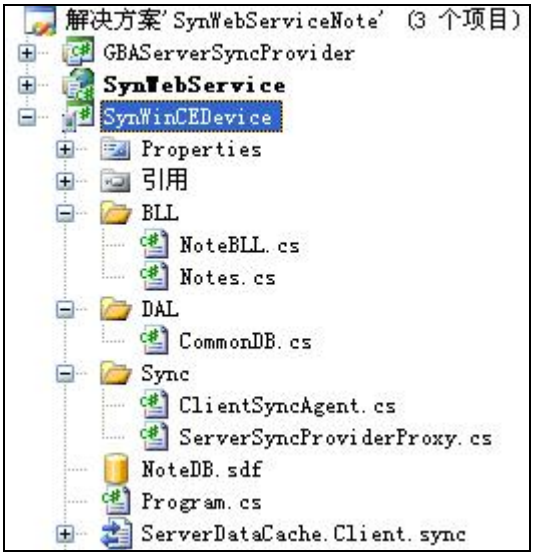

图 5-50　添加相关的类文件

2. 功能窗体的设计

在项目解决方案资源管理器中右击 SynWinCEDevice 项目，选择“添加”→“新建项”，选择 Windows 窗体模板，分别添加显示数据主窗体（MainForm.cs）、新增窗体（AddFrm.cs）以及更新窗体（EditFrm.cs），如图 5-51 所示。

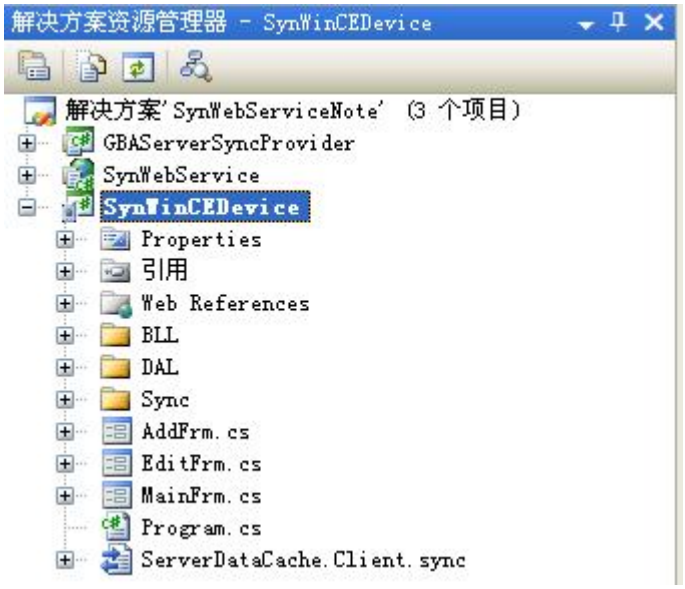

图 5-51　添加相关功能窗体文件

3. 数据访问层类的功能实现

上一章 SQL Server Compact 数据库应用程序开发已经详细介绍了数据访问层和业务逻辑层相关类的功能实现，现只简要给出有关方法功能注释和实现的主要源代码：

class CommonDB

```
{    //获取连接数据库文件的连接对象
     public SqlCeConnection GetConnection()
     {
         string constr = "Data Source=" + @"\My Documents\NoteDB.sdf";
         SqlCeConnection Con = new SqlCeConnection(constr);
         return Con;
     }
     //根据 Sql 查询语句和表名称作为参数，执行对 SQL Server Compact 数据库的查询操作，将获
     //得的数据记录填充到数据集中，并返回数据集 DataSet
public DataSet GetDataSet(string Sql, string tablename)
     {
          SqlCeConnection Con = this.GetConnection();
          SqlCeCommand Cmd = new SqlCeCommand(Sql, Con);
          SqlCeDataAdapter Sda = new SqlCeDataAdapter(Cmd);
          DataSet Ds = new DataSet();
          try
          {
               Sda.Fill(Ds, tablename);
               return Ds;
          }
          catch (SqlCeException ex)
          {
               if (Con.State != ConnectionState.Closed)
               {
                    Con.Close();
               }
               MessageBox.Show(ex.Message);
          }
          return null;
     }
     //根据（Insert、Update、Delete）语句和参数化对象集合作为参数，执行对 SQL Server Compact
     //数据库的增删改操作，并返回受影响的行数
public int ExcuteSql(string Sql, SqlCeParameter[] paras)
{
     int flag = 0;
     SqlCeConnection Con = this.GetConnection();
     SqlCeCommand Cmd = new SqlCeCommand(Sql, Con);
     foreach (SqlCeParameter p in paras)
     {
          Cmd.Parameters.Add(p);
     }
     try
     {
          Con.Open();
          flag = Cmd.ExecuteNonQuery();
     }
```

```
            catch (SqlCeException ex)
            {
                MessageBox.Show(ex.Message);
            }
            finally
            {
                Con.Close();
            }
            return flag;
        }
    }
```

4. 业务逻辑层类功能实现

（1）实体类功能实现。

```
class Notes
{
    public int NoteID
    {get;set; }
    public string Author
    {get;set; }
    public int Age
    {get;set; }
    public string Phone
    {get;set; }
    public string Address
    {get;set; }
}
```

