撰写高效率的 MTS/COM+组件和 MTS/COM+应用系统

许多人在开发了一些基本的 MTS/COM+分布式多层应用系统之后都觉得执行效率不好，因此也开始怀疑分布式多层应用系统是否能够提供合理的执行效率。事实上，MTS/COM+提供了许多主从架构无法提供的功能，这些功能能够帮助分布式多层应用系统增加执行效率以及系统的延展性。那么为什么许多人都会觉得分布式多层应用系统的执行速度很缓慢呢？

这个问题正是本章要讨论的主题。使用正确方法开发的分布式多层应用系统其执行效率可以和主从架构并驾齐驱，并且提供了更好的系统延展性。开发高效率的 MTS/COM+组件和应用系统并不困难，但是程序员必须掌握一些重要的概念和技巧。本章讨论的内容将让 Delphi 的程序员完全掌握这些开发 MTS/COM+应用系统最重要的技术。

本章重点

- ► 你应该牢记的事情
- ► 快速建立和呼叫远程 MTS/COM+对象
- ► MTS/COM+对象和数据的传递
- ► 状态信息
- ► 结论

现在你应该知道如何使用 Delphi 开发 MTS/COM+组件以及 MTS/COM+应用系统了。通过 Delphi 提供的向导，程序员可以快速地开发出以 MTS/COM+为骨架的分布式多层应用系统。但是我们使用分布式多层应用系统的目的除了因为这种架构比较好维护并且适合使用在 Internet/Intranet 或电子商务系统之外，我相信执行效率仍然是所有程序员都非常重视的。不管一个系统的架构有多么的好，没有适当的执行效率，这种系统仍然不会为用户所接受。

许多对于 MTS 应用系统有经验的程序员都觉得 MTS 的执行效率不好，我也有一些朋友在使用 MTS 开发分布式多层应用系统之后失败的例子。当然失败的例子大都不是因为 MTS 系统无法稳定的执行，事实上，MTS 是相当稳定的执行环境，Windows2000 的 COM+更比 MTS 来得稳定和有效率。这些系统真正失败的原因是因为当客户端的人数一多的时候，整个系统执行的速度就如“龟速”一般。我个人也曾接到一些读者的来信，其中有一些读者在信中说当他们决定把主从架构改为使用 MTS 的分布式多层应用系统之后，不但开发的速度变慢了，因为他们得同时开发执行在中间层的 MTS 组件以及客户端的图形用户接口应用程序。而且当系统开发完成之后，他们发现 MTS 的分布式多层系统执行效率比主从架构缓慢了数倍以上。这让他们相当地沮丧，因为他们觉得从主从架构改为分布式多层之后完全感觉不到任何的好处，反而是系统的用户不断地向他们抱怨新的系统比旧的系统更为差劲，因此这些读者想要知道如何能够解决这些问题。

当然，我无法立刻帮助他们解决问题，因为系统不是我写的，并且一个系统执行效率的问题除了和 MTS 有关之外，基本上开发人员如何设计系统是影响整个应用系统执行效率最大的因素。同样，本章讨论的内容并不在于教导程序员如何以对象导向分析/对象导向设计的方式设计 MTS/COM+的分布式多层应用系统。有关这个问题的说明，你应该参考相关的对象导向分析/对象导向设计书籍。不过本章将会从技术角度告诉你如何使用最有效率的方式来建立和调用 MTS/COM+对象，让你开发的 MTS/COM+分布式多层应用系统能够提供接近主从架构的执行效率。

10-1 你应该牢记的事情

就和几年前许多程序员从 File-Based 应用系统转换到主从架构一样，由于对于 File-Based 数据库和 RDBMS 的异同不甚了解，许多程序员仍然以 File-Based 数据库的方式来开发主从架构，因此造成了所开发的主从架构系统执行缓慢的问题。经过了一段时间的学习之后这些程序员才了解了问题的所在，慢慢地改善了开发主从架构的技术，才让主从架构系统的开发进入成熟的阶段。现在我们面对的是以 MTS/COM+为中介软件的应用系统，它和以 Delphi 的 MIDAS 应用程序服务器作为中介软件的应用系统也有所不同，即使这两种架构都是分布式多层应用系统。因此程序员也必须学习使用新的概念和技术来开发 MTS/COM+应用系统。如果你对开发 MIDAS 应用程序服务器已经有些经验，并且使用你在 MIDAS 应用程序服务器学习到的方式来开发 MTS/COM+应用系统，那么你仍然可能会觉得非常的沮丧，因为你会发现你所开发出来的 MTS/COM+应用系统的执行效率比 MIDAS 应用程序服务器缓慢了 5~10 倍的速度。当然其中一部分原因是因为 MIDAS 应用程序服务器只提供了最简单的远程数据存取的功能，而 MTS/COM+应用系统却提供了相当多的系统功能，这些是 MIDAS 应用程序服务器无法比拟的。但是，即使想以一些执行效率来交换系统功能和稳定性，我们也不希望结果是无法接受的缓慢速度。我想我们要的分布式多层应用系统是具有丰富的功能以及稳定的执行环境，但是它的执行效率也必须在可接受的范围之内。为了达成这种对于 MTS/COM+应用系统的要求，程序员在开发 MTS/COM+应用系统必须把几件事情牢记在心中。下面是一些基本的事情，程序员在开发 MTS/COM+应用系统时必须把它们牢记在心中：

- ► 对于客户端而言，应该尽早取得需要使用的 MTS/COM+对象，并且在最后使用完毕之后再释放取得的 MTS/COM+对象。

- ► 尽量避免激活不必要的事务 Context。
- ► 对于 MTS/COM+对象而言，应该让事务管理越晚发生越好。并且在执行完必要的工作之后立刻调用 SetComplete/SetAbort 和 EnableCommit/DisableCommit 方法释放占据的资源。
- ► 尽量把相关的 MTS/COM+对象放在同一个套件组件中，尽量减少不同套件组件之间 MTS/COM+对象的调用。如果一定要进行不同套件组件之间的调用，请再参考第一项的说明。
- ► 在 MTS 中使用 STA 线程模型的对象，在 COM+中使用 Neutral/Rental 线程模型的对象。
- ► 尽量利用 MTS/COM+对于数据库的 Pooling，请记得使用 MTS/COM+的对象 Pooling 来利用数据库 Pooling，而不要使用主从架构的方式来利用数据库 Pooling。

当然，除了这些规则之外，程序员也必须在适当的场合以自己的经验来判断如何以最好的方式来撰写程序代码。下面的章节会以上述的规则来实际地验证这些法则是否真的有效。

10-2、快速建立和调用远程 MTS/COM+对象

在开发 MTS/COM+应用系统时的许多时间是花费在建立和调用远程 MTS/COM+对象。你可别小看这些时间，在许多的应用系统中这些花费的时间是相当可观的。如果你使用 MTS/COM+来开发项目，那么这些花费的时间也可能是你的项目无法通过验收的主因。因此本小节要讨论的内容就是如何快速地建立和调用远程 MTS/COM+对象，让应用系统以最少的时间来完成这些工作。让我们冷静地想一想，在一个分布式多层应用系统中，应用程序所面对的是什么？那当然就是 MTS/COM+组件、企业逻辑程序代码以及数据(数据库)了。而且在分布式多层应用系统中经常需要调用 MTS/COM+组件来执行企业逻辑程序代码，并且处理数据。因此，如果我们能够使用最有效率的方式来建立和调用 MTS/COM+组件，那么我们就可以掌握系统 1/3 的执行效率因素了。

事实上，在 MTS/COM+应用系统中，程序员第一个要掌握的知识就是如何正确地使用 MTS/COM+组件。如果要详细说明的话，就是应用程序应该在什么时候建立 MTS/COM+组件？应用程序调用 MTS/COM+组件的负荷有多少？MTS/COM+组件调用 MTS/COM+组件的负荷又有多少？

在前面的小节中已经说过，应用程序应该尽早建立要使用的 MTS/COM+组件，这是因为当应用程序建立了 MTS/COM+组件之后，只是保留了 MTS/COM+组件的 ContextWrapper(CW)组件。CW 所占用的资源极少，因此对于系统的负荷不大。但是尽早建立 MTS/COM+组件，以及如何使用 MTS/COM+组件却对系统的执行效率有很大的影响。让我们以一个实际的范例来说明这些问题的重要性。这个范例会清楚地让我们了解建立 MTS/COM+组件时机的重要性，MTS/COM+组件和 MTS/COM+组件之间的调用负荷，以及主从架构和分布式多层应用系统之间的差距。

图 10-1 是一个范例 MTS/COM+客户端应用程序执行的结果画面。在这个范例中我使用了三种方法从一个数据表中取得数据。其中的“NoProxy”按钮使用的方法是每当这个按钮被点选之后，再建立并且调用 mtsCoorMonitor 这个 MTS/COM+组件，而 mtsCoorMonitor 会再建立另外一个称为 mtsdoWorkShops 的 MTS/COM+组件，并从 WorkShops 数据表中取得数据。在“NoProxy”按钮上方的 TEdit 控件显示了 mtsCoorMonitor 调用 mtsdoWorkShops 组件所花费的时间，而下方的 TEdit 控件显示了客户端应用程序调用 mtsCoorMonitor 花费的时间。“WithProxy”按钮则使用了另外一个方法来执行和“NoProxy”按钮相同的工作。只是“WithProxy”按钮并不是在被点选之后才建立 mtsCoorMonitor 组件，而是使用客户端应用程序在一开始执行之前便已经建立好 mtsCoorMonitor 组件。由于“WithProxy”按钮使用

