www.china-pub.com
ebook

第 9 章　以 Delphi 技术开发 MTS/COM+应用系统　433
9-1　一个典型的场景　434
9-2　使用 TDCOMConnection 建立 MTS/COM+对象　441
9-3　维护事务管理的正确性　444
9-4　MTS/COM+应用系统架构的实现　445
9-5　第二种思考方式　460
9-6　结论　460

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

以 Delphi 技术开发 MTS/COM+应用系统

如何使用 Delphi 快速地开发 MTS/COM+应用系统是非常重要的问题。虽然 Delphi 提供了 MTS 的向导可以帮助程序员开发 MTS/COM+应用系统，但是 Delphi 的向导只是产生一个 MTS/COM+对象。到底要如何借助 Delphi 的技术开发整个应用系统而不是单一的 MTS/COM+组件便非常重要。

本章讨论的重点便是提供一个适当的 MTS/COM+开发架构让程序员能够参考使用。此外，本章也详细地说明了如何结合 Delphi 提供的技术，例如 MIDAS，和 MTS/COM+技术为一体。让程序员能够充分地发挥 Delphi 每一分的潜力，快速而且正确地开发 MTS/COM+应用系统!

本章重点

- ► 一个典型的场景
- ► 使用 TDCOMConncetion 建立 MTS/COM+对象
- ► 维护交易管理的正确性
- ► MTS/COM+应用系统架构的实现
- ► 第二种思考方式
- ► 结论

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

在上一章中我们说明了如何使用 Delphi 开发 MTS/COM+对象，以及能够让我们轻易地存取数据的 MTS/COM+数据模块对象。由于一般的 MTS/COM+应用系统都免不了需要处理数据，许多程序员就容易直接使用 MTS/COM+数据模块作为开发的基础，因为程序员可以直接在 MTS/COM+数据模块中放入 ADO Express 组件来存取数据库。这样做并没有什么不对，但是却可能会发生许多潜在的问题。

本章要讨论的内容是在告诉你如何正确地使用 Delphi 提供的技术来建立 MTS/COM+对象。什么是 Delphi 提供的技术呢？当然主要就是 MIDAS 了，正确地使用 MIDAS 可以帮助程序员快速地开发多层应用系统、MTS/COM+应用系统以及电子商务系统。但是如何正确地结合 MIDAS 和 MTS/COM+便是非常重要的事情了。程序员如果正确地了解 MIDAS 和 MTS/COM+，就能够以数倍的生产力开发 MTS/COM+应用系统；反之则可能会让 MTS/COM+应用系统发生执行不正常的情形。也许让我们从一个典型的系统架构来说明如何从 Delphi 的角度正确地建立 MTS/COM+应用系统，让你快速地了解本章所要强调的内容。

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

9-1、一个典型的场景

图 9-1 是三个范例数据表，它们分别是 Publishers、Books 和 CDTitles。这三个数据表借助 VID 这个字段形成 Master/Detail 的关系。其中 Publishers 储存出版商的数据，Books 储存出版商出版的书籍，而 CDTitles 则储存出版商出版的光盘数据。现在我们的目标就是撰写一个 MTS/COM+应用系统以便能够处理这些数据表中的数据。对于这样的系统场景，程序员应该如何完成它呢？

由于这个系统需要处理数据，因此程序员可能会直接使用 MTS/COM+数据模块，然后在 MTS/COM+数据模块中加入 TADOConnection 连接数据库，再分别使用 TADOQuery/TADOTable 和 TDataSetProvider 组件连接到这三个数据表。此时 MTS 数据模块看起来会类似图 9-2 所示。在图 9-2 中使用了三个 TADOQuery 分别连接到 Publishers、Books 和 CDTitels，而且每一个 TADOQuery 又分别连接一个 TDataSetProvider 组件以便让客户端能够连接使用。此外，这个 MTS/COM+数据模块的事务特性还必须设定为“需要事务”，因为这个 MTS 数据模块会更新数据回数据表中。

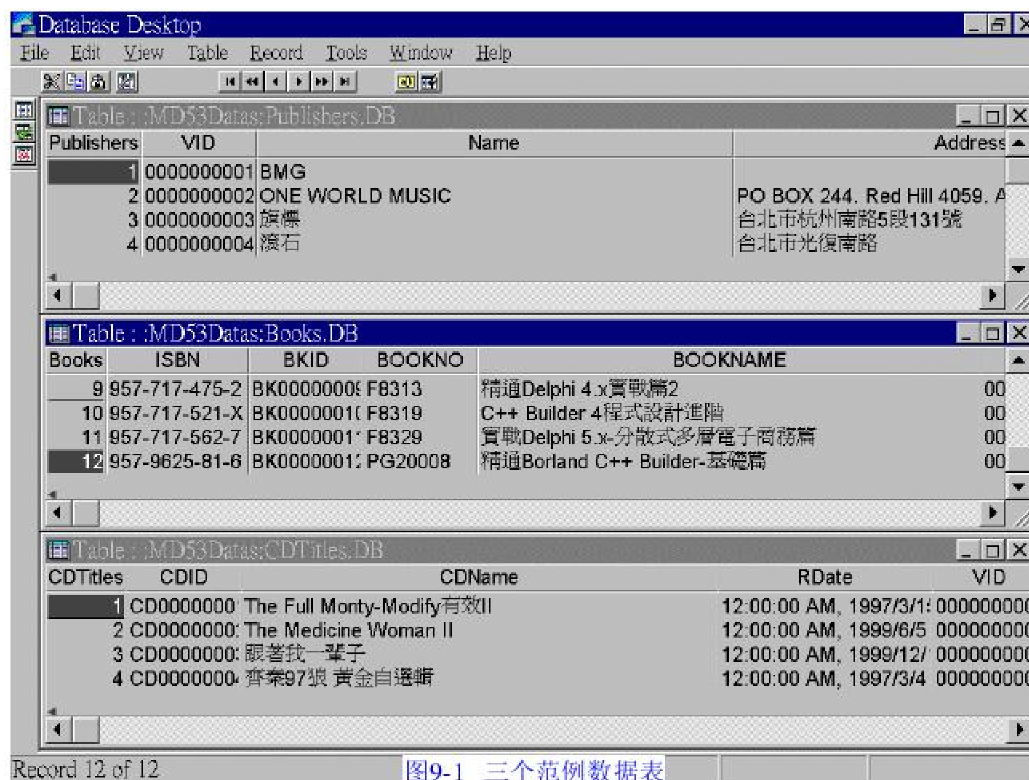

图9-1　三个范例数据表

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

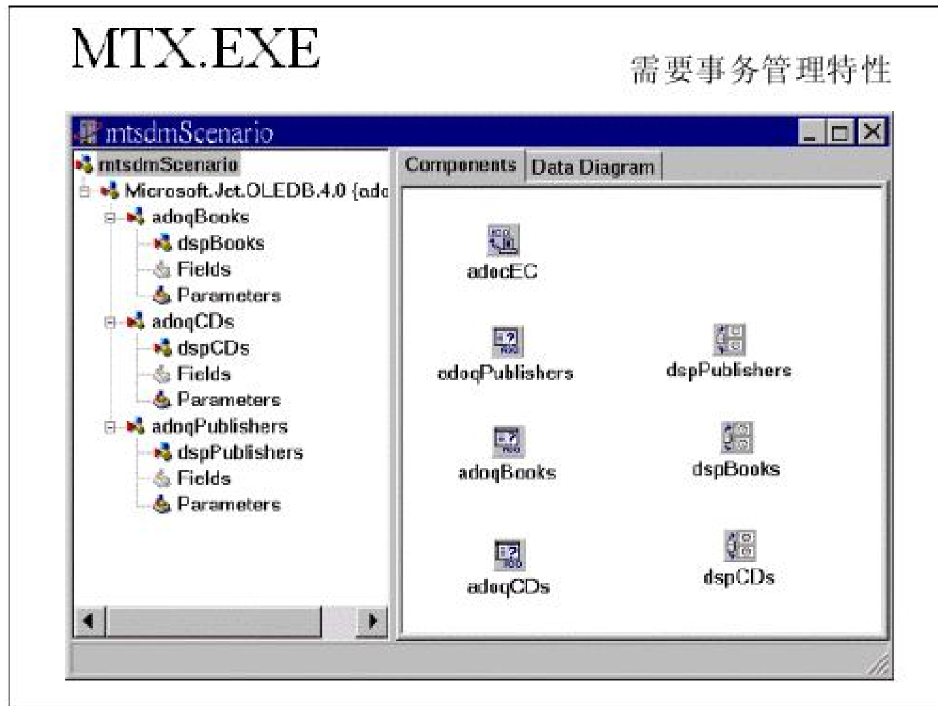

图9-2　三个范例数据表

完成了 MTS/COM+数据模块之后，程序员可以再使用 TDCOMConnection 和 TClientDataSet 组件设计客户端应用程序，并通过这两个组件直接连接到 MTS/COM+数据模块中的 TDataSetProvider 组件以取得数据。最后的系统架构就如图 9-3 所示。

