# 多层分布式应用系统中对象重用技术的研究

祝建中 (杭州师范学院 信息工程学院 310036)

摘要：在分析了基于MIDAS中间件的多层分布式系统结构的基础上，讨论了利用对象池和无状态对象技术实现应用服务器中的远程数据模块对象重用的原理与方法，并给出了应用示例。

关键词：应用服务器　对象池　无状态对象　MIDAS

## 1 引言

随着电子商务和企业网络应用需求的不断增长和计算机网络技术、分布式计算技术的迅速发展，如图1所示的多层分布式结构已经成为当前流行的具有数据分析、控制分布特征的应用系统结构之一。在这样的分层系统中，客户端实现系统的表示逻辑，应用服务器实现系统的应用逻辑，数据库服务器实现系统的数据逻辑，具有系统易维护、易扩展等明显的优点。

作为中间层的应用服务器，承担着响应客户请求、实现与数据库的连接、实现系统应用逻辑的重任，这些功能都是由应用服务器中的远程数据模块对象及其内含的数据访问、数据代理对象来实现的。每当响应客户的请求时，应用服务器就需要使用这些对象来与数据库建立连接、访问数据表、实现应用逻辑，服务完成后就需要撤消与数据库的连接并释放这些对象。这种对象的创建和释放、与数据库连接的建立和撤消都将消耗应用服务器的时间和资源，再则，数据库服务器允许的同时连接数有一定的限制，如果对每个客户的请求应用服务器都为之创建对象、新建与数据库的连接，那么当同时连接的客户数达到一定的数量时，应用服务器的时间、内存、与数据库的连接数等资源将变得极其紧张，整个系统的效率将严重下降。

本文重点研究在基于MIDAS (Multi-tiered Distributed Application Services) 中间件的多层分布式系统中，如何利用对象池 (Object Pooling) 技术和无状态对象 (Stateless Objects) 重用应用服务器中的远程数据模块对象的原理与方法。

## 2 基于MIDAS的多层分布式系统结构

多层分布式系统中常见的中间件有MTS、CORBA、MIDAS等，它们除了支持交易控制、容错能力、负载均衡以外，还通过对象池、数据库连接池、线程池等技术提高整个系统的性能。采用MIDAS 3中间件的多层分布式应用系统的构成如图2所示。

应用服务器是系统的核心，主要由Remote Data Module (简记为RDM) 构成，该模块含有一个支持无状态对象的IAppServer接口负责与客户端通信。RDM是一个容器，它通常可含有若干负责按企业应用逻辑连接并访问远程数据库服务器的Data Access对象，若干充当数据代理的DataSetProvider对象，这些DataSetProvider对象的作用是：一方面，响