了预先建立的 mtsCoorMonitor 组件，因此它符合了前面书说的应用程序应该尽早建立 MTS/COM+组件的规则。在“WithProxy”按钮上方的 TEdit 控件显示了 mtsCoorMonitor 调用 mtsdoWorkShops 组件所花费的时间，而下方的 TEdit 控件显示了客户端应用程序调用 mtsCoorMonitor 花费的时间。至于“CSConn”按钮则是使用 TADOConncetion 和 TADOQuery 组件直接连接 WorkShops 数据表并且取得数据。图 10-1 中窗体的 Caption 显示了客户端应用程序在一开始执行后先建立“WithProxy”按钮使用的 mtsCoorMonitor 对象所花费的时间。要注意的是，这个时间(0.55 秒)包含了 DLLHost.EXE 加载套件组件、客户端应用程序建立 DCOM 连接以及建立 mtsCoorMonitor 组件的时间。图 10-1 显示的结果是执行 100 次的平均时间。

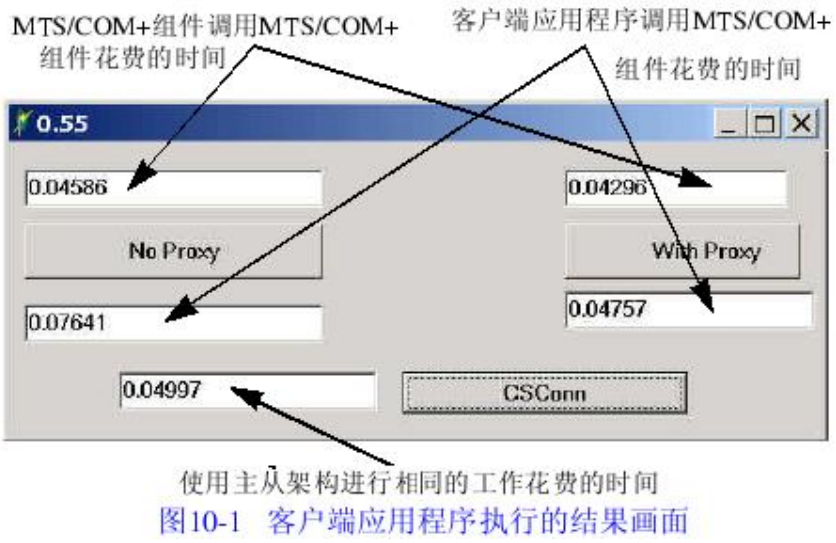

图10-1 客户端应用程序执行的结果画面

下面的表格是这三种方法的比较结果。

方 法	*执行速度*	*差 异*
不使用 Proxy	0.07641	N/A
使用 Proxy	0.04757	快 40%
主从架构	0.04997	比不使用 Proxy 快 35%，比使用 Proxy 慢 5%

上面的表格中我们可以看出尽早建立 MTS/COM+组件对于客户端应用程序的影响，它可以大大增加客户端应用程序的执行效率，甚至在少量数据的情形下(WorkShops 数据表包含的数据只有数十笔)，其执行的效率比主从架构还好。下面的程序代码就是“NoProxy”按钮的 OnClick 事件处理程序。可以看到它在循环中调用 TestPerf 函数，而 TestPerf 函数则是利用 wrapper 类别来建立 mtsCoorMonitor 组件，再调用它的方法来取得数据。

```
Procedure TForm2.btnNoProxyClick(Sender:TObject);
var
  iCount:Integer;
begin
  lStart:=GetTickCount;
  lCreateObjectTime:=0;
  for iCount:=1 to LOOPS   do   //Iterate
  begin
    lCreateObjectTime:=lCreateObjectTime+TestPerf;
  end;//for
  lEnd:=GetTickCount;
```

```
    edtNPTime.Text:=FloatToStr((lEnd-lStart)/1000.0);
    Edit1.Text:=FloatToStr(lCreateObjectTime/1000.0/LOOPS);
  end;

  function TForm2.TestPerf:Longint;
  var
    npMonitor:ImtsCoorMonitor;
  begin
    npMonitor:=ComtsCoorMonitor.CreateRemote(MTSMachine);
    npMonitor.GetWSInfo(lTime);
    Result:=lTime;
  end;
```

“WithProxy”按钮的 OnClick 事件处理程序如下：

```
  procedure TForm2.btnProxyClick(Sender:TObject);
  var
    iCount:Integer;
  begin
    lStart:=GetTickCount;
    lCreateObjectTime:=0;
    for iCount:=1 to LOOPS    do    //Iterate
    begin
      lCreateObjectTime:=lCreateObjectTime+TestPerf1;
    end;//for
    lEnd:=GetTickCount;
    edtWPTime.Text:=FloatToStr((lEnd-lStart)/1000.0);
    Edit2.Text:=FloatToStr(lCreateObjectTime/1000/LOOPS);
  end;

  function TForm2.TestPerf1:Longint;
  begin
    wpMonitor.GetWSInfo(lTime);
    Result:=lTime;
  end;

  procedure TForm2.FormActivate(Sender:TObject);
  begin
    lStart:=GetTickCount;
    wpMonitor:=ComtsCoorMonitor.CreateRemote(MTSMachine);
    lEnd:=GetTickCount;
    Self.Caption:=FloatToStr((lEnd-lStart)/1000.0);
  end;
```

在循环中它调用 TestPerf1 函数，再由 TestPerf1 调用 mtsCoorMonitor 组件的方法以取得数据。至于 TestPerf1 函数使用的 mtsCoorMonitor 组件则是在这个客户端应用程序的 FormActivate 事件处理程序中预先建立的，并且储存在类的全局变量之中，以便在 TestPerf1 函数中使用。

下面的程序代码则是 mtsCoorMonitor 组件的 GetWSInfo 方法的程序代码。可以看到在这个方法中它使用了 ObjectContext 的 CreateInstance 方法以建立其他的 MTS/COM+组件，再调用这个组件的方法来取得 WorkShops 数据表的数据。

```
Function TmtsCoorMonitor.GetWSInfo(outlPerData:Integer):OleVariant;
var
  IdoWorkShops:ImtsdoWorkShops;
begin
  try
    lPerData:=GetTickCount;
    ObjectContext.CreateInstance(CLASS_mtsdoWorkShops,IID_ImtsdoWorkShops,
                                 IdoWorkShops);
    sSQLData:='Select * from WorkShops';
    Result:=GetWorkShopsData(sSQLData);
    ObjectContext.SetComplete;
    lPerData:=GetTickCount-lPerData;
  except
    on Exception do
      ObjectContext.SetAbort;
  end;//try/except
end;
```

由于上面的范例执行了 100 次并且取用执行的平均值，因此即使是在“NoProxy”的情形下都有资源被高速缓存(cache)，所以还看不出尽早建立 MTS/COM+组件和需要时才建立 MTS/COM+组件的真正差异。图 10-2 便是让这个范例应用程序只执行一次的结果。下面的表格是这三种方法的比较：

方 法	*执行速度*
不使用 Proxy	0.21N/A
使用 Proxy	0.051　快 75%以上
主从架构	0.08 比不使用 Proxy 快 62%，比使用 Proxy 慢 37%

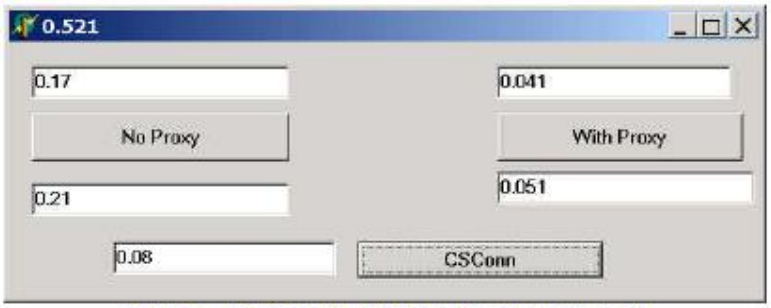

图10-2　客户端应用程序执行的结果画面

从这次的比较中我们可以真正地看到这两种不同方式执行效率的差距，使用 Proxy 比不使用 Proxy 足足快了 75%以上。事实上，这是比较接近实际的分布式多层应用系统。因为在实际的分布式多层应用系统中，虽然客户端应用程序会调用 MTS/COM+对象，但是在每一个调用之间也会执行其他的企业逻辑程序代码，或等待用户输入数据。因此 MTS/COM+可能已经暂时释放了 MTS/COM+组件。在这种情形下，尽早建立 MTS/COM+对象，并且保存 MTS/COM+对象的 Proxy(或 CW)会比每一次重新建立 MTS/COM+对象再调用快 75%以上，甚至可以快上数倍。因为在实际的应用中客户端应用程序可能会传递许多参数，这对于