图9-3　范例MTS/COM+应用系统的架构

我相信许多程序员都会设计这种架构的 MTS/COM+应用系统。这样做虽然没有错，而且可以执行，但是这种架构却有许多的问题。在你了解这些问题之后，可能就会使用不同的架构来设计这个场景系统。

那么到底这种架构有什么问题呢？其中最大的问题就是执行效率的问题。这可以从数个不同的角度来看，首先程序员把所有的数据存取组件放在一个 MTS/COM+数据模块中，因此当客户端建立这个 MTS/COM+数据模块时需要花费许多的激活时间。第二个问题是如果客户端只需要使用其中的一个数据表的数据，例如 Books，那么客户端仍然需要花费多余的时间建立不相关的对象。另外的问题就是这个 MTS/COM+数据模块需要修改数据，因此它

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

的事务特性必须设定为“需要事务”。但是如果此时客户端只是需要查询数据，例如在电子商务中，许多 Internet/Intranet 用户大部分的时间是在浏览和查询系统提供的服务和产品，并且一旦客户端建立这个 MTS/COM+数据模块查询数据，那么仍然会激活 MTS/COM+的事务模式。此时，MTS/COM+执行环境和数据库就会开始锁定一些系统资源，然而这是不必要的事情。这样做不但执行效率会受影响，也会降低应用系统可以同时服务的客户端，因为系统锁定了许多不必要的资源。最后，对于 MTS/COM+提供的 Pooling 机制而言，这样设计系统架构也是不好的，程序员应该尽量利用 MTS/COM+提供的数据库连接 Pooling 的功能。因此最好是把每一个 TADOConnection 的连接放在不同的 MTS/COM+数据模块中，因为这三个数据表都位于一个相同的数据库中。详细的原因请参考下一章调整 MTS/COM+执行效率的说明。

为了解决上述说明的许多问题，让我们重新设计一下这个架构。为了充分利用 MTS/COM+的数据库连接 Pooling 功能，并且避免客户端建立不必要的对象，让我们使用三个独立的 MTS/COM+数据模块，其中分别放入 TADOConnection、TADOQuery 和 TDataSetProvider 以连接到不同的数据表。如图 9-4 所示。

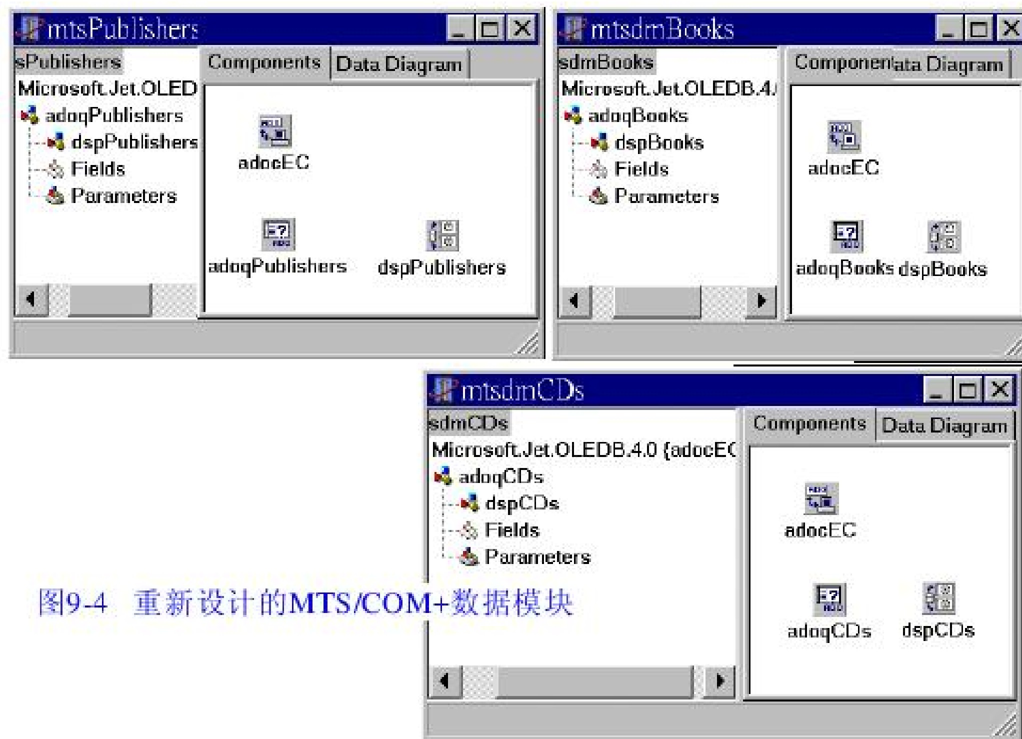

图9-4 重新设计的MTS/COM+数据模块

接着，再让我们引入协调对象的概念，看看它如何帮助我们开发 MTS/COM+应用系统。让我们建立两个 MTS/COM+对象：分别是 QueryObject 和 UpdateObject。其中 QueryObject 负责提供方法让客户端能够查询数据，并使用图 9-4 中的 MTS/COM+数据模块查询客户端需要的数据。由于 QueryObject 只是查询数据，因此它的事务特性可以只设定为支持事务。只要客户端没有事务环境，QueryObject 也就不会产生一个事务环境，这样就不会锁定不必要的资源了。而 UpdateObject 则负责把客户端修改的数据更新回数据表中，它提供了方法，例如 UpdateBooks、UpdateCDTitles 等。UpdateObject 也会分别使用上图的三个 MTS/COM+数据模块来真正地把数据更新回数据表中。由于 UpdateObject 负责更新数据，因此它的事务特性必须设定为“需要事务”。由于 QueryObject 和 UpdateObject 已经控制了事务特性必要的数值，因此图 9-4 中的三个 MTS/COM+数据模块的事务特性只需要设定为“支持事务”

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

即可。现在加入这两个协调对象到范例场景系统中，此时我们可以使用图 9-5 来代表整个系统的架构。

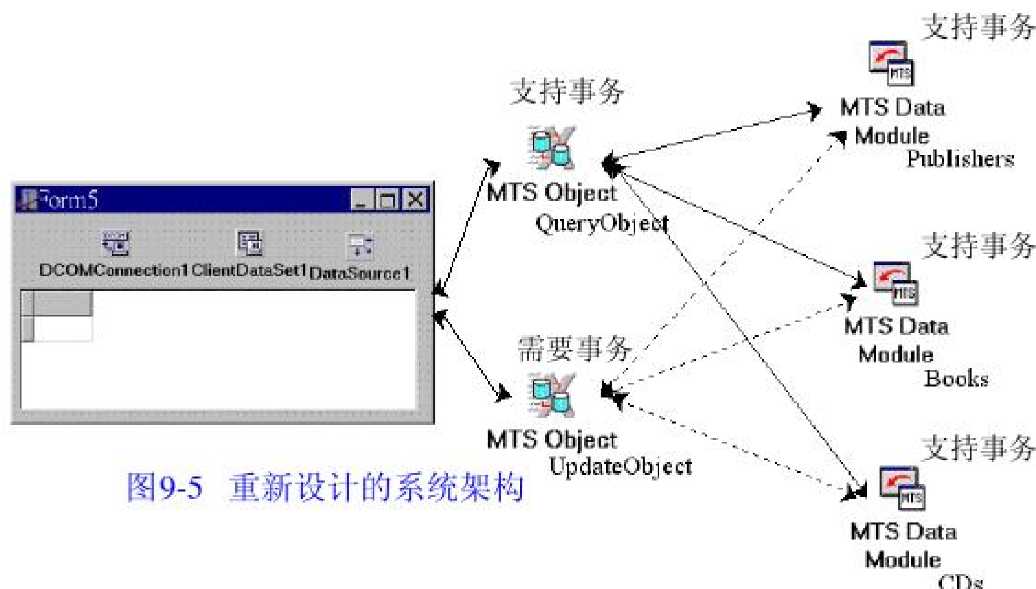

图9-5 重新设计的系统架构

由于加入了 QueryObject 和 UpdateObject，因此当客户端是浏览器查询电子商务的服务和产品时，系统可以调用 QueryObject 提供的服务，通过 MTS/COM+数据模块查询客户端需要的数据。此时 MTS/COM+执行环境和数据库只会锁定最基本的资源，因为 QueryObject 和 MTS/COM+数据模块都没有激活 MTS/COM+的事务模式，因此应用系统可以服务大量的 Internet/Intranet 客户端。但是对于真正购买服务或产品的客户端，或内部 MIS 信息系统的修改数据，系统则可以通过 UpdateObject 对象激活 MTS/COM+事务模式更新数据回数据库中。这样的设计虽然比直接使用单一的 MTS/COM+数据模块来得麻烦，但是无论在执行效率、系统的延展性以及资源的共享性上都比单一的 MTS/COM+数据模块好得多。不过，我们如果从 Delphi 的角度来看这种架构，那么程序员可能还是会觉得不便，因为在使用单一的 MTS/COM+数据模块架构下，程序员可以在客户端直接使用 TDCOMConnection 和 TClientDataSet 组件从 MTS/COM+数据模块取得封装数据的 MIDAS 数据封包。但是在图 9-5 的架构中，由于客户端是直接调用协调对象的方法，而不是直接和 MTS/COM+数据模块互动，因此就无法通过 TDataSetProvider 组件取得封装数据的 MIDAS 数据封包。那么数据该如何显示在客户端的图形用户接口或浏览器中呢？