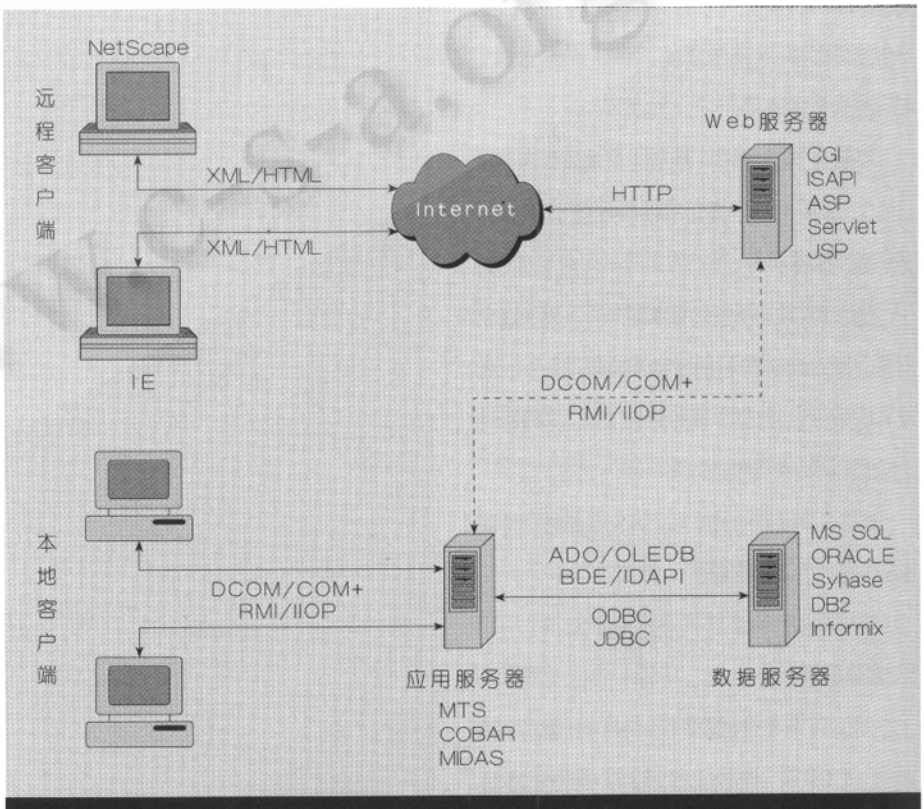


图1 多层分布式应用系统的结构

应客户端的数据请求，将通过Data Access对象从数据库服务器获取的数据封装成数据包后传回到客户端的ClientDataset；另一方面，向数据库服务器提交客户端ClientDataset中需要更新的数据，并把因各种原因不能实现更新的数据存入日志，向客户端返回这些不能实现更新操作的信息。

客户端由三部分组成：

(1) 在Form中，由Data Controls组件(如DBNavigator、DBGrid等)形成与用户交互的接口，

(2) 在Data Module容器中，包含与应用服务器建立连接的RemoteServer对象，它与应用服务器的IAppServer接口连接，以此进一步获得DataSetProvider接口，从而实现数据的获取和更新，

(3) 实现数据的存取、编辑、浏览、约束、过滤等功能的ClientDataSet对象，一旦通过RemoteServer与应用服务器建立了连接，ClientDataSet就能经由应用服务器中的DataSetProvider组件和Data Access组件访问远程数据库服务器。

## 3 远程数据模块的重用技术

在利用MIDAS 3创建应用服务器的RDM时，可以选择创建single instance或multiple instance对象。若选择单实例，则一个应用服务器进程只拥有一个RDM对象，只能为单个客户请求服务，即对每个客户请求需要创建一个新的应用服务器进程，系统开销大；若选择多实例，多个客户请求可以共享一个可同时拥有多个RDM对象的应用服务器进程，每个客户请求单独使用其中的一个RDM对象，系统开销相对较小。但在两种情况下，都需要为每个客户请求新建一个RDM对象，这涉及内存分配、数据库连接、用毕后释放等开销，如果能够实现RDM对象的重用，就可以避免这类开销，有效地提高应用服务器的性能。

MIDAS 3利用对象池(Object Pooling)技术提供RDM对象的重用机制：在应用服务器创建时，建立一个RDM对象池，每当一个客户请求到达时，MIDAS服务器检查池中是否存在空闲的RDM，若有则为之分配一个；如果没有，则在池中的对象数目未达到上限时就创建一个RDM对象，若池已满则产生"服务器忙"异常。一个RDM为客户服务完毕后就释放回RDM池，等待下一个客户请求。如果池中的某个RDM在系统设定的周期内一直没有客户端的请求可响应，则MIDAS对象池就自动将其清除出池。

MIDAS 3在创建应用服务器时，自动为RDM产生结构如下列程序段所示的UpdateRegistry方法，只是其中的语句：RegisterPooled (ClassID,16,30) 和UnregisterPooled (ClassID) 是我们添加的。

```
class procedure TAppServer.UpdateRegistry
(Register.Boolean;
const ClassID,ProgID:string);
begin
if Register then
begin
inherited UpdateRegistry(Register,ClassID,
ProgID);
EnableSocketTransport(ClassID);
EnableWebTransport(ClassID)
RegisterPooled(ClassID,16,30);
    // 池中对象数上限为16，超时限制为30
分钟
end else
begin
```