（2）对NoteInfo表的业务逻辑功能实现。现简要给出有关业务逻辑功能注释和实现的主要代码：

```
class NoteBLL
{
    private DataGrid Notedgrid;
    private BindingSource Bding;
    public NoteBLL()
    {
    }
    //通过构造方法传递 BindingSource 对象和 DataGrid 控件对象
public NoteBLL(BindingSource bds, DataGrid dgrid)
{
    Notedgrid = dgrid;
    Bding = bds;
}
    //在方法中通过创建 CommonDB 对象，调用 CommonDB 对象中 GetDataSet 的方法，获取
    //NoteInfo 表中所有数据行记录
    public DataSet GetNoteData()
    {
        CommonDB cdb = new CommonDB();
```

```
        string Sql = "select * from NoteInfo";
        DataSet Ds = cdb.GetDataSet(Sql, "NoteInfo");
        return Ds;
    }
    //在方法中根据传递的用户名、年龄、电话号码及联系地址值作为参数，调用 CommonDB 对
    //象中 ExcuteSql 方法，执行对数据库中 NoteInfo 表的新增操作
    public int InsertNoteData(Notes ns)
    {
        CommonDB cdb = new CommonDB();
        string Sql = "insert into NoteInfo (Author,Age,Phone,Address) values(@author,
        @age,@phone,@address)";
          SqlCeParameter[] paras = new SqlCeParameter[4];
          paras[0] = new SqlCeParameter("@author", ns.Author);
          paras[1] = new SqlCeParameter("@age", ns.Age);
          paras[2] = new SqlCeParameter("@address", ns.Address);
          paras[3] = new SqlCeParameter("@phone", ns.Phone);
          return cdb.ExcuteSql(Sql, paras);
    }
    //在方法中根据传递的用户名 ID、用户名、年龄、电话号码及联系地址值作为参数，调用
    //CommonDB 对象中 ExcuteSql 方法，执行对数据库中 NoteInfo 表的更新操作
    public int UpdateNoteDataByNotID(Notes ns)
    {
        CommonDB cdb = new CommonDB();
        string Sql = "update NoteInfo set Author=@author,Age=@age,Phone=@phone,
        Address=@address where NoteID=@noteid";
        SqlCeParameter[] paras = new SqlCeParameter[5];
        paras[0] = new SqlCeParameter("@author", ns.Author);
        paras[1] = new SqlCeParameter("@age", ns.Age);
        paras[2] = new SqlCeParameter("@address", ns.Address);
        paras[3] = new SqlCeParameter("@phone", ns.Phone);
        paras[4] = new SqlCeParameter("@noteid", ns.NoteID);
        return cdb.ExcuteSql(Sql, paras);
    }
    //在方法中根据传递的用户名 ID 作为参数，调用 CommonDB 对象中 ExcuteSql 方法，执行对
    //数据库中 NoteInfo 表的删除操作
    public int DeleteNoteDataByNotID(string Noteid)
    {
        CommonDB cdb = new CommonDB();
        string Sql = "delete NoteInfo where NoteID=@noteid";
        SqlCeParameter[] paras = new SqlCeParameter[1];
        paras[0] = new SqlCeParameter("@noteid", Noteid);
        return cdb.ExcuteSql(Sql, paras);
    }
    //该方法通过调用 CommonDB 对象的 GetNoteData 方法，将获得的数据集的默认视图作为
    //BindingSource 对象的数据源，然后再将 BindingSource 对象的数据源作为 DataGrid 控件的数
    //据源，以便在 DataGrid 控件中实时显示更新的信息
    public void UpdateGird()
```

```
        {
            DataSet Ds = GetNoteData();
            Bding.DataSource = Ds.Tables[0].DefaultView;
            Notedgrid.DataSource = Bding;
        }
    }
```

5. Web 服务代理类功能实现

在 N 层数据同步应用程序中，客户端与服务器之间的数据同步通信是通过中间层实现的，而中间层提供的功能操作是以 Web 服务的形式对外发布的，在中间层该服务继承 ServerSyncProvider 类并公开与其相同的方法。然后通过与服务器数据库的直接连接执行服务器同步提供程序的方法，结果通过中间层发送回客户端。这样客户端应用程序为了获得 Web 服务提供的数据同步操作功能，就需要一个代理类帮助实现，代理类可以通过在 SynWinCEDevice 项目中添加 Web 引用自动产生，通过代理类，可以引用远程的 Web 服务，在智能设备应用程序中使用其功能，返回的数据就像是本地生成一样。

（1）右击 SynWinCEDevice 项目，选择“引用”→“添加服务引用”。在“添加服务引用”对话框中，URL 栏中输入：http://192.168.1.101/SynWebService/Service1.asmx，单击“前往”按钮，这时在下方会出现可以使用的四个 Web 服务方法，如图 5-52 所示，单击“添加引用”按钮。