要解决这个问题并不困难，就如同《Delphi5.x 分布式多层应用系统篇》讨论的一样，程序员只需要把 MIDAS 的数据封包指定给 TClientDataSet 的 Data 属性值，即可以让连接到此 TClientDataSet 的组件显示数据。因此，图 9-5 中的协调对象只需要把它通过 MTS/COM+数据模块取得的数据回传给客户端中的 TClientDataSet 组件不就可以了吗？

图 9-5 中最后一个要解决的问题在于如何让协调对象回传客户端中 TClientDataSet 组件需要的 MIDAS 数据封包。由于协调对象是通过后端的 MTS/COM+数据模块来查询数据，因此只要让协调对象通过 MTS/COM+中的 TDataSetProvider 组件取得数据不就可以取得协调对象需要的 MIDAS 数据封包了吗？要从 TDataSetProvider 取得 MIDAS 数据封包最简单的方法是使用 TDCOMConnection 和 TClientDataSet 组件，因此我们只需要在协调对象中使用这两个组件不就可以从 MTS/COM+数据模块取得数据了吗？那么协调对象要如何使用 TDCOMConnection 和 TClientDataSet 组件连接 MTS/COM+数据模块中的 TDataSetProvider 组件呢？这很简单，让我们在协调对象中建立一个数据模块，再于此数据模块中加入这两个

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

组件，那么就可以连接到 MTS/COM+数据模块中的 TDataSetProvider 组件了。所以我们最后修改过的系统架构会如图 9-6 所示。

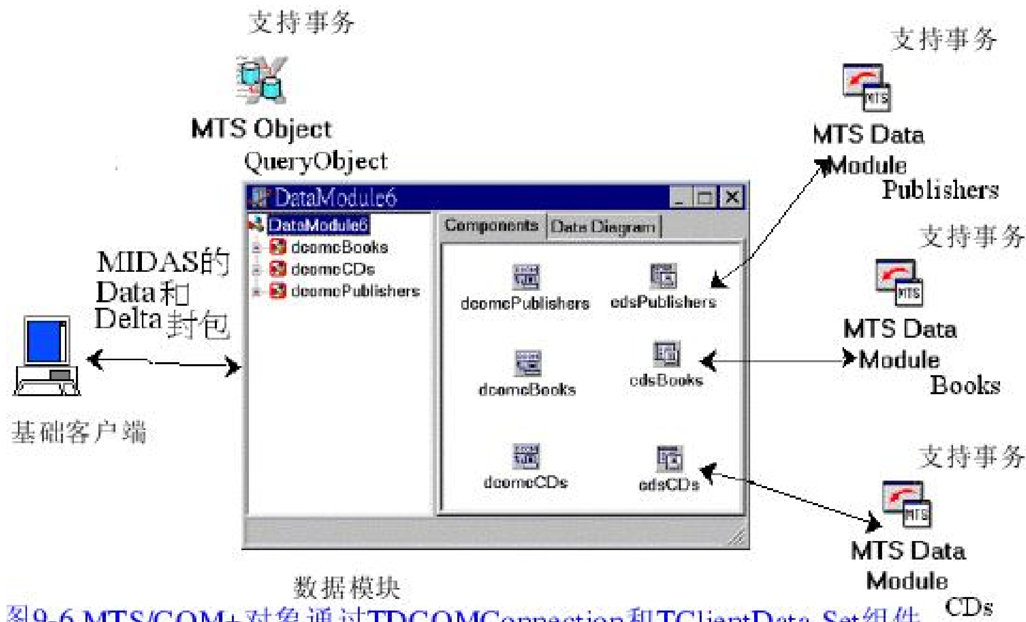

图9-6 MTS/COM+对象通过TDCOMConnection和TClientData-Set组件连接MTS数据模块以取得必要的数据

由于协调对象使用了数据模块以及 TDCOMConnection 和 TClientDataSet 组件，因此程序员只需要设定 TDCOMConnection 的 ComputerName 和 ServerName 即可让 TDCOMConnection 自动地建立 MTS/COM+数据模块。并不需要调用 IObjectContext 接口的 CreateInstance 方法，因此方便了不少。图 9-6 的系统架构不但解决大部分图 9-3 可能发生的问题，在使用上也只比直接让客户端连接 MTS/COM+数据模块麻烦了一点点，因此可以说是同时兼顾了效率、延展性和生产力。

图 9-6 的架构就是结合了 Delphi 的 MIDAS 技术、数据模块、ADO Express 组件以及 MTS/COM+本身为一体的开发方式。让程序员能够完全使用 Delphi 提供的所有功能，而不必像 VB 或 VC++一样调用 MTS/COM+和 ADO 原始的 API。使用这种方式才是程序员以 Delphi 的角度开发 MTS/COM+应用系统，同时这种方式也提供了比 VB 和 VC++高上数倍的生产力。不过要使用这种方式开发 MTS/COM+应用系统之前，我们必须先解决一些问题。

本章的重点除了介绍图 9-6 所示的系统架构之外，另外一个重点便是让你知道如何正确地使用图 9-6 的方式来开发 MTS/COM+应用系统。首先我们必须解决 TDCOMConnection 如何建立 MTS/COM+对象的问题。

9-2、使用 TDCOMConnection 建立 MTS/COM+对象

在上一章中我们已经看到，如何通过 TDCOMConnection 组件建立并且连接远程的 MTS/COM+数据模块。在上一章的范例中，我们是在基础客户端应用程序中使用 TDCOMConnection 组件连接远程 MTS/COM+数据模块，这样做不会有什么问题。但是如果你是像图 9-6 一样在 MTS/COM+执行环境中使用 TDCOMConnection 组件建立其他的 MTS/COM+组件，那么就可能会造成 MTS/COM+应用系统不正常地执行。这是因为 TDCOMConnection 组件在内部是调用 CoCreateInstance(Ex)来建立 COM 对象的，但是从前面讨论 MTS/COM+概念的章节中你应该已经知道了，在 MTS/COM+执行环境中必须通过 IObjectContext 的 CreateInstance 方法来建立 MTS/COM+对象。因此要使用类似图 9-6 的架

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

构来开发 MTS/COM+应用系统，我们首先必须让 TDCOMConnection 组件能够在 MTS/COM+执行环境中以正确的方式建立 MTS/COM+对象。

要解决这个问题我们可以使用两种方式。第一种是从 TDCOMConnection 组件继承一个新的组件，然后程序员在开发 MTS/COM+应用系统时都使用这个新的组件，而不使用 TDCOMConnection 组件。但是这种方式可能会造成许多的不便，而且从 TDCOMConnection 继承新的组件在实现上也比较麻烦，因此我并不准备这么做。第二种方式便是修改 TDCOMConnection 的原始程序，那么程序员可以继续使用 TDCOMConnection 组件。但是程序员必须把修改过的 TDCOMConnection 组件的程序单元放在程序员的项目的目录之下，让程序员的项目连接这个新的程序代码，或编译修改过的程序代码，并且把重新编译过的.DCU 文档拷贝到你的 Delphi5\Lib 目录之下。现在就让我们看看如何修改 TDCOMConnection，让它可以正常地在 MTS/COM+执行环境下建立 MTS/COM+对象。

TDCOMConnection 组件在它的 Connected 属性值被设定为 True 时便会真正地建立它连接的远程 MTS/COM+对象，而此时 TDCOMConnection 是调用它的 DoConnect 方法建立远程对象的。而 DoConnect 则是调用了位于 ComObj 程序单元中的 CreateComObject 来建立远程对象：

```
Procedure TCOMConnection.DoConnect;
begin
    SetAppServer(CreateComObject(GetServerCLSID) as IDispatch);
end;
```

因此我们只需要修改 ComObj 程序单元中的 CreateComObject 和 CreateRemoteComObject 方法即可让 TDCOMConnection 正确地建立 MTS/COM+组件。下面的程序代码是如何修改 ComObj 程序单元中的 CreateComObject 和 CreateRemoteComObject 方法。首先，我们通过调用 GetObjectContext 来判断目前是否是位于 MTS/COM+执行环境中。如果不是，那么 GetObjectContext 就会回传 nil，此时我们就直接使用 CreateComObject 和 CreateRemoteComObject 方法原来的程序代码。如果 GetObjectContext 回传的不是 nil，就表示目前的 TDCOMConnection 是执行在 MTS/COM+执行环境中，那么我们就必须使用 GetObjectContext 回传的接口，即 IObjectContext 接口的 CreateInstance 方法来建立 MTS/COM+对象。

```
Function CreateComObject(const ClassID:TGUID):IUnknown;
begin
  //!!!ModifiedByGordonLi
  If (GetObjectContext=nil) then
    OleCheck(CoCreateInstance(ClassID, nil, CLSCTX_INPROC_SERVER or
                CLSCTX_LOCAL_SERVER, IUnknown, Result))
  Else begin
    GetObjectContext.CreateInstance(ClassID,IUnknown,Result);
  end;
end;

Function CreateRemoteComObject(constMachineName:WideString;
                               constClassID:TGUID):IUnknown;
const
  LocalFlags=CLSCTX_LOCAL_SERVER or CLSCTX_REMOTE_SERVER or
```