客户端应用程序的执行效率会有重大的影响。使用 Proxy 方法还有另外一个好处，那就是可以避免 DLLHost.EXE 释放套件组件并且结束执行。这样可以让套件组件停留在内存中，以加快 MTS/COM+应用系统的执行效率。因为如果有其他的客户端应用程序也需要调用这个软件包中的 MTS/COM+组件，就不需要再花费激活 DLLHost.EXE 并且加载软件包的时间了。例如图 10-3 是执行另外一个客户端应用程序的画面。从图 10-3 的窗体 Caption 中我们可以清楚地看到，它一开始建立 mtsCoorMonitor 花费的时间大幅地减少了 30％以上。下面的表格是这两种方法的比较：

方 法	*执行速度*	
第 1 次激活 MTS/COM+	0.521	N/A
第 2 次之后激活 MTS/COM+	0.2	快 30％以上

因此使用 Proxy 的方式不但可以增加这个应用程序的执行效率，还能够增加其他客户端应用程序的效率，进而增加整个分布式多层应用系统的效率。在下面的小节中我们从几个方面来说明如何在 MTS/COM+应用系统中增加系统的执行效率。

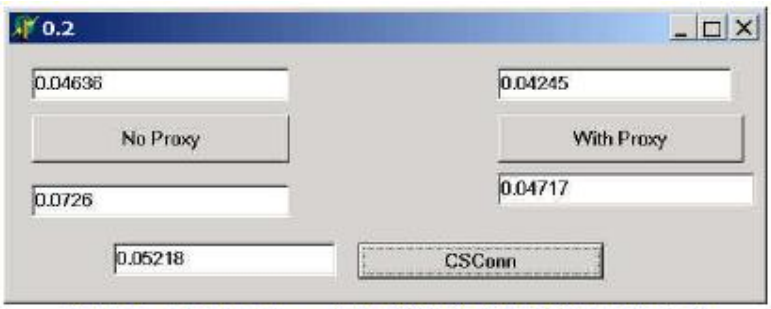

图10-3 执行另外一个客户端应用程序的结果画面

1、客户端和远程 MTS/COM+对象

程序员应该尽早在客户端应用程序中建立要使用的 MTS/COM+组件，并且储存 MTS/COM+组件的 proxy 在类别变量中，以便在不同的函数中使用。这样比在不同的函数中再建立 MTS/COM+组件可以大大地改善执行效率。如果程序员再配合使用协调对象的方式，不要建立大量的 MTS/COM+组件，那么又可以再增加执行效率，并且节省系统使用的资源(因为这可以减少 MTS/COM+执行环境需要维护的 CW)。

2、同一套件组件之间的 MTS/COM+对象

从上面的范例中我们可以知道，在 MTS/COM+执行环境中，MTS/COM+组件建立其他的 MTS/COM+组件的负荷是非常小的，因此程序员不需要担心 MTS/COM+组件建立其他 MTS/COM+组件需要花费的时间，而把所有的企业逻辑程序代码撰写在一个单一的 MTS/COM+组件之中。而且请你仔细观察前面 mtsCoorMonitor 的 GetWSInfo 方法的程序代码。在 GetWSInfo 方法中它用来建立 mtsdoWorkShops 组件的方式是在每一次被调用时才建立 mtsdoWorkShops 组件，而不是预先建立 mtsdoWorkShops。但是这样所花费的时间仍然很小。这是因为这些组件都是在一个相同的套件组件之中，因此彼此之间的建立组件花费的时间也就很小。如果我们再把上面的程序代码改成预先建立 mtsdoWorkShops 组件，那么又可以增加一些效率。

3、不同套件组件之间的 MTS/COM+对象

当然，如果客户端应用程序调用的 MTS/COM+组件分属于不同的套件组件，例如，如果 mtsCoorMonitor 组件和 mtsdoWorkShops 组件是在不同的套件组件之中，那么 mtsCoorMonitor 组件调用 mtsdoWorkShops 组件需要花费的时间就很可观了。甚至可能比前面客户端应用程序使用“NoProxy”的方法建立 MTS/COM+组件还多。因此，当程序员在开发 MTS/COM+应用系统时，一定要好好地规划 MTS/COM+组件之间的关系，应该尽量把相

关的 MTS/COM+组件放在相同的套件组件之中，并且尽量避免跨套件组件之间的组件建立和组件调用。虽然在实际的 MTS/COM+应用系统中跨套件组件或跨机器之间的组件建立和调用是不可避免的，例如为了提供容错能力以及平均负荷的能力，但是只要程序员有良好的规划，仍然可以有效地降低系统的负荷。

从上面的实际范例中我们学习到了什么呢？非常重要的，那就是程序员应该尽量先建立应用程序需要使用的 MTS/COM+对象，而且应该把建立的 MTS/COM+对象的接口变量储存在类别的全域变量之中，而不要在每一个独立的程序或函数中再建立 MTS/COM+对象。这样可以大大加快应用程序调用远程 MTS/COM+对象的速度。因为当应用程序以全域变量储存建立的 MTS/COM+对象时，事实上，客户端应用程序是保留了 MTS 的 ContextWrapper 对象，或 COM+的 Context-awareProxy。如此做可以减少连接远程 MTS/COM+执行环境的次数以及减少 MTS/COM+执行环境需要初始化以及建立 ContextWrapper 对象或 COM+的 Context-awareProxy 的时间，因此应用程序会执行得比较快速。至于客户端应用程序保留 ContextWrapper 对象或 Context-awareProxy 在 MTS/COM+执行环境中花费的内存空间大约是 600K 到 800K 不等，可以说是相当的少。使用这么少的空间却得到良好的执行效率是值得的。如果程序员又使用了协调对象的模型，减少客户端应用程序需要建立大量 MTS/COM+组件的话，那么就可以同时节省内存，又能够得到执行效率，可以说是两全其美。另外，在相同的套件组件中对于 MTS/COM+对象的建立以及相互之间的调用时间都非常的快速，因此程序员更应该使用协调对象模型，让协调对象在 MTS/COM+执行环境中调用其他的 MTS/COM+组件来完成工作，而不要让客户端应用程序调用大量的 MTS/COM+组件来执行工作。最后，程序员在设计 MTS/COM+组件和套件组件时，必须好好地规划哪些 MTS/COM+组件应该实现在哪一个套件组件之中。因为跨越套件组件之间的调用速度是非常缓慢的，因此规划良好的 MTS/COM+组件以及套件组件能够增加 MTS/COM+应用系统的执行效率。

10-3、MTS/COM+对象和数据的传递

在开发 MTS/COM+应用系统时，我相信大部分的系统都会和数据有关系。事实上，使用 MTS/COM+开发分布式多层应用系统的部分原因也是因为这种架构能够处理更多的客户端要求，整合 Web 应用，以及使用更有效率的方式来处理数据，以便增加系统的延展性。但是许多人在尝试开发了一些分布式多层应用系统之后，都会觉得分布式多层应用系统比起主从架构缓慢了许多，因此许多人也开始怀疑为什么要使用分布式多层应用系统来取代主从架构。当然，如果你只是需要开发内部的信息系统，人数在 200 人以内，不需要 Internet 应用，不需要 Intranet 应用，也不需要电子商务或是任何形式的 Web 应用，那么主从架构的确是一个不错的选择，因为经过适当调整的主从架构的确非常有效率。主从架构到底已经是一个技术成熟的系统架构。除了系统的维护成本较高之外，似乎没有什么重大的缺点。不过分布式多层应用系统毕竟是现在的趋势，也提供了许多主从架构无法提供的能力，因此我们现在要解决的问题应该是如何在分布式多层架构中尽量提供和主从架构类似的效率。这并不是一个简单的问题，

在前面的小节中我们说明了如何使用最有效率的方式调用 MTS/COM+组件，本小节将以如何使用有效率的方式来处理数据作为主题，让你了解通过适当的调整和遵循一些原则，分布式多层应用系统也能够提供接近主从架构的执行效率。

10-3-1、分布式多层架构和主从架构

为什么主从架构会比分布式多层架构来得有效率？你可曾自己想过这个问题？记得数年前当分布式多层应用系统刚开始出现时，我曾在中科院做了一次演讲，介绍分布式多层应用系统、中介软件、COM/DCOM 以及 CORBA 的概念和技术。当时就有一位先生问我，既然分布式多层应用系统多了一个应用程序服务器(或中介软件)，那么这样的架构就一定比主

从架构缓慢，因为在主从架构中客户端应用程序是直接和数据库服务器连接的，客户端应用程序需要的数据也就直接由数据库传递到客户端，不再需要经过应用程序服务器。由于少了应用程序服务器这一层，所以主从架构会比较快速，不是吗？这个问题的答案牵涉了许多因素。简单地说，如果就实际的架构来说，由于数据需要从数据库服务器先传递到应用程序服务器，再由应用程序服务器传递到客户端应用程序，因此会比主从架构直接从数据库服务器传递到客户端应用程序来得缓慢。而且分布式多层使用了 COM/DCOMMarshalling 的方式传递数据，负荷也比直接使用 TCP/IP 等通信协议来得大。不过在主从架构中客户端应用程序必须持续地和数据库服务器保持联机的状态，因此对于网络的使用量又比分布式多层系统大，加上分布式多层系统可以通过中介软件享有各种 Pooling 的好处，因此又可以拉近和主从架构的差距。所以我说如何设计和开发应用系统的架构才是影响分布式多层应用系统执行效率的最大因素，经过适当设计和调整的分布式多层应用系统是可以提供类似主从架构效率的系统的。