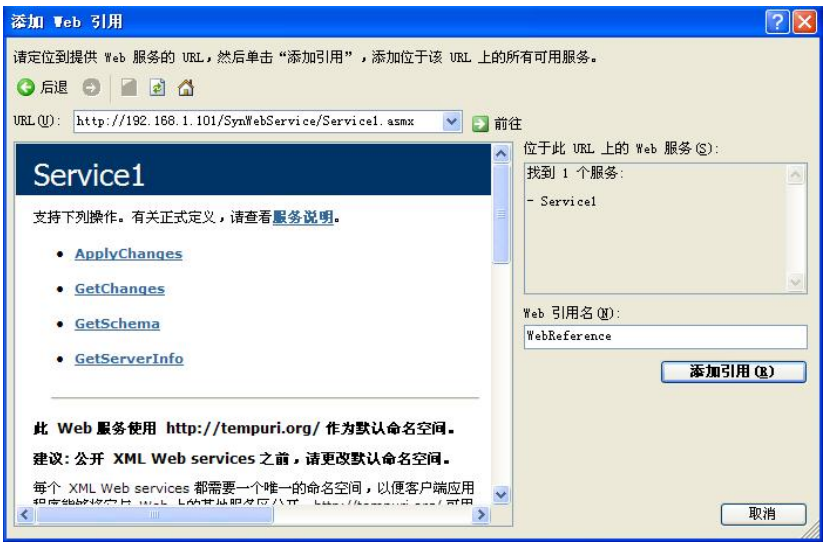

图 5-52　在客户端添加 Web 服务引用

（2）在 SynWinCEDevice 项目中添加 Web 引用之后，可以在解决方案资源管理器中查看到 Web References 文件夹中包含很多文件，其中 Reference.cs 文件包含 Web 服务的代理类。如图 5-53 所示。

（3）打开 Reference.cs 文件，只保留 Service1 代理类，其他类全部注释掉或者直接删除即可。

在 Service1 类的构造方法中设置 Web Service 服务的 URL 链接地址，这里为 http://192.168.1.101/SynWebService/Service1.asmx。

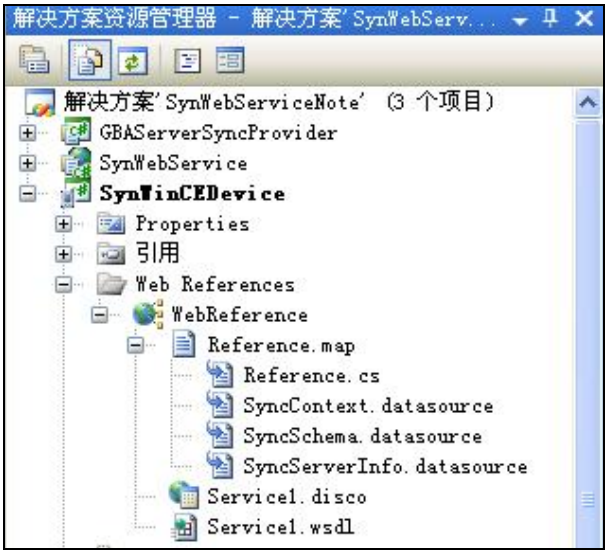

图 5-53　显示 Web References 文件夹

修改完成之后，代码结构如下：

```
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Web.Services.WebServiceBindingAttribute(Name="Service1Soap",
Namespace="http://tempuri.org/")]
public partial class Service1 : System.Web.Services.Protocols.SoapHttpClientProtocol {
public partial class Service1 : System.Web.Services.Protocols.SoapHttpClientProtocol
{
public Service1()
{
this.Url = "http://192.168.1.101/SynWebService/Service1.asmx";
}
public SyncContext ApplyChanges(SyncGroupMetadata groupMetadata, System.Data.DataSet dataSet,
SyncSession syncSession)
{…}
public System.IAsyncResult BeginApplyChanges(SyncGroupMetadata groupMetadata,
System.Data.DataSet dataSet, SyncSession syncSession, System.AsyncCallback callback, object
asyncState)
{…}
public SyncContext EndApplyChanges(System.IAsyncResult asyncResult)
{….}
public SyncContext GetChanges(SyncGroupMetadata groupMetadata, SyncSession syncSession)
{….}
public System.IAsyncResult BeginGetChanges(SyncGroupMetadata groupMetadata, SyncSession
syncSession, System.AsyncCallback callback, object asyncState)
{…}
public SyncContext EndGetChanges(System.IAsyncResult asyncResult)
{…}
```

```
public SyncSchema GetSchema(string[] tableNames, SyncSession syncSession)
{…}
public System.IAsyncResult BeginGetSchema(string[] tableNames, SyncSession syncSession,
System.AsyncCallback callback, object asyncState)
{…}
public SyncSchema EndGetSchema(System.IAsyncResult asyncResult)
{…}
public SyncServerInfo GetServerInfo(SyncSession syncSession)
{…}
public System.IAsyncResult BeginGetServerInfo(SyncSession syncSession, System.AsyncCallback
callback, object asyncState)
{…}
public SyncServerInfo EndGetServerInfo(System.IAsyncResult asyncResult)
{…}
}
```

6. ServerSyncProviderProxy 派生类功能实现

ServerSyncProviderProxy 继承 ServerSyncProvider 类，ServerSyncProvider 类对象用于提供与服务器数据存储区（Server Database）进行通信，并使同步代理与该数据存储区的特定实现隔离服务器同步提供程序。该对象用于与远程数据存储区进行通信。ServerSyncProvider 类中包含前面 Web 引用之后的代理类对象 ServiceProxy 以及四个重载方法。