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

```
                    CLSCTX_INPROC_SERVER;
  RemoteFlags=CLSCTX_REMOTE_SERVER;
var
  MQI:TMultiQI;
  ServerInfo:TCoServerInfo;
  IID_IUnknown:TGuid;
  Flags,Size:DWORD;
  LocalMachine:array[0..MAX_COMPUTERNAME_LENGTH] of char;
begin
  //!!!ModifiedByGordonLi
```

```
If (GetObjectContext=nil) then
begin
  if @CoCreateInstanceEx=nil then
```

```
  raise Exception.CreateRes(@SDCOMNotInstalled);
  FillChar(ServerInfo,sizeof(ServerInfo),0);
  ServerInfo.pwszName:=PWideChar(MachineName);
  IID_IUnknown:=IUnknown;
  MQI.IID:=@IID_IUnknown;
  MQI.itf:=nil;
  MQI.hr:=0;
  {If aMachine Name is specified check to see if it the local machine.
  If it isn't, do not allow Local Servers to beused.}
  If Length(MachineName)>0 then
  begin
    Size:=Sizeof(LocalMachine);//Win95ishypersensitivetosize
    If GetComputerName(LocalMachine,Size) and
                  (AnsiCompareText(LocalMachine,MachineName)=0) then
      Flags:=LocalFlags
    else
      Flags:=RemoteFlags;
   end
   else
     Flags:=LocalFlags;
  OleCheck(CoCreateInstanceEx(ClassID,nil,Flags,@ServerInfo,1,@MQI));
  OleCheck(MQI.HR);
  Result:=MQI.itf;
end
```

```
else
begin
GetObjectContext.CreateInstance(ClassID,IUnknown,Result);
```

请把这个修改过的 ComObj.Pas 放到你的项目目录下，让它和你的项目连接在一起，或把它编译之后放到 Delphi5\Lib 目录下。但是请注意，当你的项目是使用 Package 形式编译和连

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

接时，最好是把 ComObj.Pas 放到你的项目目录下。现在你就可以放心地在一般的 COM/DCOM 应用程序或 MTS/COM+应用系统中使用 TDCOMConnection 组件了。

9-3、维护事务管理的正确性

除了修改 TDCOMConnection 组件之外，在 Delphi5 中还有另外一个地方需要注意的，那便是 MTS/COM+的事务状态。由于 Delphi 为了同时管理 BDE/IDAPI、ADO 以及 Third-Party 提供的组件的 Two-PhaseCommit 的功能，因此在 Delphi5 中以一个接口统一提供了数据库修改、执行和事务管理的功能，那就是 IProviderSupport 接口。

```
IProviderSupport=interface
  Procedure PSEndTransaction(Commit:Boolean);
  ...
  procedure PSStartTransaction;
  ...
end;
```

在 IProviderSupport 接口中有两个方法：PSStartTransaction 和 PSEndTransaction，都和事务管理有关。这两个方法会实际地调用 BDE/IDAPI 的 StartTransactin 和 EndTransaction，以及 ADO 的 BeginTrans、CommitTrans 或 RollbackTrans 等的方法。然而我们知道，在 MTS/COM+应用系统中 MTS/COM+对象是通过事务特性让 MTS/COM+执行环境控制事务管理的状态，而不应该由个别的 MTS/COM+对象自己调用这些 API。如果 MTS/COM+对象这样做的话，很可能会造成 MTS/COM+执行环境事务管理状态的混乱因此，我们也必须修改 Delphi 如何在 MTS/COM+执行环境中控制管理事务的状态。我并不准备详细地说明为什么要在某个地方修改程序代码，因为这牵涉到 VCL 内部的运作机制，这说来话长。程序员只需要知道我们必须修改 Provider 程序单元中的 InternalApplyUpdates 方法。下面便是修改过的程序代码：

```
function TDataSetProvider.InternalApplyUpdates(constDelta:OleVariant;
                        MaxErrors:Integer;outErrorCount:Integer):OleVariant;
var
  TransactionStarted:Boolean;
begin
  CheckDataSet;
  TransactionStarted:=not IProviderSupport(DataSet).PSInTransaction;
  //ModifiedByGordonLi
  if ((TransactionStarted) and (GetObjectContext=nil)) then
    ProviderSupport(DataSet).PSStartTransaction;
    try
      CheckResolver;
      Resolver.FUpdateTree.InitData(DataSet);
      try
        Result:=inheritedInternalApplyUpdates(Delta,MaxErrors,ErrorCount);
      finally
        Resolver.FUpdateTree.InitData(nil);
      end;
    finally
```

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

```
      if ((TransactionStarted)and(GetObjectContext=nil)) then
        IProviderSupport(DataSet).PSEndTransaction((ErrorCount<=MaxErrors)or(MaxError
              s=-1));
    end;
  end;
```

从 InternalApplyUpdates 中你可以看到，我也是通过 GetObjectContext 来判断目前是否在 MTS/COM+执行环境中。如果是的话，就避免调用 IProviderSupport 的 PSStartTransaction 和 PSEndTransaction 方法，而让 MTS/COM+执行环境来管理事务状态。这样就可以让 Delphi 的 BDE/IDAPI、ADOExpress 和 Third-Party 的数据存取组件正确地执行在 MTS/COM+执行环境中。经过这两个修改之后，Delphi 的组件与 MIDAS 技术就可以和 MTS/COM+水乳交融地在一起运作了。Delphi 的程序员可以立即使用 Delphi 提供的技术和组件在一般的主从架构、分布式多层、MTS 和 COM+环境中使用相同的方式来开发应用系统，不再需要花费时间使用不同的方式在不同的系统中使用不同的程序代码来开发应用系统。

9-4、MTS/COM+应用系统架构的实现

经过了前面几节的说明之后，你应该了解了如何使用 Delphi 开发 MTS/COM+的应用系统，修改 ComObj 和 Provider 程序单元能够让程序员很方便地开发正确的 MTS/COM+应用系统。本小节要讨论的内容就是用前面讨论的概念以实际的范例实现出来。本小节的目的是让程序员更熟悉如何使用 Delphi 开发 MTS/COM+应用系统，如何结合数个 MTS/COM+共同开发系统，并且会实际地讨论一些设计的问题，当然这些设计的问题都和实现技巧有关，但是并不会牵涉到对象导向分析/对象导向设计的主题。

现在就让我们实现图 9-6 的架构，看看如何用 Delphi 实际地开发出这种系统架构。首先让我们先实现连接到 Publishers、Books 和 CDTitles 这三个数据表的 MTS/COM+数据模块。请在 Delphi 中建立三个 ActiveX Library 项目，然后在每一个 ActiveX Library 项目中各自建立一个 MTS/COM+数据模块，并且设定每一个 MTS/COM+数据模块的事务模式都是“支持事务”，如图 9-7 所示。

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

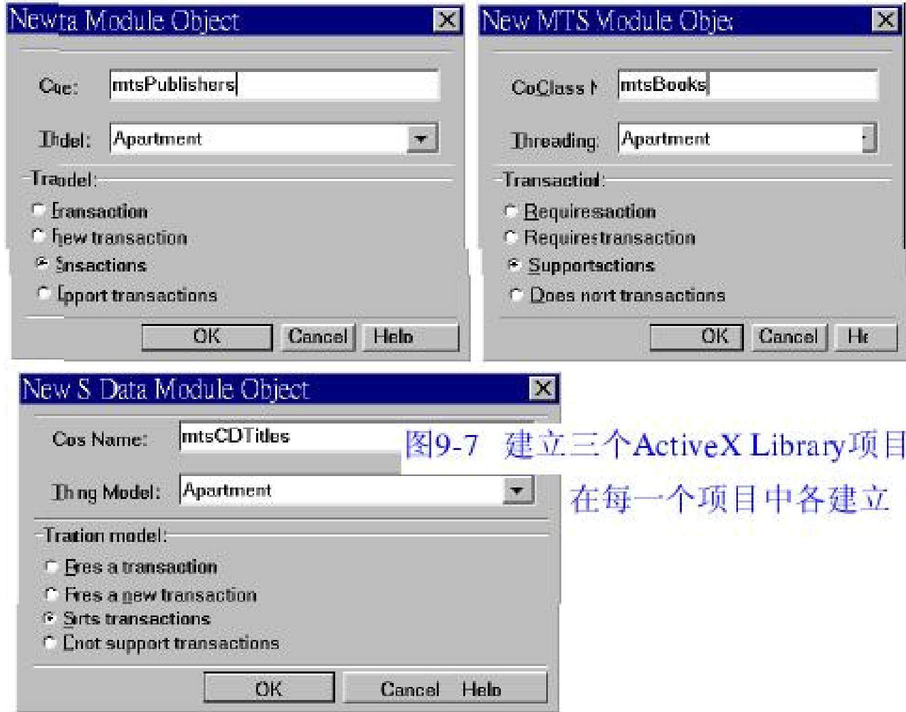

图9-7 建立三个ActiveX Library项目，在每一个项目中各建立

这三个 MTS/COM+对象会同时被稍后开发的查询对象以及更新对象使用。当查询对象使用它们时只是读取数据查询，因此事务模式只需要设定为支持事务；而当更新对象使用它们时却会更新数据，由于更新对象本身会设定为“需要事务”模式，因此这三个 MTS/COM+对象也只需要设定为“支持事务”。这样设计可以减少这些 MTS/COM+对象锁定的系统资源，进而增加效率和延展性，这在前面已经说明过了。在这里我们实际地利用我们对于 MTS/COM+的了解来设计这些 MTS/COM+对象。接着保存这三个项目，并且把它们加入到一个项目组中。图 9-8 是在我的环境中此时的项目组。