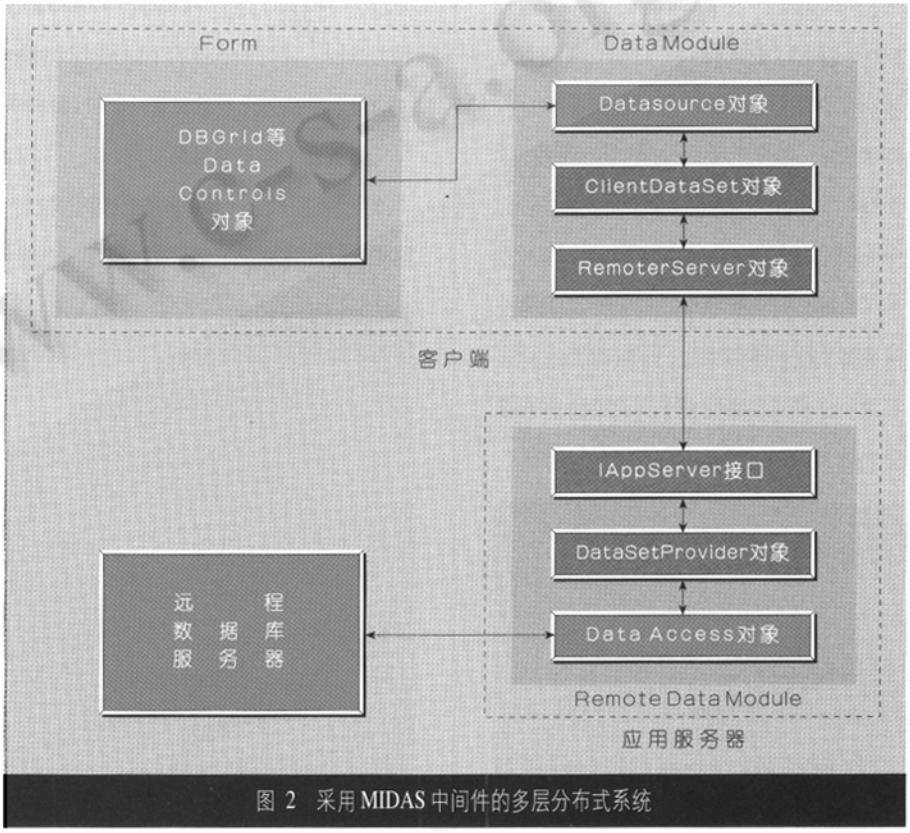


图2 采用MIDAS中间件的多层分布式系统

```
UnregiserPooled(ClassID);
DisableSocketTransport(ClassID);
DisableWebTransport(ClassID);
inherited UpdateRegistry(Register,ClassID,
ProgID);
end;
end;
```

添加这两条语句的目的是在注册应用服务器时调用RegisterPooled函数建立对象池，在注销应用服务器调用UnregisterPooled函数撤消对象池。函数RegisterPooled的第一个参数ClassID是UpdateRegistry方法传递过来的类标识符，第二个参数MaxInstances指定池中对象数目的上限、第三个参数TimeOut指定超时限制，如果对象池中的空闲RDM实例在TimeOut之后仍无客户请求可服务，则MIDAS服务器将自动释放这些空闲超时对象。TimeOut为0表示永不超时。在此例中，MaxInstances设定为16，TimeOut设定为30分钟。实际应用时，可以根据系统的具体特征通过建立配置文件的方式动态设定。

## 4 无状态对象的应用

无状态对象(Stateless Objects)是多层分布式应用系统中实现对象重用的关键技术，这种对象在完成了客户端请求的服务之后，就不再为该客户端维持任何状态信息，因此，当它下一次为其他客户端请求服务时，就如同一个新创建对象。如果应用服务器不使用无状态对象，则这些对象在客户端应用程序没有结束之前就无法释放，那么当应用服务器连接了一定数量的客户端后，就再也无法为其他户端提供服务了。而采用无状态对象就可以在使用之后立即释放给其他客户端使用，从而提高应用服务器可同时服务的客户端请求的数目，增加系统的可扩展性。

MIDAS 3的RDM默认是无状态对象，因此用它开发的应用服务器可以利用MTS、COBAR、MIDAS的对象池功能，从而被多个客户端应用程序重复使用。

要RDM成为无状态对象，则其包含的组件对象必须是无状态对象。采用MIDAS 3为中间件时，客户端中的TClientDataSet对象能够控制它所连接的应用服务器中的TDataSetProvider对象是状态对象还是无状态对象。当TClientDataSet对象的FetchOnDemand属性值为true时，表示需要应用服务器为之维持状态信息，此时它所连接的TDataSetProvider对象就自动成为状态对象，这种情形下的RDM就无法利用中间件的对象池实现重用。相反，如果TClientDataSet对象的FetchOnDemand属性值为false，则它所连接的TdataSetProvider对象就自动成为无状态对象，此时RDM对象可以被多个客户端重用。

显然，应用服务器端采用无状态对象后，客户端应用程序必须自己来维护其状态信息，从而在需要应用服务器提供服务时，客户端与应用服务器如何实现状态信息的通信已成为关键。

表1列出了Delphi 5中TClientDataSet类用于将状态信息从客户端传递到服务器的8对事件。对应地，在服务器端TDataSetProvider类中可以找到与表1中前5对同名的10个事件。