（1）ApplyChanges：将把同步表中的数据插入、更新和删除应用到服务器数据库。

（2）GetChanges：从服务器数据存储区中选择要在客户端存储区应用的同步组中每个表的数据增量进行插入、更新和删除。

（3）GetSchema：从服务器数据库中检索一个或多个表的架构。

（4）GetServerInfo：获取服务器同步信息。

具体代码实现如下：

```
public class ServerSyncProviderProxy : ServerSyncProvider
{
    private WebReference.Service1 serviceProxy;
    public ServerSyncProviderProxy(object serviceProxy)
    {
        this.serviceProxy = (WebReference.Service1)serviceProxy;
    }
    public ServerSyncProviderProxy()
    {
    }
    public override SyncContext ApplyChanges(SyncGroupMetadata groupMetadata, DataSet dataSet,
    SyncSession syncSession)
    {
        return serviceProxy.ApplyChanges(groupMetadata,dataSet,syncSession);
    }
    public override void Dispose()
    {
```

```
        }
        public override SyncContext GetChanges(SyncGroupMetadata groupMetadata, SyncSession syncSession)
        {
            return serviceProxy.GetChanges(groupMetadata, syncSession);
        }
        public override SyncSchema GetSchema(Collection<string> tableNames, SyncSession syncSession)
        {
            string[] strTableNames = new string[tableNames.Count];
            tableNames.CopyTo(strTableNames, 0);
            return serviceProxy.GetSchema(strTableNames, syncSession);
        }
        public override SyncServerInfo GetServerInfo(SyncSession syncSession)
        {
            return serviceProxy.GetServerInfo(syncSession);
        }
    }
```

7. ClientSyncAgent 派生类功能实现

ClientSyncAgen 继承 SyncAgent 类，SyncAgent 类对象用于组织客户端与服务器之间的同步过程。其中 LocalProvider 属性用于获取一个派生自 ClientSyncProvider 抽象类的 SqlCeClientSyncProvider 对象，它表示客户端同步提供程序。RemoteProvider 属性用于获取一个派生自 ServerSyncProvider 抽象类的 ServerSyncProviderProxy 对象，它表示服务器端同步提供程序。将 NoteInfo 表增加到 SyncTable 同步表，并设置同步方向为 Bidirectional（双向同步），若客户端数据库中已经存在此表，则删除后再重新创建，最后将 SyncTable 添加到 SyncAgent 中。

具体代码实现如下：

```
public class ClientSyncAgent : SyncAgent
{
    public ClientSyncAgent()
    {
        string constr = "Data Source=" + @"\My Documents\NoteDB.sdf";   //连接客户端数据库的连接字符串
        this.LocalProvider = new SqlCeClientSyncProvider(constr, true);
        this.RemoteProvider =new ServerSyncProviderProxy(new WebReference.Service1());
        this.Configuration.SyncTables.Add("NoteInfo");
        this.Configuration.SyncTables["NoteInfo"].SyncDirection = SyncDirection.Bidirectional;
        this.Configuration.SyncTables["NoteInfo"].CreationOption= TableCreationOption.DropEx
        istingOrCreateNewTable;
    }
}
```

5.4.8　智能设备项目窗体功能实现

1. 数据信息显示（MainForm）窗体功能实现

（1）MainForm 窗体界面设计，如图 5-54 所示。

图 5-54　MainForm 窗体界面设计

（2）MainForm 窗体代码文件(MainForm.cs)结构。

```
public partial class MainForm : Form
    {
      private BindingSource bs = null;
      private string Noteid = null;
      private ClientSyncAgent _syncAgent;
      public MainForm()
      {
            InitializeComponent();
      }
      private void SyncNow()
      {
      }
      private void MainForm_Load(object sender, EventArgs e)
      {
      }
      private void btnAdd_Click(object sender, EventArgs e)
      {
      }
      private void btnEdit_Click(object sender, EventArgs e)
      {
      }
      private void btnDelete_Click(object sender, EventArgs e)
      {
      }
      private void btnSyn_Click(object sender, EventArgs e)
      {
      }
}
```

（3）SyncNow 数据同步方法。在该方法中，首先创建 ClientSyncAgent 同步代理对象，它是用于组织客户端与服务器之间的同步过程，然后调用 ClientSyncAgent 对象的 Synchronize 方法完成数据的双向同步操作。

具体代码实现如下：

```
private void SyncNow()
{
    Cursor.Current = Cursors.WaitCursor;
    try
    {
        if (null == _syncAgent)
          _syncAgent = new ClientSyncAgent();
        _syncAgent.Synchronize();
        MessageBox.Show("同步完成！ ");
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error during synchronization..." + ex.ToString());
    }
    finally
    {
        Cursor.Current = Cursors.Default;
    }
}
```

（4）MainForm_Load 事件处理方法。在 MainForm 窗体加载事件处理过程中，首先调用 File 类的 Exists 方法，判断设备端上指定的数据库文件是否存在，如果不存在，调用 SyncNow 方法完成设备端数据库的创建以及数据同步的操作，然后创建 BindingSource 对象，将 BindingSource 对象和 Datagrid 控件对象作为参数创建业务逻辑层 NoteBLL 类的对象，最后调用 NoteBLL 对象的 UpdateGird 方法，获得 NoteInfo 表的数据记录，并显示在窗体 Datagrid 列表控件中。

具体代码实现如下：

```
private void MainForm_Load(object sender, EventArgs e)
{
    string FilePath =@"\My Documents\NoteDB.sdf";
    if (!File.Exists(FilePath))
    {
        SyncNow();
    }
     bs = new BindingSource();
     NoteBLL ndb = new NoteBLL(bs, dgidNote);
     ndb.UpdateGird();
}
```

（5）btnAdd_Click 事件处理方法。单击 MainForm 窗体中的“新增”按钮之后，调用 AddFrm 窗体中带参数的构造函数，参数分别为 BindingSource 对象和 Datagrid 控件对象，创建完 AddFrm 窗体对象之后，打开新增客户信息窗体。

具体代码实现如下：

```
private void btnAdd_Click(object sender, EventArgs e)
```

```
{
    AddFrm af = new AddFrm(bs, dgidNote);
    af.ShowDialog();
}
```