在分布式多层应用系统中，程序员应该使用类似 HTTP 通信协议的概念来存取数据。这也就是说，客户端应用程序应该尽量只选择需要的数据，而不要使用类似 Select * from TableName 的 SQL 命令来存取数据，以避免从应用程序服务器一次取得大量的数据到客户端。使用少量、多次的方式存取数据可以大幅地减少系统的响应时间，而让用户对你的分布式多层应用系统更为满意。

10-3-2、传递数据的数量

在分布式多层应用系统中程序员应该牢记的是千万不要传递大量的数据，特别是客户端应用程序不需要的数据。程序员应该戒除使用 Select * From TableName 这种 SQL 命令来存取数据的习惯。在分布式多层应用系统中，如果程序员在客户端应用程序中尽量只存取目前需要的数据，并且以少量、多次的方式存取数据，那么 MTS/COM+将能够使用非常有效率的方式提供客户端需要的数据。基本上，客户端应用程序一次存取的数据如果少于一定的数量，那么分布式多层应用系统存取数据的速度几乎是和主从架构一样有效率。但是如果大于这个数量，那么分布式多层应用系统的执行效率将会大大地低于主从架构，相差的效率可达 10 倍以上。这是相当重要的，因为超过这个临界点的话，那么分布式多层应用系统的执行效率和主从架构比起来实在是相差太远。但是如果在临界点之内，那么分布式多层应用系统的执行效率却非常良好。那么这个临界点是在哪里呢？当然，这就和你的系统建制架构有关了。但是就我的经验来说，900 笔之内的数据似乎是最好的范围。超过 1000 笔数据的话，那么执行效率就非常不理想了。一次存取 900 笔的数据在大部分的应用中都已经足够了，我想没有什么用户能够在一秒之内浏览/查阅完 900 笔的数据。因此，即使我们需要处理的数据可能有 2000 笔，那么分布式多层应用程序仍然可以使用 3~4 次的存取，让用户处理完毕。这样不但有效率，而且也能够节省资源，因为用户可能根本不需要处理最后的数据，因此可能在存取两次数据之后就结束应用程序的执行。因此程序员在分布式多层应用系统中应该掌握的技术便是如何能够以少量、多次的技术来满足客户端的要求。这需要结合 MTS/COM+组件以及状态信息的技术。

> 一个实例：我的一个朋友曾经询问我一个问题，那就是他在 MTS 应用系统中撰写一个报表程序。这个报表程序需要处理大量的数据(数万笔的数据)，因此它使用 QuickReport，调用中间的 MTS 组件取得数据，再传递到客户端的 QuickReport 中处理并且打印数据。但是在撰写完之后，他说这个 MTS 应用程序需要执行 3~4 个小时，他问我有没有什么方法可以增加这个报表程序的执行效率，因为用户实在无法忍受这种执行速度。虽然他一直问我有没有什么多线程的技术可以增加这个程序的执行效率，但是我说基本上这已经是一个错误的设计，再使用多线程的技术只不过能够增加些许的效率，还是无法让用户接受的。这个程序的问题在于程序员根本不应该这样设计。

对于这种需要传递大量数据到客户端的应用程序即使是在主从架构中也非常吃力，何况是对于早已超出数据存取临界点的分布式多层应用系统只要想一想如果在网络中同时有数个人在执行这个程序的话，那么会有多少的数据在网络中来回地传递呢？因此我的答案很简单，使用存储过程来处理报表最后需要的结果数据，这样可以避免在网络中传递大量的数据。当存储过程处理完毕之后，再把处理结果的数据传递到客户端的 QuickReport 中打印。因此客户端调用的 MTS 组件应该先调用这个存储过程，再传递结果数据回客户端。这样不但是多层的架构，也兼顾了执行效率。

那么这个程序最后的结果是什么呢？结果非常令人满意，执行的时间从原先的 3~4 个小时降低到 10 分钟之内，更重要的是系统使用的资源(中介软件和客户端的内存)大幅地减少，进而增加了整个系统的执行效率。其他的客户端应用程序再也不会因为有用户执行这个报表程序而明显地变慢了。

结论：程序应该根据程序的特性选择适当的架构，而不是整个系统都使用相同的方法来完成。

让我们看看一个实例，比较一下主从架构和分布式多层应用系统在存取数据时的表现。下面的图 10-4~图 10-7 分别是主从架构存取 10000 笔数据，而分布式多层应用系统存取 900、1000、3000 和 10000 笔数据的情形。从图中我们可以看到几个现象。第一是主从架构在存取数据时都有很稳定的存取时间，没有太大的变化。但是对于分布式多层架构来说，其间的变化就相当大了。分布式多层架构在 1000 笔之内还具有合理的存取速度。但是一旦超过 1000 笔数据，其执行速度就下降得非常迅速。在超过 3000 笔的情形下已经到了无法忍受的程度，因为光是存取数据就超过 3 秒，这在许多的项目中已经是不可接受的，更何况下图的数据只是执行一个客户端应用程序，如果同时有许多客户端用户同时执行应用程序的话，那么执行的时间一定需要更久。

图10-4 主从架构存取10000笔，多层架构存取900笔数据

图10-5　主从架构存取10000笔，多层架构存取1000笔数据

图10-6　主从架构存取10000笔，多层架构存取3000笔数据

图10-7　主从架构存取10000笔，多层架构存取10000笔数据

图 10-8 则是把主从架构和分布式多层架构存取数据笔数需要的时间同时显示在一个单一的图形中。从这个图形中我们也可以非常清楚地看到这两种架构的差异。因此许多人在开发了一些分布式多层应用系统的先导雏型系统之后，都觉得分布式多层应用系统在执行效率方面和主从架构比起来差得太多了。我想这大都是因为许多人仍然使用在主从架构或File-Based(FoxPro，dBase 等)系统的习惯来开发分布式多层应用系统，这样当然是不适当的。

因此程序员必须谨记在分布式多层应用系统中应该只存取客户端应用程序需要的数据，而不要使用 Select * From TableName 等 SQL 命令一次取得所有的数据，这样分布式多层仍然可以提供类似主从架构的执行效率。

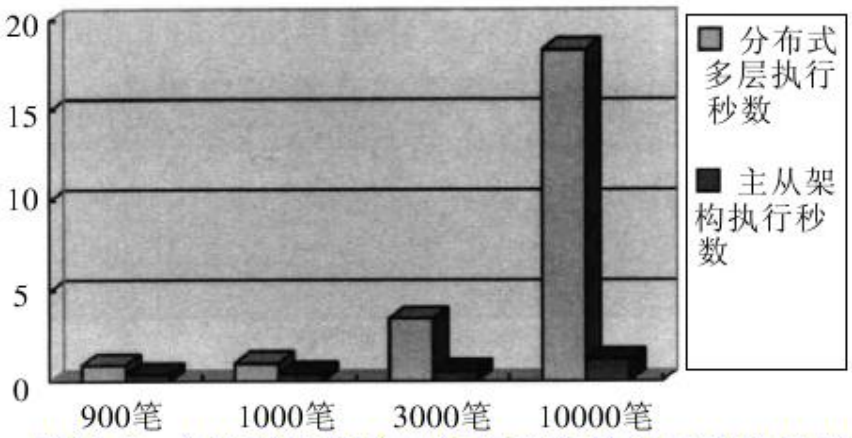

图10-8 主从架构和多层架构存取数据量的趋势图


你应该根据你的系统结构来做实验以便找出存取数据的临界点在哪里。

程序员在分布式多层应用系统中应该使用一次存取少量，并且在用户真的需要存取其他的数据时，再使用多次的技术来存取数据。事实上，这就是 MIDAS 的分段存取的概念，就像在 TClientDataSet 的 PacketRecords 特性中不要设定为-1 来一次取得所有的数据，而应该设定 PacketRecords 特性为 20 或其他适当的数值，一次只存取少量的数据。问题是如何在 MTS/COM+应用系统中使用分段的方式存取呢？这需要一点技巧，让我们使用一个非常有用的范例来展示这个技巧。

少量、多次存取的范例

MIDAS3.0 版本最重要的新功能之一便是允许程序员建立无状态对象，3.0 新的接口 IAppServer 就是为了帮助程序员开发无状态的 MIDAS 应用程序服务器。由于在 MTS/COM+中非常适合使用无状态对象，因此程序员如果能够结合 MIDAS 开发无状态的 MTS/COM+对象，那么不但可以让 MTS/COM+帮助应用系统 Pooling 数据库连接，还能够以分段的方式一次存取少量的数据，并且在用户真的需要其他数据时再存取随后的数据。从前面的观察中我们可以发现，如果 MTS/COM+应用系统一次存取的数据量不多，那么 MTS/COM+应用系统的执行效率会比主从架构还好。因此，在 MTS/COM+应用系统中使用 MIDAS 的无状态对象不但可以节省资源，利用 MTS/COM+的 Pooling 功能，还能够增加系统的执行效率。在本系列的第一本书《Delphi5.x 分布式多层应用系统篇》中已经说过，要建立无状态对象的 MIDAS 组件，程序员可以设定 TClientDataSet 组件的 PacketRecords 特性值为一个特定的数值。这个数值就代表 TClientDataSet 组件一次从应用程序服务器取得的数据笔数。设定 PacketRecords 特性值再结合 TClientDataSet 组件的 BeforeGetRecords 事件处理程序，以及 TDataSetProvider 的 BeforeGetRecords 事件处理程序，程序员既可以开发 MIDAS 无状态对象，又能够享有分段存取数据的好处，是一个兼顾资源和效率的鱼与熊掌兼得之法。不过，如果按照《Delphi5.x 分布式多层应用系统篇》在 MTS/COM+中开发 MIDAS 无状态对象，那么可能会碰到困难，因为在 MIDAS 应用程序服务器和在 MTS/COM+中有一些行为是不太一样的。为了能够在 MTS/COM+中开发 MIDAS 无状态对象以便以少量、多次的方式存