TClientDataSet类的前5个BeforeXX事件句柄，用于设置供服务器端对应的BeforeXX事件句柄需要获取的客户端保存的状态信息；TClientDataSet类的前5个AfterXX事件句柄用于读取服务器端对应的AfterXX事件句柄设置的回传状态信息。对应地，TDataSetProvider类的BeforeXX事件句柄，用于读取客户端对应的BeforeXX事件句柄所设置的客户端保存的状态信息；TDataSetProvider类的AfterXX事件句柄用于设置供客户端对应的AfterXX事件名柄读取的回传状态信息。

在上述两端交换状态信息的20个事件句柄中，都有一个名为OwnerData的OleVariant类型的参数，用于来回传递状态信息。

例如，在全校选课系统中，多个学生可能

表1 TClientDataSet类的状态事件

| BeforeXX事件 | AfterXX事件 |
|---|---|
| BeforeApplyUpdates | AfterApplyUpdates |
| BeforeExecute | AfterExecute |
| BeforeGetParams | AfterGetParams |
| BeforeGetRecords | AfterGetRecords |
| BeforeRowRequest | AfterRowRequest |
| BeforeRost | AfterPost |
| BeforeRefresh | AfterRefresh |
| BeforeScroll | AfterScroll |

通过各自的客户端同时浏览数据库服务器中的公选课数据表。由于每门公选课的介绍信息较多，客户端每屏同时只能显示其中2门课的介绍信息，各客户都有可能在每一屏信息上停留一定的时间，然后再进入下一屏，因此每个客户都需要维持当前记录的状态信息。如果不采用无状态对象，则应用服务器必须为每个连接创建一个RDM对象并一直维持到客户端应用程序结束，这样当许多同学同时浏览选课介绍时，就可能出现应用服务器负荷过重的情况。

若采用MIDAS 3支持的无状态对象技术，在应用服务器端的RDM中加入一个TTable对象PubCourseTable，使之与数据库服务器中的公选课数据表连接；再加入一个TDataSetProvider对象PubCourseProvider，并设置它的DataSet属性为PubCourseTable。将客户端的TClientDataSet对象ClientDataSet1的FetchOnDemand属性值设置为false，PacketRecord属性值设置为2，ProviderName属性设置为PubCourseProvider。这样，应用服务器的RDM就成为无状态对象，客户端每次从PubCourseTable获取2记录后，该RDM就不再为其维护任何状态信息，因而又可以响应其他客户的请求了。

每当客户端要获取下2门课的信息时，就需要通过客户端ClientDataSet1的BeforeGetRecords事件句柄，利用其参数OwnerData设置本客户端当前显示的最后一条记录的位置状态，供应用服务器端定位下一个数据包时使用：

```
procedure TForm1.ClientDataSet1BeforeGet
Records(
Sender.TObject; var OwnerData:OleVariant);
var CurRecord:TBookMark;
begin
with Sender as TClientDataSet do
begin
CurRecord:=GetBookMark;// 记录光标位置
try
Last; // 光标定位到客户端的最后一条记录
将当前记录的CourseNo值设置为要传递的
状态信息：
OwnerData:=FieldByName('CourseNo').Value;
GotoBookMark (CurRecord); // 光标位置复原
finally
FreeBookMark (CurRecord);
end
end
end;
```

其中的CourseNo是公选课数据表中的课程编号，是该数据表的索引字段。接着，客户端就可以调用GetNextPacket方法，请求获取下2条记录。

应用服务器收到客户端的获取下一个数据包的请求后，首先触发其RDM中PubCourseProvider对象的BeforeGetRecords事件：

```
procedure TAppServer.PubCourseProvider
BeforeGetRecords ( Sender.TObject;var
OwnerData:OleVariant); begin if not(VarIsEmpty
(OwnerData)) then with Sender as
TDataSetProvider do begin
// 获取客户端BeforeGetRecords事件通过参数
OwnerData传递的状态，定位该客户端当前最
后一条记录在PubCourseTable中的位置：
DataSet.Locate ('CourseNo',OwnerData,[]);
DataSet.Next; // 定位回传数据包的记录起始
位置
end
end;
```

然后从PubCourseTable数据表获取一个含2条记录的数据包，经由PubCourseProvider数据代理对象回传客户端，至此，PubCourseProvider对象的本次服务完毕，可接着为下一个客户端请求服务了。

## 5 小结

应用实践表明，充分利用MIDAS 3的对象池技术和对无状态对象的支持，实现应用服务器的RDM对象的重用，能够更有效地利用应用服务器的资源，提高其同时可服务的客户端数量，从而提升整个系统的性能。

参考文献

1 李维，Delphi 5.x分布式多层应用系统篇[M]，机械工业出版社，2000。

2 左彦忠、盛智，基于MIDAS分布式多层系统执行效率的研究[J]，计算机应用，2001，21(5):84-85。