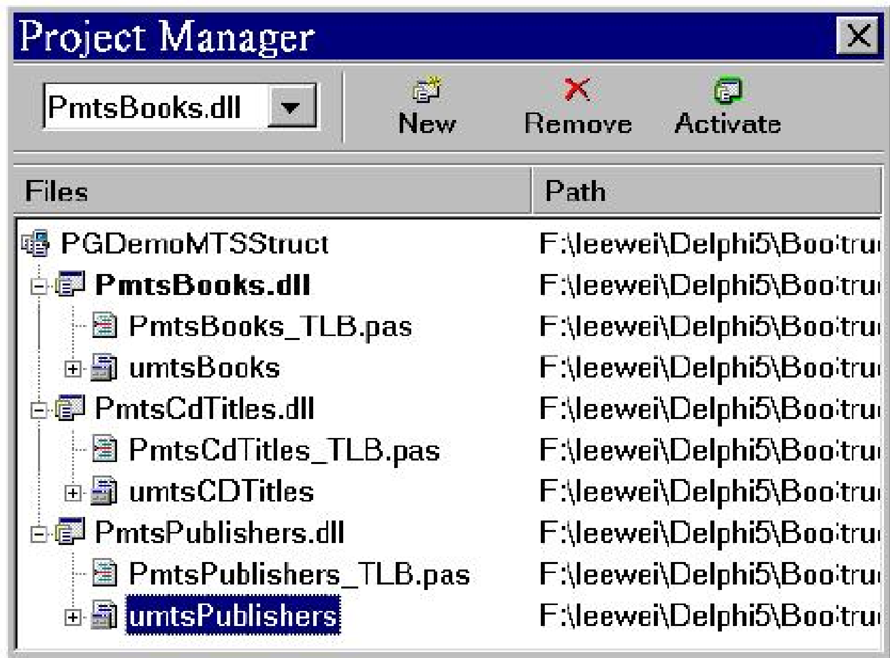

图9-8 范例项目组和三个ActiveX Library项目

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

接着分别在每一个 MTS/COM+数据模块中放入 TADOConnection 组件连接到数据库，再使用 TADOQuery 到三个数据表中查询数据(设定 TADOQuery 组件的 SQL 属性值为 Select * from TableName)，最后再加入 TDataSetProvider 组件，并且设定它的 DataSet 属性值为 MTS/COM+数据模块中的 TADOQuery 组件。图 9-9 是这三个 MTS/COM+数据模块在我的开发环境中的情形。现在我们就可以把三个 MTS/COM+对象加入到 MTS/COM+软件套件中了。请点选 Delphi 集成开发环境中 Run|InstallMTSObject...菜单，并且把这些 MTS/COM+数据模块加入到一个 MTS/COM+软件套件中。例如图 9-10 显示了我把 mtsBooksObj 这个 MTS/COM+数据模块加入到一个称为 MD53StructDemo 的软件套件中。

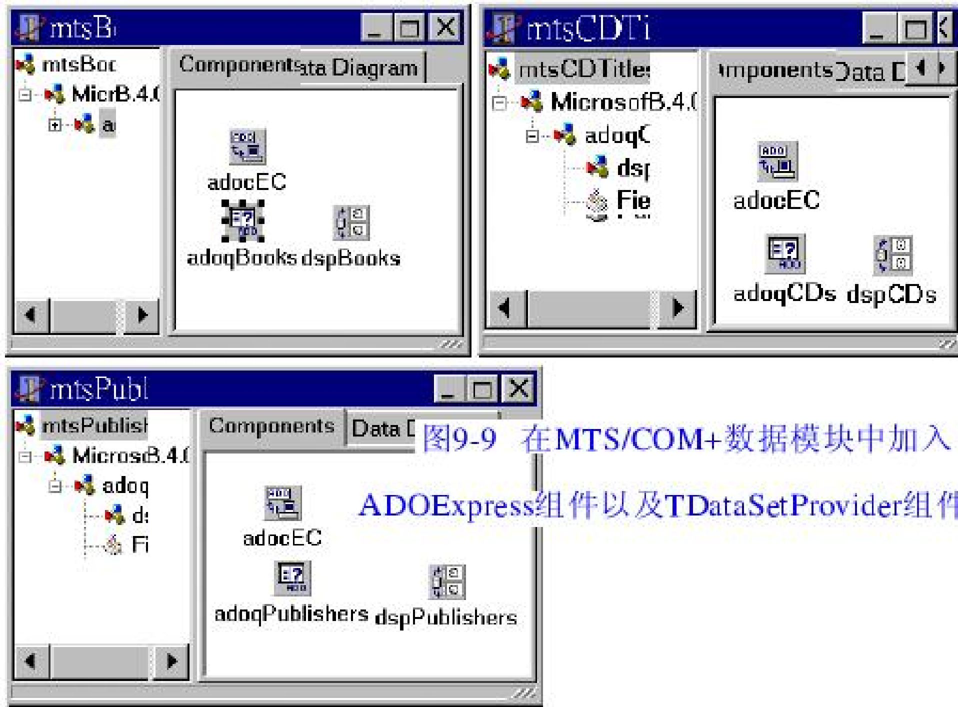

图9-9 在MTS/COM+数据模块中加入 ADOExpress组件以及TDataSetProvider组件

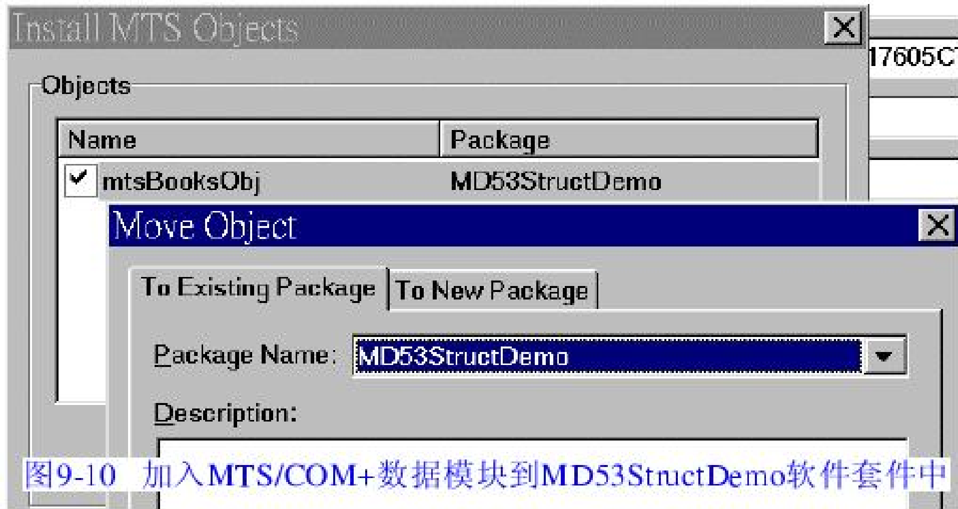

图9-10 加入MTS/COM+数据模块到MD53StructDemo软件套件中

接下来就是实现协调对象—MTS/COM+查询对象和 MTS/COM+更新对象的阶段了。

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

1、实现 MTS/COM+查询对象

图 9-5 中的 MTS/COM+查询对象是一个 MTS/COM+对象。由于这个 MTS/COM+对象的主要功能只是提供客户端查询数据使用的，因此它的事务模式只需要设定为支持事务即可。请在 Delphi 中建立一个新的 ActiveX Library 项目，再于项目中建立一个 MTS/COM+对象。如图 9-11 所示。