（6）btnEdit_Click 事件处理方法。在 MainForm 窗体中，用手写笔单击触摸屏，选择欲编辑的某条数据记录，然后单击编辑按钮，这时在事件方法处理过程中，调用 EditFrm 窗体中带参数的构造方法，参数分别为 Datagrid 控件对象和 BindingSource 对象，创建完 EditFrm 窗体对象之后，打开更新客户信息窗体。

具体代码实现如下：

```
private void btnEdit_Click(object sender, EventArgs e)
{
    EditFrm ef = new EditFrm(dgidNote, bs);
    ef.ShowDialog();
}
```

（7）btnDelete_Click 事件处理方法。在 MainForm 窗体中，用手写笔单击触摸屏，选择欲删除的某条数据记录，然后单击“删除”按钮，这时在事件方法处理过程中，由 BindingSource 对象的 Current 属性获得选中的数据行视图 DataRowView，通过这一数据行视图记录获得 NoteInfo 表中 Noteid 主键值，然后调用业务逻辑层 NoteBLL 类中带参数的构造方法创建 NoteBLL 类的对象，参数分别为 BindingSource 对象和 Datagrid 控件对象，创建完 NoteBLL 对象之后，调用 NoteBLL 对象的 DeleteNoteDataByNotID 方法，并传递进 Noteid 主键值参数，如果删除记录成功，MainForm 窗体会实时显示更新之后的客户信息。

具体代码实现如下：

```
private void btnDelete_Click(object sender, EventArgs e)
{
    DataRowView dv = (DataRowView)bs.Current;
    Noteid = dv["NoteID"].ToString();
    NoteBLL ndb = new NoteBLL(bs, dgidNote);
    if (ndb.DeleteNoteDataByNotID(Noteid) > 0)
    {
        MessageBox.Show("删除成功！", "删除");
        ndb.UpdateGird();
    }
}
```

（8）btnSyn_Click 事件处理方法。在 MainForm 窗体中，用手写笔单击 Datagrid 列表控件中欲删除的某条数据记录，然后单击“删除”按钮，选中的记录被成功删除。这时设备端数据库中的数据记录和远程数据库服务器中的数据记录不一致，为了将删除之后记录和远程数据库服务器中的数据记录保持一致，可以通过执行远程数据同步操作来实现。首先调用 SyncNow 方法执行数据同步，然后创建 BindingSource 对象，将 BindingSource 对象和 Datagrid 控件对象作为参数创建业务逻辑层 NoteBLL 类的对象，最后调用 NoteBLL 对象的 UpdateGird 方法，获得 NoteInfo 表的数据记录，并显示在窗体 Datagrid 列表控件中。

具体代码实现如下：

```
private void btnSyn_Click(object sender, EventArgs e)
{
    SyncNow();
    bs = new BindingSource();
    NoteBLL ndb = new NoteBLL(bs, dgidNote);
    ndb.UpdateGird();
}
```

运行效果如图 5-55 所示。

图 5-55　设备端执行同步操作成功

（9）在 SQL Server 2005 的 Management Studio 数据库管理工具中，打开 SQL Server 2005 的 NoteDB 数据库中的 NoteInfo 表，可以查看到刚才在设备端执行同步操作之后，NoteInfo 表中的数据记录和设备端中显示的数据记录一致，这表示设备端和服务器同步操作成功，如图 5-56 所示。