取数据，所以我修改了在《Delphi5.x 分布式多层应用系统篇》一书中的范例，让它能够正确地在 MTS/COM+应用系统中执行。首先，让我们建立一个 ActiveX Library 项目，然后在此项目中建立一个 MTS 数据模块，并且在这个数据模块中放入 TADOConnection、TADOQuery 和 TdataSet Provider 组件。在这个范例中我把 ADO 的数据存取组件连接到 MSSQLServer7 的 Pubs 数据库中的 employee 数据表。图 10-9 是 MTS 数据模块的画面：

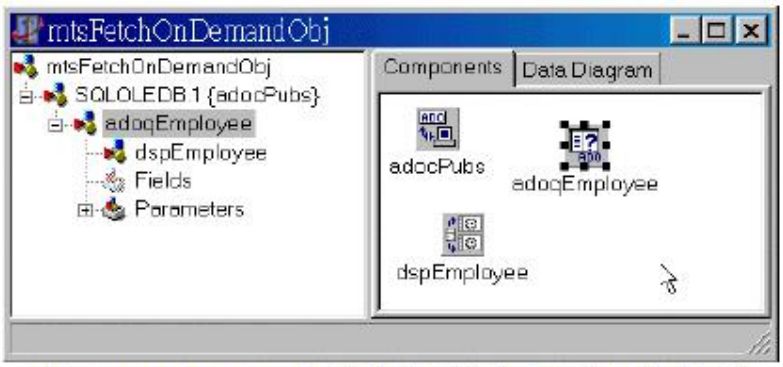

图10-9 MTS/COM+组件的数据模块以及数据存取组件

接着再建立一个客户端应用程序项目。在客户端应用程序中使用 TDCOMConnection、TClientDataSet 连接到刚才开发的 MTS/COM+组件，并且设定 TClientDataSet 组件的 PacketRecords 特性值为 5，以便一次从 MTS/COM+组件中取得 5 笔数据。图 10-10 是客户端应用程序的主画面，其中的“ ConnectMTS/COM+ 对 象 ” 按 钮 会 设 定 TDCOMConnection 为 True 以连接 MTS/COM+组件，并且设定 TClientDataSet 组件的 Active 为 True 以便从 MTS/COM+组件取得第一次的 5 笔数据。至于“More”按钮则可以让程序员依序取得随后的数据。由于在客户端应用程序中设定了 TClientDataSet 的 PacketRecords 特性值为 5，而且中间是使用 MTS/COM+组件，因此当第一次客户端应用程序取得前 5 笔数据之后，中间的 MTS/COM+组件便会被释放。因此，当用户点选了客户端应用程序的“More”按钮之后，客户端应用程序必须把上一次最后一笔的数据告诉中间的 MTS/COM+组件，再由 MTS/COM+组件从上一次最后存取的数据的下一笔数据处开始存取数据。

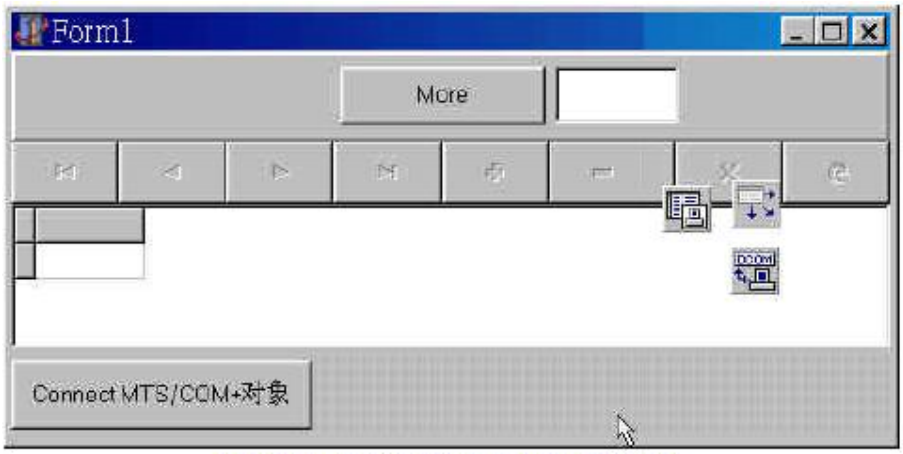

图10-10 客户端应用程序的主画面

要让客户端应用程序传递状态信息给 MTS/COM+组件，我们可以使用 MIDAS 的 BeforeGetRecord 事件处理程序来完成。在 BeforeGetRecord 事件处理程序中，这个范例客户端应用程序把上次存取数据的最后一笔数据的键值传递给 MTS/COM+组件，以便让 MTS/COM+组件能够从这笔数据之后存取新的分段数据。在下面的程序代码中，客户端应

用程序的 BeforeGetRecord 事件处理程序调用了 GetKeyFieldValue 函数取得这个键值，并且把键值指定给 OwnerData 以便通过 MIDAS 传递给 MTS/COM+组件。至于上图中“More”按钮的 OnClick 事件处理程序则非常简单，它调用了 TClientDataSet 组件的 GetNextPacket 方法以取得下一分段的数据。而 GetNextPacket 方法则会触发 BeforeGetRecord 事件处理程序把键值正确的传递给中间的 MTS/COM+组件。

```
Procedure TForm1.ClientDataSet1BeforeGetRecords(Sender:Tobject; var
                    OwnerData:OleVariant);
begin
  try
    vOwnerData:=GetKeyFieldValue(Sender);
    OwnerData:=vOwnerData;
  except
    on exception do
  end;
end;

function TForm1.GetKeyFieldValue(Sender:TObject):OleVariant;
var
  aCDS:TClientDataSet;
begin
  try
    try
      aCDS:=TClientDataSet.Create(Self);
      aCDS.Data:=ClientDataSet1.Data;
      aCDS.Last;
      Result:=aCDS.FieldByName('emp_id').Value;
    finally//wrapup
      aCDS.Free;
    end;//try/finally
  except
    one:Exception do
      raise;
  end;//try/except
end;

procedure TForm1.btnMoreClick(Sender:TObject);
begin
  try
    if (notbTrueEOF) then
    begin
      if (ClientDataSet1.GetNextPacket<ClientDataSet1.PacketRecords) then
          bTrueEOF:=True;
     end;
   except
```

```
        on Exception do;
      end;
      edtRecordCount.Text:=IntTostr(ClientDataSet1.RecordCount);
    end;
```

至于客户端的键值传递到中间的 MTS/COM+组件之后，MIDAS 会触发 TDataSetProvider 组件的 BeforeGetRecord 事件处理程序。在这个事件处理程序中，我们只需要把 TADOQuery 的 cursor 定位到此键值的下一笔数据处，MIDAS 即可回传从定位处开始的下一分段的数据。下面即是 TDataSetProvider 组件的 BeforeGetRecord 事件处理程序的代码：

```
    procedure TmtsFetchOnDemandObj.dspEmployeeBeforeGetRecords(Sender:
                                                TObject;varOwnerData:OleVariant);
    var
      bResult:Boolean;
    begin
      if not ((VarIsEmpty(OwnerData)) or (VarIsNull(OwnerData))) then
        with SenderasTDataSetProvider do
        begin
          DataSet.Open;
          bResult:=DataSet.Locate('emp_id',OwnerData,[]);
          DataSet.Next;
        end;//with
    end;
```

现在我们如果编译这个范例 MTS/COM+应用系统，就会看到如图 10-11 和图 10-12 所示的执行画面。从下面的画面中可以看到，客户端应用程序果然可以在使用 MTS/COM+组件的情形下以分段的方式(一次存取 5 笔数据)从 Employee 数据表取得数据。这个范例应用系统可以很正确地执行。但是对于一些大型的数据表，例如包含数万笔或数十万笔的数据而言，在中间的 MTS/COM+组件如果使用上述的程序代码来定位键值数据的话，那么由于上述的程序代码是使用 Locate 来搜寻键值数据，因此 MTS/COM+组件仍然会从数据库服务器传递大量的数据到中间的 MTS/COM+组件。这可能会造成系统沉重的负担，而且这样做也传递了大量没有用的数据非常浪费资源。因此我们最好只让数据库服务器直接传递我们需要的分段数据，而不需要传递其他不相关的数据。