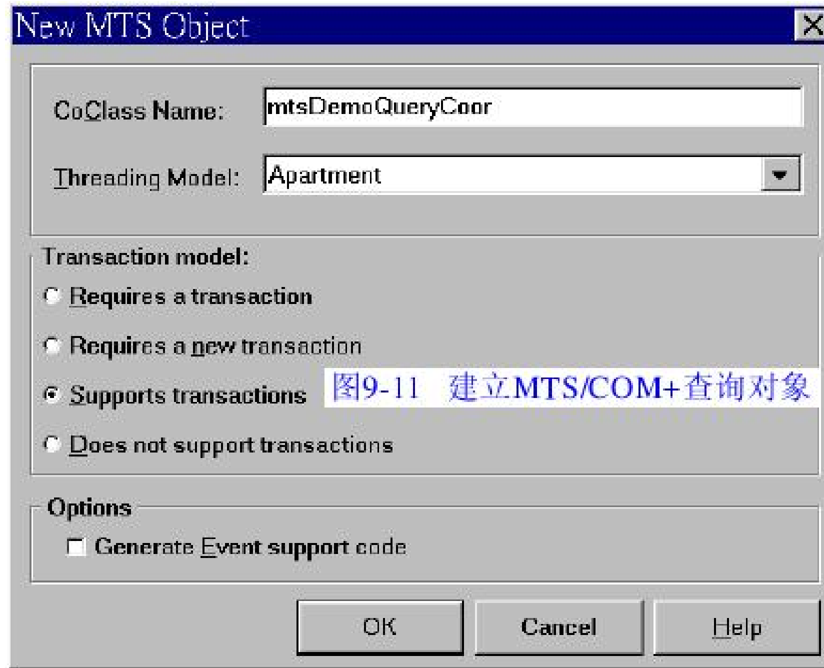

图9-11 建立MTS/COM+查询对象

接着就如图 9-6 所示，让我们在 MTS/COM+查询对象中加入一个数据模块，这样我们就可以在数据模块中直接使用 TDCOMConnection 和 TClientDataSet 组件来建立和连接刚才实现的 MTS/COM+数据模块，而不需要撰写 IObjectContext.CreateInstance 等的程序代码。而且通过 TDCOMConnection 组件我们可以直接取得 MIDAS 的数据封包，也就不需要再撰写复杂的程序代码来处理数据，这正是我们使用 MIDAS 的原因之一。因此请点选 File|New...，并且选择建立一个数据模块，然后在数据模块中加入三个 TDCOMConnection 组件，分别设定每一个 TDCOMConnection 组件连接到刚才实现的 MTS/COM+数据模块。再加入三个 TClientDataSet 组件，连接到 MTS/COM+数据模块中的 TDataSetProvider 组件以取得数据。图 9-12 显示了此时 MTS/COM+查询对象中的数据模块的情形，最后请记得设定 TDCOMConnection 组件的 Connected 属性值为 True，以便建立并且连接 MTS/COM+数据模块。

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

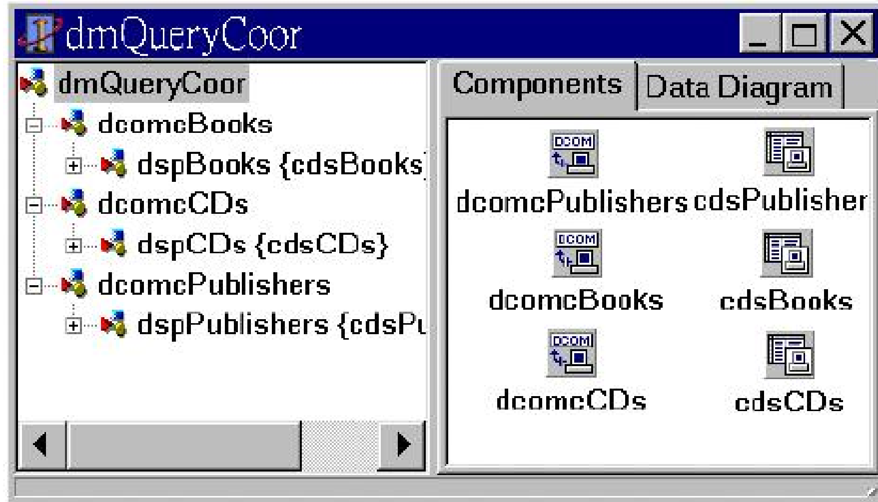

图9-12 在MTS/COM+查询对象中建立一个数据模块，以便使用TDCOMConnection和TClientDataSet组件从MTS/COM+数据模块取得查询的数据

现在 MTS/COM+查询对象已经可以通过 MTS/COM+数据模块取得数据了，接着这个MTS/COM+查询对象必须加入方法以便让客户端调用来查询数据。请激活可视化 Type Library 编辑器，并且如图 9-13 所示在 MTS/COM+查询对象的接口中加入 GetBook、GetCDTitles 和 GetPublishers 三个方法。这三个方法可以分别回传 Books、CDTitles 和 Publishers 数据表中的数据。当然也可以加入其他的方法，例如 QueryBooksByAuthor 和 QueryCDTitlesBySinger 等的方法。不要忘记这三个方法参数的定义。例如图 9-13 显示了 GetBooks 会从客户端接受一个 Variant*类型的参数，并且会把查询的结果数据放在这个参数之中，再回传给客户端。

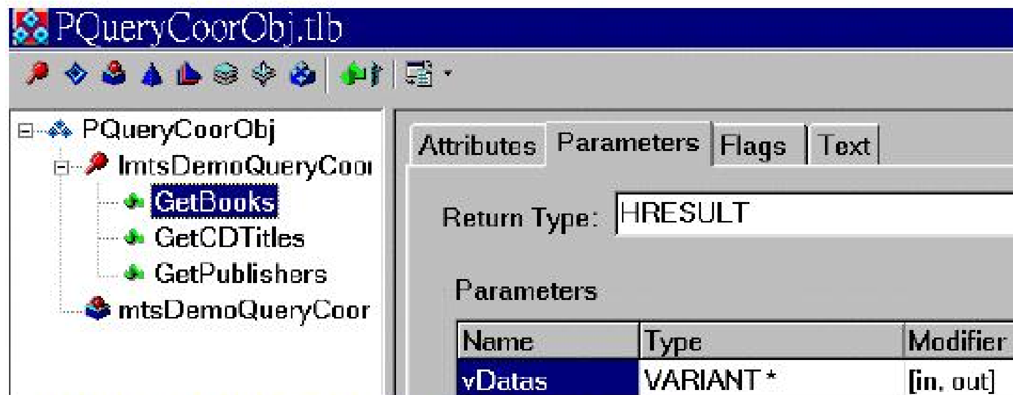

图9-13 在MTS/COM+查询对象中加入方法以便让客户端应用程序调用来查询数据

MTS/COM+查询对象的最后一个实现步骤便是撰写程序代码。首先我们必须在 mtsDemoQueryCoor 程序单元中声明一个如图 9-11 所示的数据模块的变量如下所示：

FMyDM:TdmQueryCoor;

当然，mtsDemoQueryCoor 程序单元必须在它的 uses 句子中加入如图 9-11 所示的数据模块的程序单元：

uses

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

```
ActiveX, MtsObj, Mtx, ComObj, PQueryCoorObj_TLB, StdVcl,Forms,
udmQueryCoor;
```

有了 FMyDM 变量之后，我们必须在这个 MTS/COM+查询对象被建立之后再动态地建立数据模块，因此我们必须为 MTS/COM+查询对象的类别加入 Destroy 和 Initialize 这两个重载的方法。然后在 Destroy 析构函数中释放动态建立的数据模块，而在 Initialize 方法中动态地建立数据模块：

```
destructor TmtsDemoQueryCoor.Destroy;
begin
  inherited;
  FMyDM.Free;
end;
procedure TmtsDemoQueryCoor.Initialize;
begin
  inherited;
  FMyDM:=TdmQueryCoor.Create(Forms.Application);
end;
```

最后的程序代码便是实现刚才在 Type Library 编辑器中定义的方法。当客户端调用这三个方法时，我们只需要开启数据模块中的 TClientDataSet 组件，再把 TClientDataSet 组件的 Data 属性值指定给客户端传入的参数即可。实现这三个方法的程序代码之所以这么简单的原因便在于我们使用了 MIDAS 技术。当然，为了尽早释放每一个方法使用的资源，我们必须调用 SetComplete/SetAbort 方法，而且在取得数据之后立刻关闭 TClientDataSet 组件。

```
Procedure TmtsDemoQueryCoor.GetBooks(varvDatas:OleVariant);
begin
  try
    try
      FMyDM.cdsBooks.Active:=True;
      vDatas:=FMyDM.cdsBooks.Data;
      SetComplete;
    except
      SetAbort;
    end;
  finally//wrapup
    FMyDM.cdsBooks.Active:=False;
  end;//try/finally
end;

procedure TmtsDemoQueryCoor.GetCDTitles(varvDatas:OleVariant);
begin
  try
    try
      FMyDM.cdsCDs.Active:=True;
      vDatas:=FMyDM.cdsCDs.Data;
      SetComplete;
```

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

```
        except
          SetAbort;
        end;
      finally//wrapup
        FMyDM.cdsCDs.Active:=False;
      end;//try/finally
    end;

    procedure TmtsDemoQueryCoor.GetPublishers(varvDatas:OleVariant);
    begin
      try
        try
          FMyDM.cdsPublishers.Active:=True;
          vDatas:=FMyDM.cdsPublishers.Data;
          SetComplete;
        except
          SetAbort;
        end;
      finally
        FMyDM.cdsPublishers.Active:=False;
      end;
    end;
```

现在我们就可以编译这个 MTS/COM+查询对象，并且把它安装到 MD53StructDemo 软件套件之中。接着让我们立刻撰写一个范例客户端应用程序，调用 MTS/COM+查询对象，看看它是否能够正确地工作。图 9-14 便是范例客户端应用程序执行的画面。从图 9-14 中我们可以看到，当我们点选表格中的“取得 Publishers”、“取得 Books”和“取得 CDTitles”按钮时，果然可以通过 MTS/COM+查询对象从 MTS/COM+数据模块中取得我们需要的数据。因此 MTS/COM+查询对象和三个 MTS/COM+数据模块对象可以正确地合作并提供基础客户端查询的数据。