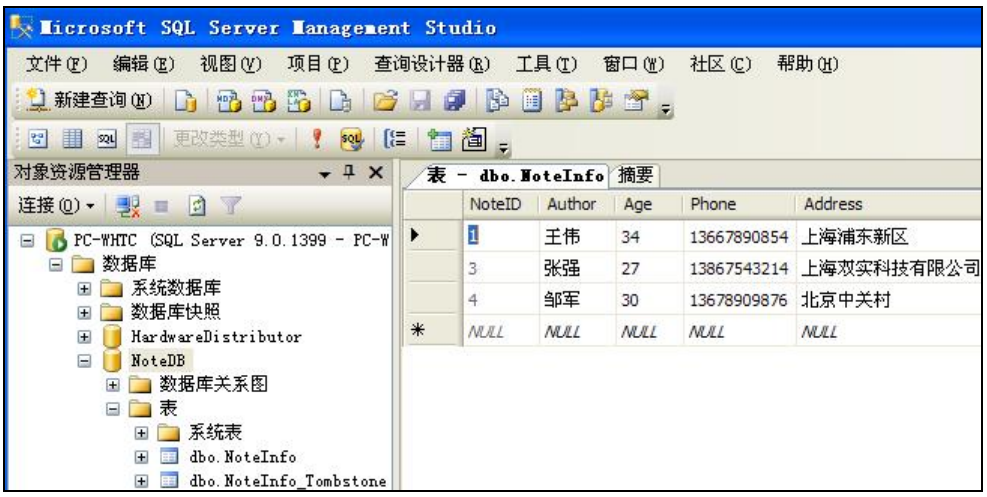

图 5-56　服务器端显示同步操作之后的数据记录

2. 新增数据（AddFrm）窗体的功能实现

（1）AddFrm 窗体界面设计，如图 5-57 所示。

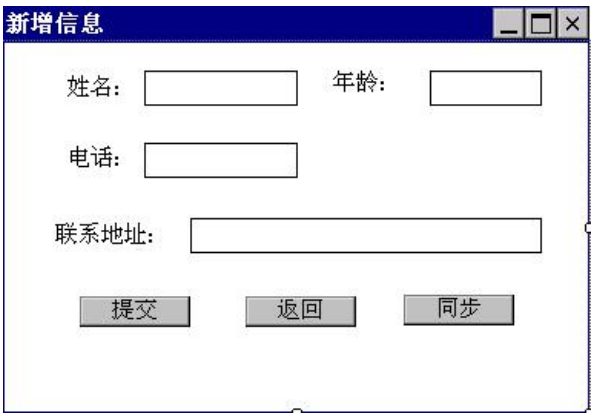

图 5-57　AddFrm 窗体界面设计

（2）AddFrm 窗体代码文件（AddFrm.cs）结构。

```
public partial class AddFrm : Form
{
    private DataGrid Notedgrid;
    private BindingSource Bding;
    private ClientSyncAgent _syncAgent;
    public    AddFrm(BindingSource bds, DataGrid dgrid)
    {
        InitializeComponent();
    }
    private void btnAdd_Click(object sender, EventArgs e)
    {
    }
    private void btnback_Click(object sender, EventArgs e)
    {
    }
    private void btnSync_Click(object sender, EventArgs e)
    {
    }
}
```

（3）AddFrm 构造方法。AddFrm 构造方法是一个带两个参数的构造方法，参数分别为 BindingSource 对象和 Datagrid 控件对象，它是在 AddFrm 窗体对象被创建时自动调用的。

具体代码实现如下：

```
public AddFrm(BindingSource bds, DataGrid dgrid)
{
    Notedgrid = dgrid;
    Bding = bds;
    InitializeComponent();
}
```

（4）btnAdd_Click 事件处理方法。在 AddFrm 窗体的运行界面上，分别输入姓名、年龄、电话号码、联系地址值，单击提交按钮之后，在事件方法处理过程中，首先创建业务逻辑层 Notes 类的对象，将界面上输入的这些值封装进刚创建的 Notes 对象中，然后调用业务逻辑层 NoteBLL 类中带参数的构造方法创建 NoteBLL 类的对象，参数分别为为 BindingSource 对象和 Datagrid 控件对象，创建完 NoteBLL 对象之后，调用 NoteBLL 对象的 InsertNoteData 方法，并传递进 Notes 对象参数，如果新增记录成功，MainForm 窗体会实时显示新增之后的客户信息。

具体代码实现如下：

```
private void btnAdd_Click(object sender, EventArgs e)
{
    Notes ns = new Notes();
    ns.Author = this.txtname.Text;
    ns.Age = Int32.Parse(txtage.Text);
    ns.Phone = txtphone.Text;
    ns.Address = txtaddress.Text;
    NoteBLL ndb = new NoteBLL(Bding, Notedgrid);
    if (ndb.InsertNoteData(ns) > 0)
    {
        MessageBox.Show("新增成功! ", "增加");
        ndb.UpdateGird();
        txtname.Text = "";
        txtage.Text = "";
        txtphone.Text = "";
        txtaddress.Text = "";
    }
}
```

执行完成新增数据操作之后的运行效果如图 5-58 所示。

图 5-58　执行新增数据成功

（5）btnback_Click 事件处理方法。当要关闭 AddFrm 窗体，并返回到 MainForm 窗体时，执行该方法。

具体代码实现如下：

```
private void btnback_Click(object sender, EventArgs e)
{
    this.Close();
}
```

（6）btnSync_Click 事件处理方法。在新增信息窗体中，增加新的数据记录之后，这时设备端数据库中的数据记录和远程数据库服务器中的数据记录不一致，为了将新增之后记录和远程数据库服务器中的数据记录保持一致，可以通过执行远程数据同步操作来实现。这里首先创建 ClientSyncAgent 同步代理对象，它是用于组织客户端与服务器之间的同步过程，然后调用 ClientSyncAgent 对象的 Synchronize 方法完成数据的双向同步操作，如图 5-57 所示。

具体代码实现如下：

```
private void btnSync_Click(object sender, EventArgs e)
{
    Cursor.Current = Cursors.WaitCursor;
    try
    {
        if (null == _syncAgent)
        _syncAgent = new ClientSyncAgent();
        _syncAgent.Synchronize();
        MessageBox.Show("同步完成！ ");
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error during synchronization..." + ex.ToString());
    }
    finally
    {
        Cursor.Current = Cursors.Default;
    }
}
```

执行完成同步操作之后的运行效果如图 5-59 所示。

图 5-59　新增窗体中执行同步操作成功

（7）在如图 5-57 所示的界面中单击“返回”按钮，窗体将返回到如图 5-58 所示的窗体中，这时可以查看到刚刚新增的一条数据记录，如图 5-60 所示。

图 5-60　MainForm 窗体显示的新增记录

（8）在 SQL Server 2005 的 Management Studio 数据库管理工具中，打开 SQL Server 2005 的 NoteDB 数据库中 NoteInfo 表，可以查看到刚才在设备端执行同步操作之后，NoteInfo 表中的数据记录和设备端中显示的数据记录一致，这表示设备端和服务器同步操作成功，如图 5-61 所示。