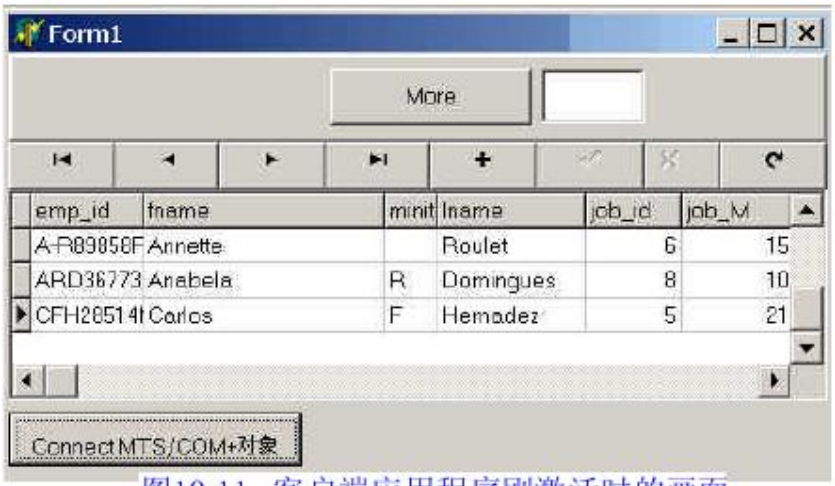

图10-11　客户端应用程序刚激活时的画面

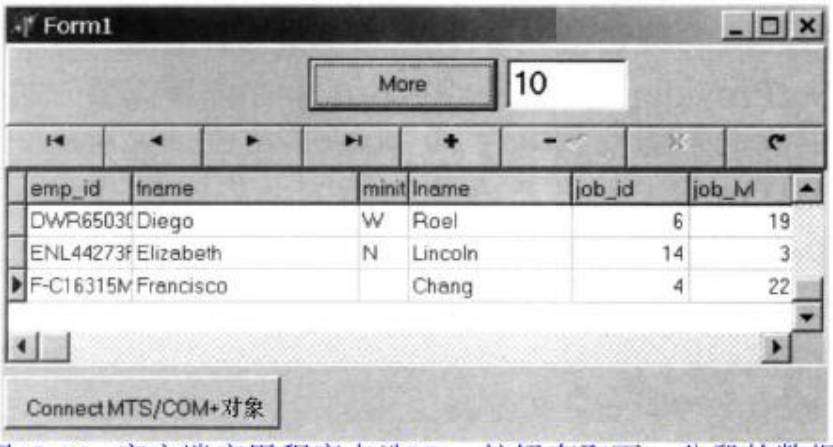

图10-12　客户端应用程序点选More按钮存取下一分段的数据

为了增加系统的执行效率，我们可以使用数个方法来解决。例如直接撰写存储过程。在这个范例中由于我是使用 MSSQLServer7.0，因此我把 MTS/COM+组件中的 TADOQuery 组件使用 SQL 命令：

```
select * from employee
```

改成如下的 SQL 命令：

```
select top 5 * from employee where emp_id>:idorderbyemp_id
```

这个 SQL 命令可以只传递符合条件的 5 笔数据给 MTS/COM+组件。其中的:id 这个动态 SQL 参数就是客户端应用程序传入的键值信息。修改了 SQL 命令之后，就让我们修改 TDataSetProvider 组件的 BeforeGetRecords 事件处理程序如下：

```
procedure TmtsFetchOnDemandObj.dspEmployeeBeforeGetRecords(Sender:
                                                    TObject;varOwnerData:OleVariant);
var
  bResult:Boolean;
begin
  if not ((VarIsEmpty(OwnerData)) or (VarIsNull(OwnerData))) then
  begin
    adoqEmployee.Parameters.ParamByName('id').Value:=OwnerData;
  end
  else
    adoqEmployee.Parameters.ParamByName('id').Value:='';
end;
```

在上面的程序代码中我们判断，如果客户端应用程序传递来键值信息，我们就把它指定给 id 这个动态参数。如果客户端应用程序没有传递任何的键值信息，那么这个情形就是客户端应用程序第一次开启 Employee 数据表时，我们就把 id 这个动态参数设定为空字符串(因为 Employee 数据表的键值字段 emp_id 是字符串类型)，以便从第一笔数据开始存取。使用了刚才修改的方法和程序代码之后，现在这个范例 MTS/COM+应用系统在处理大型数据表的情形下比刚才使用 Locate 的方法快了数倍到十几倍。当然，在大型数据表的情形下只使用 Locate 方法也比不使用少量、多次的无状态对象 MTS/COM+组件的方式快了许多。有关 MIDAS 的处理流程以及 BeforeGetRecords 事件处理程序等的说明，请参考《Delphi5.x 分布式多层应用系统篇》一书。

10-3-3、数据库连接 Pooling

MTS/COM+由于充分使用了各种 Pooling 的技术，因此在理论上能够在客户端人数渐多的应用中提供较好的执行效率和延展性。特别是当程序员开发 Web 解决方案或客户端应用程序以少量、多次的方式存取数据时，都能够表现得非常优秀。因此程序员应该使用适当的 ADO 驱动程序和数据库服务器，并且尽量充分利用 MTS/COM+提供的数据库 Pooling 机制。同样，让我们以一个实际的范例来观察数据库 Pooling 对于分布式多层应用系统的影响，并且和主从架构比较一下，看看数据库 Pooling 机制对于分布式多层应用系统的帮助。下面的一些观察结果是我使用 ADO For SQLServer 驱动程序，连接 MS SQL Server7.0 所得到的。这些结果也应该出现在 Oracle、Sybase 或 Informix 等的数据库服务器中，只要能够找到你使用的数据库的真正 ADO 驱动程序。这些观察的结果对于我们了解以及比较主从架构和分布式多层是非常重要的，因为这可以让我们了解为什么分布式多层应用系统比较适合使用在用户众多的系统，以及 Web 和电子商务等系统之中。

图 10-13 是我同时执行两个主从架构应用程序，并且使用 MS SQL Server 的 Profiler 观察的情形。其中的主从架构应用程序直接使用 TADOConnection、TADOQuery 组件连接数据库服务器，而 TADOQuery 组件则使用了 Select * fromCust 的 SQL 命令存取 Cust 数据表的数据。在下面的图形中显示了许多非常有意思的结果。首先，让我们注意的是这两个不同的主从架构应用程序在连接数据库服务器时都各自建立了独立的数据库连接。这可以从下图的 EventClass 字段中的 Connect 事件中看到。其次，由于这两个主从架构应用程序是一样的程序，因此 SQLServer 执行的命令都是一样的，并且这两个主从架构应用程序执行 SQL 命令所花费的时间也几乎是一样。这可以由下图的最后一个字段看到。这些观察结果代表对于主从架构应用程序来说，MSSQL 数据库服务器对于不同的主从架构应用程序可以重复利用的资源并不多。因此每一个不同的主从架构应用程序的连接数据库以及存取数据的时间几乎都一样，这也是在上一小节我们观察到主从架构应用程序执行的行为。

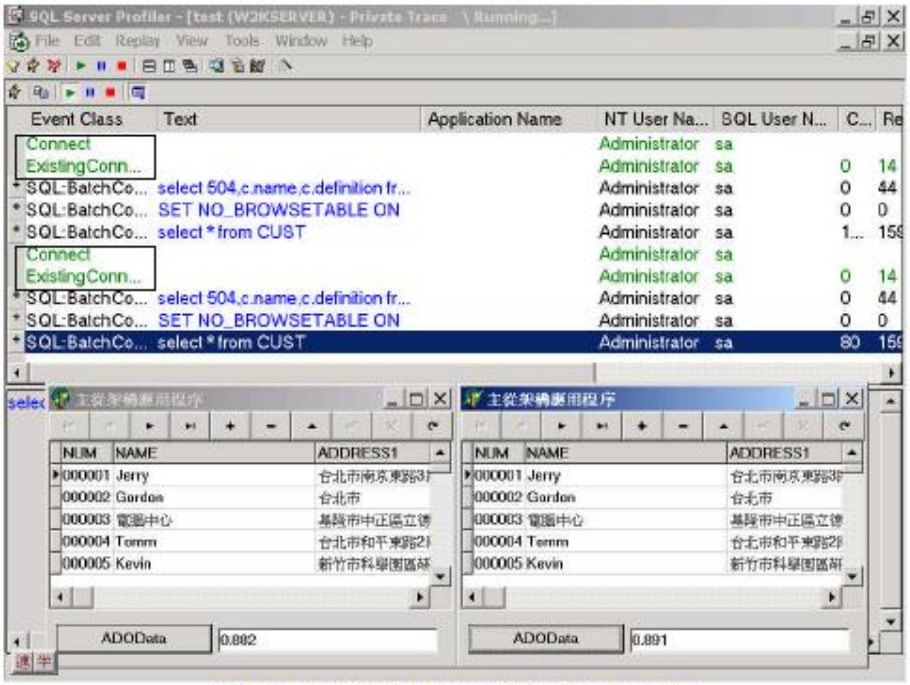

图10-13 同时执行两个主从架构应用程序存取数据

现在再让我们继续观察分布式多层应用程序。图 10-14 是在同样的状态下所得到的观察结果，图 10-14 和图 10-13 呈现了许多非常有意思的不同点。首先从图 10-14 中我们似乎看到

了这两个相同的分布式多层应用程序也似乎都建立了独立的数据库连接，所以这似乎和主从架构应用程序是一样的。不过这是因为在下图中我们只执行了两个分布式多层应用程序。稍后我们会执行多个分布式多层应用程序，到那时你便会了解这两者的不同。