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

图9-14　基础客户端应用程序执行的画面

那么“取得 Publishers”、“取得 Books”和“取得 CDTitles”按钮到底是如何取得数据的呢？由于这三个按钮都是调用 MTS/COM+查询对象以取得数据，因此我们只需要看其中一个方法的实现程序代码即可了解。下面是“取得 Publishers”按钮的 OnClick 事件处理程序：

```
procedure TForm7.btnPublishersClick(Sender:TObject);
var
  qCoor:ImtsDemoQueryCoor;
  vDatas:OleVariant;
begin
  qCoor:=ComtsDemoQueryCoor.CreateRemote('Gordon');
  qCoor.GetPublishers(vDatas);
  cdsPublishers.Data:=vDatas;
end;
```

从上面的程序代码我们可以看到，“取得 Publishers”按钮的 OnClick 事件处理程序首先调用了 MTS/COM+查询对象的 wrapper 类别 ComtsDemoQueryCoor 以建立 MTS/COM+查询对象，然后调用它的 GetPublishers 方法取得查询数据的 MIDAS 数据封包。最后再直接把 MIDAS 数据封包指定给表格中的 TClientDataSet 组件—cdsPublishers—的 Data 属性值即可显示查询的数据，相当简单。

查询数据是一般应用系统一定要提供的功能，但是更新数据回数据库的功能更为重要。接下来再让我们说明如何实现 MTS/COM+更新对象。

2、实现 MTS/COM+更新对象

首先如同前面实现 MTS/COM+查询对象一样，请先建立一个 ActiveX Library 项目，然后在项目中建立一个 MTS/COM+对象。但是由于这个对象是负责更新数据的，因此设定它的事务模式为“需要事务”，如图 9-15 所示：

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

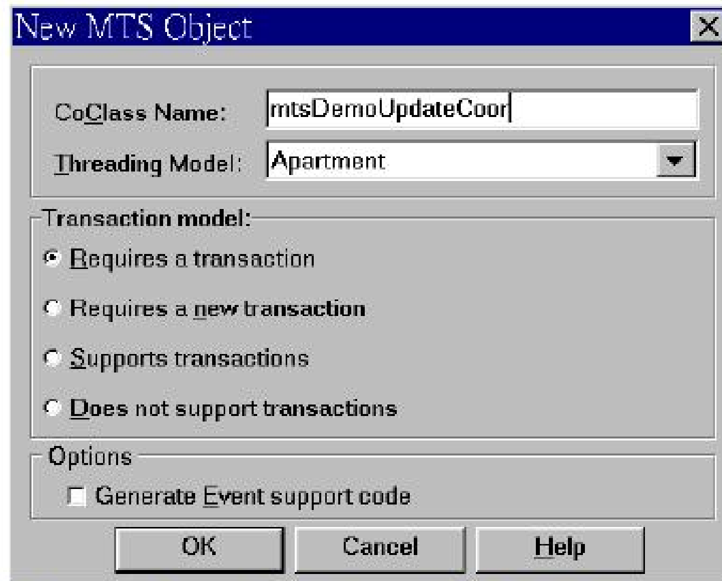

图9-15　在ActiveX Library项目中建立MTS/COM+对象

接着再于项目中建立一个数据模块，并且在数据模块中加入三个 TDCOMConnection 组件，分别连接到前面实现的三个 MTS/COM+数据模块。如图 9-16 所示。

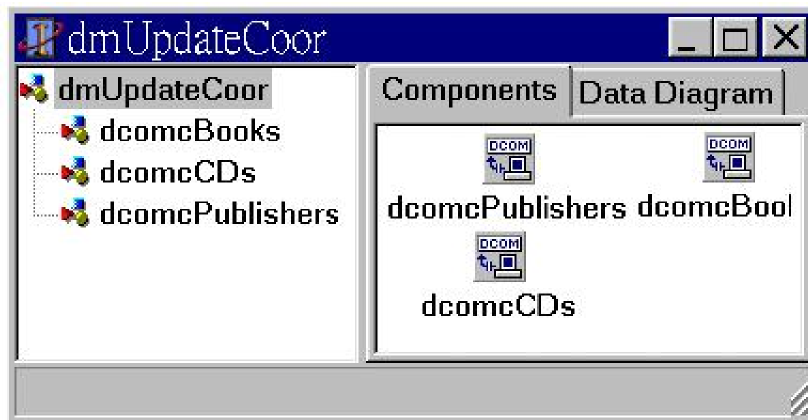

图9-16 在MTS/COM+更新对象的数据模块中加入TDCOMConnection组件以建立和连接MTS/COM+数据模块

为什么 MTS/COM+更新对象只使用了 TDCOMConnection 组件连接 MTS/COM+数据模块，而不像 MTS/COM+查询对象也使用 TClientDataSet 组件呢？这是因为这个 MTS/COM+更新对象是从客户端接收更新数据的 MIDAS 数据封包。当它收到这个 MIDAS 封包之后，MTS/COM+更新对象只要再把 MIDAS 数据封包直接传递给 MTS/COM+数据模块更新数据即可。MTS/COM+更新对象本身并不再需要使用 TClientDataSet 组件暂时储存任何数据，这样可以加快它的执行效率。由于 MTS/COM+更新对象也使用了数据模块，因此它也和 MTS/COM+查询对象一样需要在析构函数和 Initialize 方法中释放和建立数据模块对象：

```
destructor TmtsUpdateCoorObj.Destroy;
begin
```

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

```
    inherited;
    FMyDM.Free;
  end;
  procedureTmtsUpdateCoorObj.Initialize;
  begin
    inherited;
    FMyDM:=TdmUpdateCoor.Create(Forms.Application);
  end;
```

接着激活可视化 Type Library 编辑器，为这个 MTS/COM+更新对象定义方法以便让客户端调用。图 9-17 显示了 MTS/COM+更新对象对外界提供的三个方法。此外，每一个更新数据的方法都接受三个参数：第一个参数 vDatas 是需要更新的数据；第二个参数 iMaxError 是客户端可以接受的更新错误的笔数；最后一个参数是[in,out]的参数，这个参数会由 MTS/COM+数据模块更新数据完毕之后填入在更新数据回数据库的过程中到底发生了多少笔错误。事实上，这三个参数的意义和使用的方法和 IAppServer 接口的 AS_ApplyUpdates 方法是一样的。

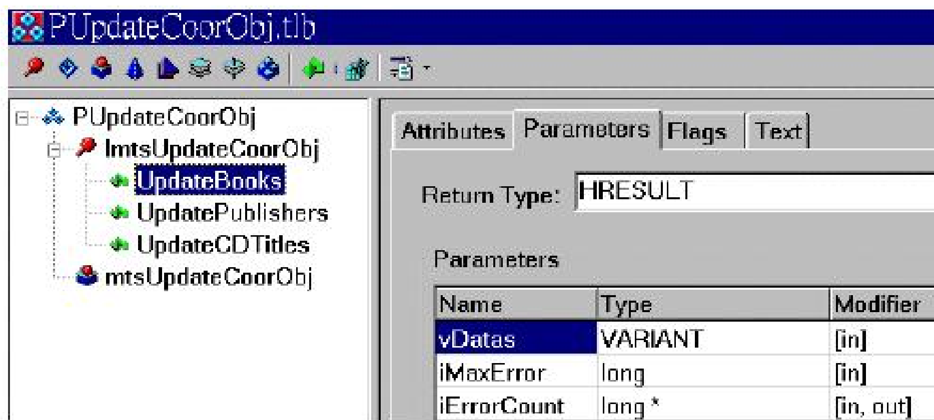

图9-17　在MTS/COM+更新对象中加入方法

下面的程序代码显示了这个 MTS/COM+更新对象如何把数据真正更新回数据库中，你可以看到，MTS/COM+更新数据通过它的数据模块中的 TDCOMConnection 组件连接到 MTS/COM+数据模块，再调用 MTS/COM+数据模块的 UpdateDatas 方法以更新数据。最后根据状态来调用 SetComplete 和 SetAbort 以便释放 MTS/COM+更新对象占用的资源。

```
  Procedure TmtsUpdateCoorObj.UpdateBooks(vDatas:OleVariant;iMaxError:Integer;
                       variErrorCount:Integer);
  begin
    try
      FMyDM.dcomcBooks.AppServer.UpdateDatas(vDatas,iMaxError,iErrorCount);
      SetComplete;
    except
      SetAbort;
    end;
  end;
  procedure TmtsUpdateCoorObj.UpdateCDTitles(vDatas:OleVariant;iMaxError:Integer;
                       variErrorCount:Integer);
  begin
    try
```

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

```
        FMyDM.dcomcCDs.AppServer.UpdateDatas(vDatas,iMaxError,iErrorCount);
        SetComplete;
      except
        SetAbort;
      end;
    end;

    procedure TmtsUpdateCoorObj.UpdatePublishers(vDatas:OleVariant;iMaxError:Integer;
                        variErrorCount:Integer);
    begin
      try
        FMyDM.dcomcPublishers.AppServer.UpdateDatas(vDatas,iMaxError,iErrorCount);
        SetComplete;
      except
        SetAbort;
      end;
    end;
```