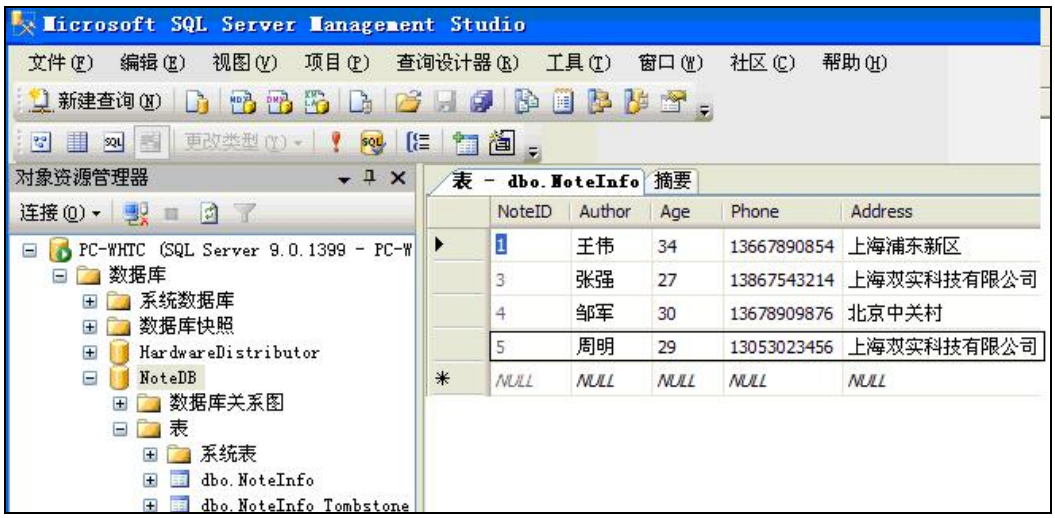

图 5-61　服务器端显示同步操作之后的数据记录

3. 数据更新窗体（EditFrm）的功能实现

（1）EditFrm 窗体界面设计，如图 5-62 所示。

图 5-62　EditFrm 窗体界面设计

（2）EditFrm 窗体代码文件（EditFrm.cs）结构

```
public partial class EditFrm : Form
{
    private string NoteID;
    private DataGrid Notedgrid;
    private BindingSource Bding;
    private ClientSyncAgent _syncAgent;
    public   EditFrm (DataGrid dgrid, BindingSource bds)
    {
        Notedgrid = dgrid;
        Bding = bds;
        InitializeComponent();
    }
    private void FrmModify_Load(object sender, EventArgs e)
    {
    }
    private void btnEdit_Click(object sender, EventArgs e)
    {
    }
    private void btnback_Click(object sender, EventArgs e)
    {
    }
    private void btnSync_Click(object sender, EventArgs e)
    {
    }
}
```

（3）EditFrm 构造方法。EditFrm 构造方法是一个带参数的构造方法，参数分别为 BindingSource 对象和 Datagrid 控件对象，它是在 EditFrm 窗体对象被创建时自动调用的。

具体代码实现如下：

```
public EditFrm(DataGrid dgrid, BindingSource bds)
{
```

```
        Notedgrid = dgrid;
        Bding = bds;
        InitializeComponent();
    }
```

（4）EditFrm_Load 事件处理方法。在 EditFrm 窗体加载事件方法处理过程中，由 BindingSource 对象的 Current 属性获得选中的数据行视图 DataRowView，通过这一数据行视图记录可以分别获得 NoteInfo 表中 NoteID 主键值、姓名、年龄、电话号码以及联系地址值，然后分别显示在窗体文本框上，以方便后面更新这条记录。

具体代码实现如下：

```
    private void EditFrm_Load(object sender, EventArgs e)
    {
        DataRowView Dv = (DataRowView)Bding.Current;
        txtname.Text = Dv["Author"].ToString();
        txtage.Text = Dv["Age"].ToString();
        txtphone.Text = Dv["Phone"].ToString();
        txtaddress.Text = Dv["Address"].ToString();
        NoteID = Dv["NoteID"].ToString();
    }
```

（5）btnEdit_Click 事件处理方法。在 EditFrm 窗体的运行界面上，分别输入想要更新的姓名、年龄、电话号码以及联系地址值。单击“提交”按钮之后，在事件方法处理过程中，首先创建业务逻辑层 Notes 类的对象，将界面上输入的这些更新值封装进刚创建的 Notes 对象中，然后调用业务逻辑层 NoteBLL 类中带参数的构造方法创建 NoteBLL 类的对象，参数分别为为 BindingSource 对象和 Datagrid 控件对象，创建完 NoteBLL 对象之后，调用 NoteBLL 对象的 UpdateNoteDataByNotID 方法，并传递进 Notes 对象参数，如果更新记录成功，MainForm 窗体会实时显示更新之后的客户信息。

具体代码实现如下：

```
    private void btnEdit_Click(object sender, EventArgs e)
    {
        Notes ns = new Notes();
        ns.NoteID = Int32.Parse(NoteID);
        ns.Author = txtname.Text;
        ns.Age = Int32.Parse(txtage.Text);
        ns.Phone = txtphone.Text;
        ns.Address = txtaddress.Text;
        NoteBLL ndb = new NoteBLL(Bding, Notedgrid);
        if (ndb.UpdateNoteDataByNotID(ns) > 0)
        {
            MessageBox.Show("更新成功！", "更新");
            ndb.UpdateGird();
        }
    }
```

编辑信息完成之后的运行效果如图 5-63 所示。

图 5-63　编辑信息执行成功

（6）btnback_Click 事件处理方法。当要关闭 EditFrm 窗体，并返回到 MainForm 窗体时，执行该方法。

具体代码实现如下：

```
private void btnback_Click(object sender, EventArgs e)
{
    this.Close();
}
```