另外要注意的是，对于分布式多层应用程序而言，MS SQL Server 选择执行的 SQL 命令是不一样的。MS SQL Server 会先准备 SQL 命令，然后以存储过程的方式来执行 TADOQuery 组件的 SQL 命令。由于 MS SQL Server7.0 可以高速缓存(cache)存储过程，因此第一个分布式多层应用程序需要的执行时间较多。而第 2 个分布式多层应用程序可以直接使用 cache 在内存中的存储过程，因此第 2 个分布式多层应用程序的执行时间大幅地减少，这可以从下图的最后一个字段看到。这个观察结果非常重要，因为这表示在分布式多层系统中，MTS/COM+和 MS SQL Server/ADO 驱动程序做了非常好的配合，可以尽量重复使用系统资源，所以可以加快分布式多层应用程序的执行速度。

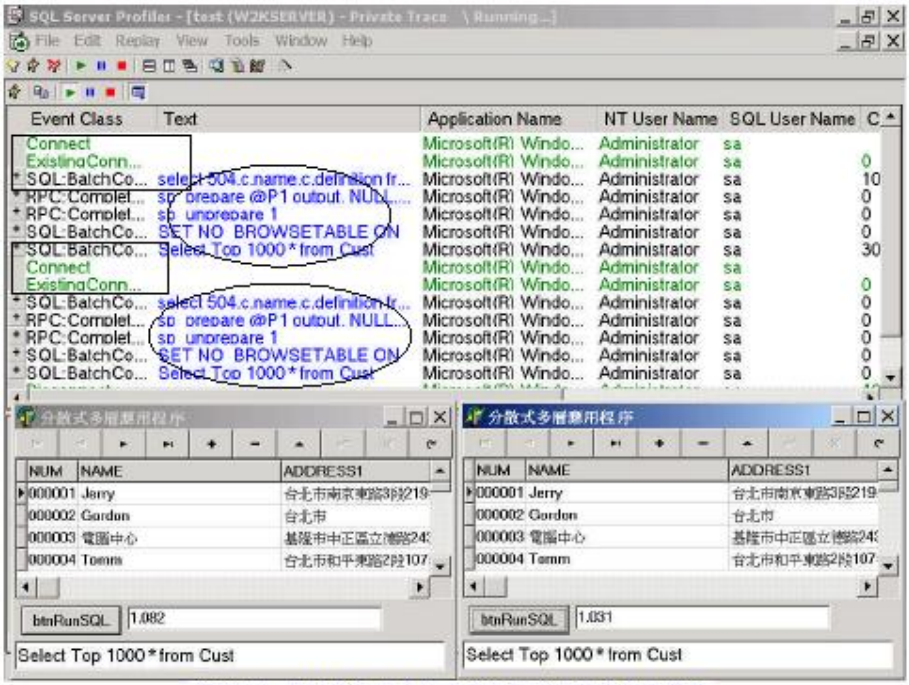

图10-14　同时执行两个分布式多层应用程序存取数据

现在再让我们观察程序对于数据库连接的使用状况。图 10-15 是执行 4 个主从架构应用程序的画面。我们可以看到这 4 个主从架构应用程序分别建立了和数据库独立的连接。至于图 10-16 则是执行 4 个分布式多层应用程序的画面，我们可以清楚地看到这 4 个分布式多层应用程序只建立了两个数据库连接。这代表 MTS/COM+有效地减少了需要的数据库连接，让这 4 个分布式多层应用程序能够重复地使用 MTS/COM+提供的数据库连接。这很明显地呈现了 MTS/COM+中数据库 Pooling 的功能。

分布式多层应用系统的确能够通过 MTS/COM+的数据库 Pooling 技术减少客户端实际需要的数据库连接数目。这样不但可以降低数据库服务器的负荷，减少网络的负荷，更能够让更多的客户端应用程序或 Web 同时存取数据库服务器。例如，如果你购买了一个 100 个用户的数据库连接，在主从架构中最多只有 100 个客户端用户能够使用。但是在分布式多层应用系统中，通过 MTS/COM+的数据库 Pooling 功能，系统却能够让多于 100 个的用户存取系统。这对于 Web 或电子商务系统来说是非常重要的，因为这样的系统具有较好的延展性。分布式多层应用系统通过数据库 Pooling 降低数据库服务器和网络的负荷而提供比主从架构更好的执行效率。因此当客户端用户人数逐渐增加以后，分布式多层的执行效率将会慢慢地

比主从架构更好。这一方面是分布式多层优于主从架构的地方。

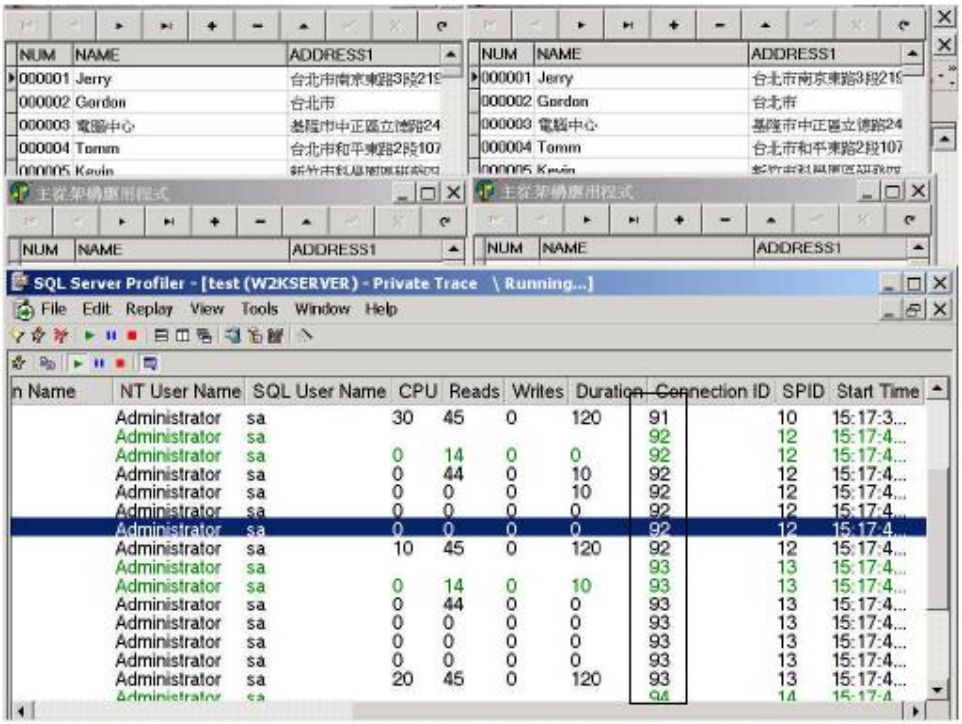

图10-15 同时执行四个主从架构应用程序存取数据，它使用了四个不同的数据库连接

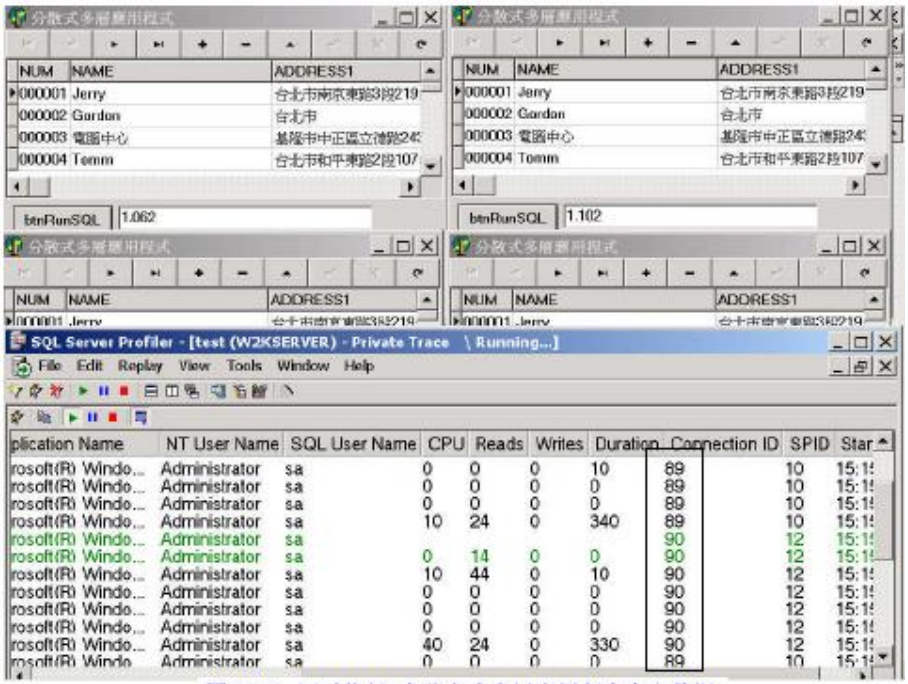

图10-16 同时执行4个分布式多层应用程序存取数据，它使用了两个不同的数据库连接

 还是那句话，你使用的 ADO 驱动程序的品质决定了系统的行为和特性。应该根据你使用的数据库服务器以及 ADO 驱动程序来观察分布式多层应用系统是否能够充分地使用 MTS/COM+的数据库 Pooling 功能。如果不能，建议你马上找一个品质较好的 ADO 驱动程序，或要求你的数据库服务器厂商提供 ADO 驱动程序。