从上面的程序代码中我们可以知道，在每一个 MTS/COM+数据模块中必须定义一个方法 UpdateDatas，让 MTS/COM+更新对象调用，以便更新数据。这是因为 MTS/COM+更新对象不知道如何更新数据，这个工作应该由单个的 MTS/COM+数据模块来进行，因此 MTS/COM+更新对象调用了 MTS/COM+数据模块的 UpdateData 方法来实际地更新数据。这种设计除了前面说明的可以增加 MTS/COM+应用系统的资源使用率和执行效率之外，也非常符合对象导向的设计。虽然在这个范例中 MTS/COM+更新对象似乎除了调用 MTS/COM+数据模块更新数据之外并没有做其他的工作，因此看起来似乎是多余的。但是你不要忘了，即使是在这个简单的范例中 MTS/COM+更新对象也具备了激活事务管理的功能，因此它能够让更新数据的过程保护在 Two-PhaseCommit 的功能之中。在实际的应用系统中，MTS/COM+更新对象可以再执行安全管理以及执行状态登录等的工作。

完成这个范例的最后一个步骤是再为每一个 MTS/COM+数据模块加入一个 UpdateDatas 方法，让 MTS/COM+更新对象调用以便把数据真正更新回每一个 MTS/COM+数据模块代表的数据表中。因此请你在 Delphi 的集成开发环境中激活前面实现的 MTS/COM+数据模块，再激活可视化 TypeLibrary 编辑器，然后加入这个方法。例如，图 9-18 便是在 mtsPublishers 这个 MTS 数据模块中加入 UpdateDatas 的画面。

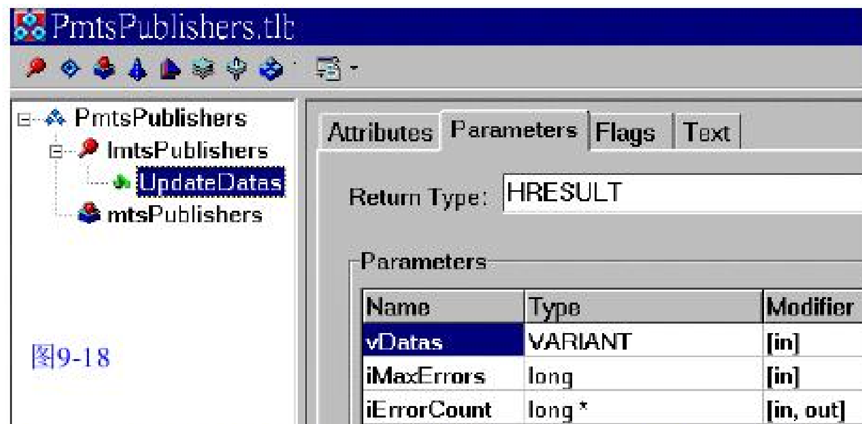

图9-18

在每一个MTS/COM+数据模块对象中加入方法让MTS更新对象调用

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

当然，MTS/COM+数据模块也必须实现 UpdateDatas 方法。由于 MTS/COM+数据模块中已经使用了 TDCOMConnection、TClientDataSet 和 TDataSetProvider 组件，因此我们只需要简单地调用 TDataSetProvider 组件的 ApplyUpdates 方法更新即可。下面就是 mtsPublishers 实现 UpdateDatas 的程序代码：

```
procedure TmtsPublishers.UpdateDatas(vDatas:OleVariant;iMaxErrors:Integer;
                             variErrorCount:Integer);
begin
  try
  //statementstotry
    dspPublishers.ApplyUpdates(vDatas,iMaxErrors,iErrorCount);
    SetComplete;
  except
    SetAbort;
  end;//try/except
end;
```

从这个范例中可以看到我们如何融合 Delphi 的 MIDAS 和 MTS/COM+程序设计为一体。由于 MIDAS 的数据封包可以包含新增的数据、更新的数据以及删除的数据，因此上面的范例可以简单地声明一个 UpdateDatas 方法就可以更新各种的数据回数据库中，程序员不再需要像 VB 或 VC++一样撰写更新各种类型数据的复杂又无趣的程序代码。MIDAS 不但已经实现了这些功能，还能够更新 Master-Detail 类型的数据，因此我们直接使用 MIDA 不但可以提高生产力，而且也不容易出错。

在上面的 MTS/COM+更新对象中，它使用 TDCOMConnection 对象并且需要 MTS/COM+数据模块定义一个更新数据的方法让 MTS/COM+更新对象调用。当然你也可以不这样做，可以直接在 MTS/COM+更新对象中取得 MTS/COM+数据模块的 IAppServer 接口，再直接调用 IAppServer 接口的 AS_Applyupdates 方法。例如，我们可以重新撰写 MTS/COM+数据模块的 UpdateBooks 如下:

```
procedure TmtsUpdateCoorObj.UpdateBooks(vDatas:OleVariant;iMaxError:Integer;
                 variErrorCount:Integer);
var
  ServerObj:IAppServer;
  OwnerData:OleVariant;
begin
  try
    ServerObj:=FMyDM.dcomcBooks.AppServer as IAppServer;
    IAppServer.AS_Applyupdates('dspBooks',vDatas,iMaxError,iErrorCount,OnwerData);
    SetComplete;
  except
    SetAbort;
  end;
end;
```

使用这种方法就不需要在 MTS/COM+数据模块中再定义方法让 MTS/COM+更新对象调用。不过这种方法也有一些问题，那就是 MTS/COM+更新对象必须在程序代码中写死要使用的 TDataSetProvider 组件的名称。要解决这个问题，你可以使用 IAppServer 接口中的 AS_GetProviderNames 来弹性地取得 MTS 数据模块中的 TDataSetProvider 组件的名称。在稍

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com

后的章节中将会讨论更多的实现程序代码。

9-5、第二种思考方式

除了前面讨论的方式之外，如果你不喜欢把功能根据事务特性分解成数个不同的MTS/COM+组件，那么也可以选择把相关的功能都放在一个 MTS/COM+组件中，并且把MTS/COM+组件的事务特性设定为“支持事务”以避免激活不必要的事务管理。但是，对于 MTS/COM+组件中需要修改数据的方法，则可以直接使用前面章节说明的ITransactionContext 或 ITransactionContextEx 接口，由程序代码直接控制事务管理。这样不但可以使用一个单一的 MTS/COM+组件，并且能够在方法层级控制事务管理，也是不错的方法。只是程序员必须要小心，由于在方法中直接控制事务管理。这样不但可以使用一个单一的 MTS/COM+组件，并且能够在方法层级控制事务管理，也是不错的方法。只是程序员必须要小心，由于在方法中直接控制了事务管理，因此当这个 MTS/COM+组件和其他的MTS/COM+组件在一起使用时可能会发生事务冲突的情形。因此，如果程序员自行使用ITransactionContext 或 ITransactionContextEx 接口控制事务管理时，最好在使用这两个接口激活事务管理之前先调用 ObjectContext 的 IsInTransaction 方法来判断客户端是否已经激活了事务管理，再来决定是否真的需要再使用 ITransactionContext 或 ITransactionContextEx 接口来激活事务管理。

9-6、结论

本章说明了程序员如何以 Delphi 的角度来开发 MTS/COM+应用系统，并且提供了一些建议的架构让程序员用来开发 MTS/COM+应用系统。当然程序员可以自己决定是否要使用本章建议的架构来开发应用系统，或根据本章的想法来开发出更好的架构。但是本章中建议使用这种架构来开发 MTS/COM+应用系统是因为它能够解决把所有组件和应用逻辑程序代码放在一个单一的 MTS/COM+数据模块中可能会产生的问题。同时这种架构比较有效率。当然天下没有免费的午餐，使用这种架构在开发时会比直接使用单一的 MTS/COM+数据模块有一点不方便，但是我们仍然可以使用一些技术来克服这些问题。

不管你决定使用哪一种架构来开发 MTS/COM+应用系统，在 Delphi 中程序员如果要使用 TDCOMConnection 组件以及数据存取的话，那么程序员必须要修改一些 Delphi 的执行行为，才不会造成 MTS/COM+应用系统执行的异常，特别是在和事务管理方面有关的问题。修改某些 Delphi 的执行行为的另外一个好处便是程序员可以使用相同的开发模式和组件来开发各种不同的应用系统，而不需要在不同的应用系统中使用不同的组件和程序代码来实现。

在阅读完本章之后，你应该已经了解了 MTS/COM+应用系统的概念以及如何使用Delphi 的组件开发 MTS/COM+应用系统。下一章要讨论的便是非常重要的主题，即如何开发有效率的 MTS/COM+应用系统。经过适当调整并且遵照一些基本的程序代码技巧可以让MTS/COM+应用系统执行得非常有效率。即使是在客户端不多的情形下 MTS/COM+应用系统仍然有可能提供和主从架构相近的执行效率。当然，在客户端众多，或开发 Internet/Intranet或电子商务系统时，使用 MTS/COM+架构的系统其效益和延展性就不是主从架构所能够比拟的了。

PDF created with FinePrint pdfFactory trial version http://www.fineprint.com