（7）btnSync_Click 事件处理方法。在更新数据窗体中增加新的数据记录之后，这时设备端数据库中的数据记录和远程数据库服务器中的数据记录不一致，为了将更新之后的数据记录和远程数据库服务器中的数据记录保持一致，可以通过执行远程数据同步操作来实现。这里首先创建 ClientSyncAgent 同步代理对象，它是用于组织客户端与服务器之间的同步过程，然后调用 ClientSyncAgent 对象的 Synchronize 方法完成数据的双向同步操作。

具体代码实现如下：

```
private void btnSync_Click(object sender, EventArgs e)
{
    Cursor.Current = Cursors.WaitCursor;
    try
    {
        if (null == _syncAgent)
        _syncAgent = new Sync.ClientSyncAgent();
        _syncAgent.Synchronize();
        MessageBox.Show("同步完成！");
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error during synchronization..." + ex.ToString());
    }
    finally
    {
        Cursor.Current = Cursors.Default;
    }
}
```

执行完成同步操作之后的运行效果如图 5-64 所示。

图 5-64　编辑窗体中执行同步操作成功

（8）在 SQL Server 2005 的 Management Studio 数据库管理工具中，打开 SQL Server 2005 的 NoteDB 数据库中 NoteInfo 表，可以查看到刚才在设备端执行同步操作之后，NoteInfo 表中的数据记录和设备端中显示的数据记录一致，这表示设备端和服务器同步操作成功，如图 5-65 所示。

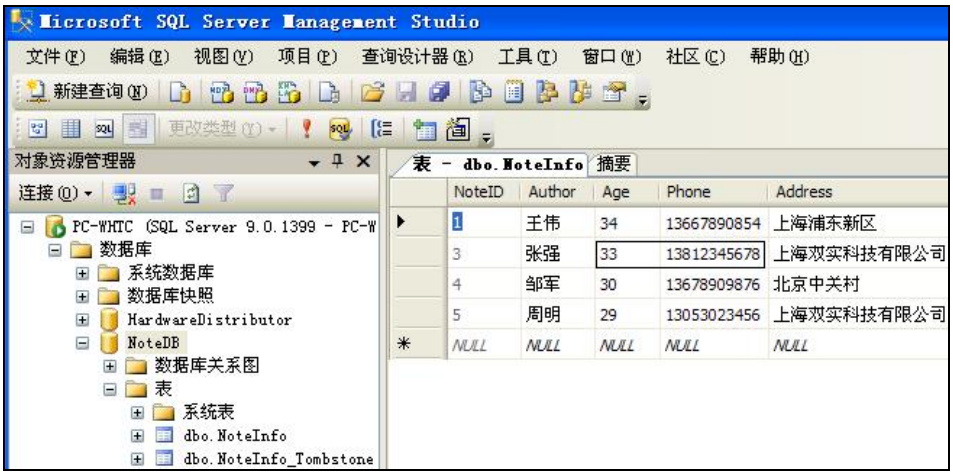

图 5-65　服务器端显示同步操作之后的数据记录

5.5　实验内容：数据库同步应用编程

1. 实验目的

通过本实验，掌握在 VS.NET 2008 平台中利用 Synchronization Services for ADO.NET 同步技术开发 SQL Server Compct 嵌入式数据库程序，使读者能够快速掌握 Windows CE 下的数据库同步编程方法。

2．实验设备

（1）硬件：

PC 机一台

HMI 实验设备一套

（2）软件：

Visual Studio 2008

Microsoft SQL Server 2005

ActiveSync 同步软件

3．实验内容

（1）创建 C#的 Windows CE 嵌入式应用程序工程项目。

（2）搭建 Synchronization Services for ADO.NET 数据同步环境。

（3）设计窗体界面，完成数据显示控件（DataGrid）、数据记录删除 Button 按钮以及数据同步 Button 按钮的事件方法设计。

（4）编程实现移动客户端的 SQL Server Compact 3.5 数据库的数据显示、数据记录删除以及数据同步的功能代码。

（5）利用 ActiveSync 同步软件保持 HMI 设备与开发 PC 机同步通信。

（6）利用网线搭建 HMI 设备与开发 PC 机的网络环境。

（7）将开发 PC 机中编译完成的应用程序在线下载至 HMI 设备中运行。

本章小结

本章首先介绍了有关数据同步的基本概念和三种数据同步方式的功能特性，然后重点介绍 Synchronization Services for ADO.NET 的数据同步技术，并通过一个实例来讲解如何搭建 Synchronization Services for ADO.NET 数据同步环境以及编程实现数据同步功能，最后安排一个有关数据同步的实验，希望读者能够动手练习，以便快速掌握 Synchronization Services for ADO.NET 数据同步的编程方法。

习题五

一、填空题

1．嵌入式数据库的数据同步技术可以有三种方式实现，它们分别为_____、_____、_____。

2．在 Synchronization Services for ADO.NET 中包含一组 DLL，这些 DLL 提供了用于在数据服务和本地存储区之间同步数据的组件，而不是仅仅用于复制数据库及其架构。其中 Microsoft.Synchronization.Data.dll 包含_____、_____和_____；Microsoft.Synchronization.Data.SqlServerCe.dll 包含_____；Microsoft.Synchronization.Data.Server.dll 包含_____。

3．同步表是为每个进行同步的表定义的。它可以设置四种同步方向，它们分别_____、_____、_____、_____。

二、简答题

1．简述并比较嵌入式数据库的三种数据同步技术。
2．简述客户端同步提供程序的主要功能。
3．简述服务器同步提供程序的主要功能。
4．简述数据同步的几种同步方向特性。