10-3-4、数据库驱动程序的设定

最后一个有关数据存取效率的影响因素是许多人经常忽略的，那就是对于数据集组件特性值的设定，例如 TADOConnection、TADOQuery 等组件的特性值。分布式多层应用系统和主从架构使用的数据集组件的特性值应该是不一样的，程序员不应该把主从架构的特性值使用在分布式多层应用系统之中，因为这可能会造成 MTS/COM+组件不必要的负荷。程序员在设计 MTS/COM+组件时，如果使用了数据集组件，就应该仔细地考虑一下这些组件的设定值。虽然要如何在分布式多层应用系统中设定数据集组件的特性值并没有标准的答案，而应该视系统的应用、架构等的因素来决定，但是有一些基本的原则却是可以讨论的。下面要说明的就是这些基本的原则。程序员在开发 MTS/COM+的应用系统时，应该时时想到这些基本原则，除非必要，否则应该尽量遵照这些基本的原则来设定数据集组件的特性值。

1、ResolveToDataSet：序员应该设定 TDataSetProvider 组件的 ResolveToDataSet 特性值为 True，本书前面已经说明过，要开发 MTS/COM+应用系统最好是使用真正的 ADO 驱动程序，因为 BDE/IDAPI 并不适合使用在 MTS/COM+中。如果程序员结合 MIDAS 来开发 MTS/COM+应用系统，那么由于 MIDAS 在更新数据回数据库中时有两种方式可以完成更新的动作，即使用 MIDAS 自己产生的 SQL 命令，或通过数据集组件本身更新数据，这由 TDataSetProvider 组件的 ResolveToDataSet 特性值来决定。因此程序员必须决定要使用哪一种方式来更新数据。由于 ADO 对于从多个数据表 Join 来的数据都会适当地把数据正确地更新回每一个数据表，不管后端的数据库是 File-Based 或是真正的 SQL 服务器，而这是 BDE/IDAPI 无法做到的，因此我建议程序员把 TdataSetProvider 组件的 ResolveToDataSet 设定为 True，让 ADO 自动产生更新数据的 SQL 命令。这样不但省事，而且 ADO 会自动参加 MTS/COM+的事务管理，正确地更新数据。此外，把 ResolveToDataSet 设定为 True 也可以避免 MIDAS 在 MTS/COM+中触发额外事务管理的臭虫。

2、CursorLocation：在本书前面讨论 ADO 的章节中已经介绍了 CursorLocation。在 MTS/COM+分布式多层应用系统中，不管你是使用 MIDAS、RDS 或其他的方法来处理数据，这些数据都会由客户端的引擎(MIDAS，RDS 等)来负责处理。中间的 MTS/COM+组件在传递数据给客户端之后就会自动被释放或被其他的客户端重复使用，因此 ADO 本身的 CursorLocation 就没有什么作用了。所以建议各位在分布式多层应用系统中千万不要设定 CursorLocation 为 clUseServer。这样不但没有用，而且会减慢 MTS/COM+组件存取数据的速度。

3、CursorType：由于在 MTS/COM+应用系统中不像主从架构一样是持续地和数据库连接，因此客户端浏览或处理的数据只是 cache 在客户端内存中的数据。所以在 MTS/COM+中的 MTS/COM+组件在使用 TADOQuery 或 TADOTable 存取数据时，便不需要设定等级较高的 CursorType，这样不但没有作用，反而增加了 MTS/COM+组件的负担。而且客户端的引擎(MIDAS 或 RDS)都有自己维护 cursor 的机制，因此在 MTS/COM+中我们应该只使用最经济、负担最小的 cursortype。因此建议你只需要使用 ctStatic 这种 cursortype 即可。

4、IsolationLevel：IsolationLevel 设定的原因也和前面说的一样，在客户端应用程序中处理的数据只是 cache 在客户端的数据。在分布式多层应用系统中客户端的数据一定是和数

据库服务器没有直接连接的。程序员如果使用太严格的 IsolationLevel 也没有多大的作用，反而会增加系统使用的资源(锁定机制)，因此建议在 MTS/COM+组件中使用的 ADO 数据集组件只使用 ilCursorStability 或 ilReadCommitted 即可。如果程序员真的很在意数据的完整性，那么可以撰写较多的数据处理程序代码以及数据例外处理程序代码。

5、 KeepConnection：在 MTS/COM+分布式多层应用系统中，MTS/COM+在执行完工作之后就应该关闭数据库和数据库的连接，并且被释放。因此 KeepConnection 这个特性值可以设定为 False，以避免不必要的负担。

6、 LockType：当然，对于 LockType 来说，MTS/COM+组件使用的 ADO 数据集应该使用 ltOptimistic 或 ltBatchOptimistic，因为分布式多层应用系统中客户端的数据在被修改时是和数据库没有直接连接的，因此使用 ltPessimistic 没有作用，反而会在 MTS/COM+组件存取数据时使用了不必要的锁定资源。这会造成系统的执行效率降低，并且增加数据库服务器的负担。

7、 MarshalOptions：如果使用 MIDAS 做为 MTS/COM+应用系统中传递数据的技术，那么由于客户端应用程序或浏览器传递到 MTS/COM+组件中的数据已经只是经过修改的数据，因此可以设定这个特性值为 moMarshalModifiedOnly，以减少 ADO 驱动程序不必要的工作。

虽然，即使是使用这些 ADO 数据集组件的默认属性值设定也可以让 MTS/COM+分布式多层正确地执行，但是程序员如果做适当的设定，那么不但可以增加执行效率，还能够降低数据库服务器不必要的负担，进而增加系统的延展性。举手之劳又何乐而不为呢？

在前面的小节中我们已经讨论了如何有效率地开发 MTS/COM+组件以及在分布式多层应用系统中存取数据。在下一小节中将会讨论如何在分布式多层应用系统中有效率地处理状态信息。

10-4、状态信息

在前面的小节中我们介绍了如何以少量、多次的方式在分布式多层应用系统中处理数据，这个范例就是运用状态信息的范例。虽然 MTS/COM+建议程序员尽量开发和使用无状态对象，但是在许多场合应用系统可能仍然需要使用状态对象。在 MTS/COM+分布式多层应用系统中使用状态对象并不是什么不正确或错误的事情，只是程序员应该比较小心地使用状态对象。如果真的需要使用状态对象，就应该使用它。但是要使用状态对象，程序员必须处理如何储存状态信息的问题。因为 MTS/COM+组件会被释放或重复使用，因此如果客户端应用程序需要把 MTS/COM+组件当成是状态对象来使用，那么必须负责把 MTS/COM+组件的状态回复到上一次执行结束之后的状态。这就有赖于使用状态信息了。在前面小节的范例中我们是把状态信息储存在客户端应用程序中，但是在“设计 MTS/COM+对象和 MTS/COM+应用系统”一章中我们是使用 SPM 来储存状态信息。事实上，MTS/COM+应用系统有许多种不同的方式储存状态信息，每一种方式都有其优缺点。不过大致上状态信息可以储存在下列的三个地方：

- ► 储存在客户端应用程序中—可以储存在变量、文档或系统注册表中
- ► 储存在中介服务器中—可以储存在 SPM、文档或系统注册表中
- ► 储存在后端数据库服务器中

下面的表格描述了每一种方法的优缺点：

状态信息储存的位置	*优缺点*	*适用场合*
客户端应用程序	每一个客户端调用远程 MTS/COM+组件时都必须传递状态信息给 MTS/ COM+组件，因此需要频繁并且大量地传递数据。但是对于 MTS/COM 和数据库的负荷是最轻的，因为每一个客户端应用程序分别储存各自的状态信息	客户端需要处理并且计算状态信息。在这种情形下比较适合使用这种方式
MTS/COM+中(使用 SPM)	SPM 必须储存大量客户端应用程序的状态信息。由于 SPM 的延展性并不好，因此在客户端众多或状态信息很多时，容易造成 MTS/COM+的沉重负担。因为 SPM 储存的信息在一段时间没有使用之后便会自动被释放，因此状态信息容易自己消失	并不建议大量使用 SPM，除非状态信息不多，并且会经常使用
后端数据库中	MTS/COM+必须频繁地建立数据库连接并且存取数据库，比起前面两个方案来说执行速度比较缓慢。但是通过 MTS/COM+的数据库 Pooling 可以加快存取的速度。此外，使用数据库储存状态信息也是延展性最好的，因为数据库本来就适合储存信息	客户端众多而且分布式多层应用系统需要处理大量的状态信息。但是如果你的系统真的要处理大量的描述信息的话，我建议你重新检查一下你设计的 MTS/COM+组件和应用系统架构

当程序员真的需要使用状态对象并且储存状态信息时，必须根据分布式多层应用系统的执行特性、系统的建制状态以及执行效率和延展性做适当的考虑和选择。

10-5、结论

本章讨论了许多开发 MTS/COM+多层应用系统重要的概念，这些概念对于 MTS/COM+多层应用系统的执行效率有非常重大的影响。适当使用这些概念在应用程序中将会大大增加应用系统的响应时间以及执行效率。本章从数个不同的角度来讨论如何增加 MTS/COM+多层应用系统的效率，包括如何正确地建立和使用 MTS/COM+组件，MTS/COM+多层应用系统应该如何处理数据的存取，MTS/COM+的数据库 Pooling 机制，以及如何在 MTS/COM+多层应用系统中以少量、多次的方式来处理数据。

程序员只要把这些基本的原则牢记在心，并且不断地积累实际的经验，我相信一个用心的 Delphi 程序员一定能够开发出高效率的分布式多层应用系统。