神通数据库

.Net Data Provider开发指南

版本 7.0

天津神舟通用数据技术有限公司

2010 年 1 月

版权声明

前 言

.Net Data Provider 表示.net 数据库连接，由一组用.net 编程语言编写的类和接口组成。.Net Data Provider 为.net 程序访问关系型数据库提供了编程接口，为数据库开发人员提供了一个标准的 API，使他们能够用.net API 来编写数据库应用程序，而不必为访问不同数据库编写不同的程序。

随着 Framwork 版本的不断发展，.Net 规范也不断完善。发展到目前为止,.Net Data Provider 规范已经发到 3.5 版本。该版本为 2.0 版本，提供了很多特性，比如保存点、查询计划的参数信息、缓存结果集、多个活动结果集 (MARS)等。

神通数据库 .Net Data Provider 驱动程序基本实现了 Ado.net 2.0 规范，给出了很好的支持。对于这些新的特性的支持，为用户提供了更为方便的编写数据库应用程序的方法，使得用户编写应用程序将更加高效，编写的应用程序也更加健壮。

本手册结合神通数据库数据库的特点，介绍了.Net Data Provider 的基本概念和基本技术，帮助用户更好地学习和使用神通数据库数据库。

本手册适用于所有神通数据库数据库的用户。

阅读指南

〖阅读对象〗

本手册的阅读对象为神通数据库用户，用户若对.NET 和数据库基本概念有一定了解，将对本手册的理解有很大的帮助。

〖内容简介〗

通过本手册了解.Net Data Provider 的基本概念和基本技术，本指南示例代码采用 C#，对于其他.Net 开发语言同样适用，帮助用户更好地学习和使用神通数据库数据库。

本手册适用于所有神通数据库数据库的用户。

〖手册构成〗

本手册由 9 部分组成：

第 1 章，“概述”，简单介绍 ADO.NET 和.Net Data Provider。

第 2 章，“神通数据库 PROVIDER Driver 的安装与配置”。

第 3 章，“神通数据库 .NET DATA PROVIDER 快速入门”，介绍了如何使用神通数据库 PROVIDER Driver 以及如何处理由 PROVIDER 抛出的异常。

第 4 章，“执行 SQL 语句和处理结果集”，主要介绍神通数据库 .NET DATA PROVIDER 驱动如何执行 SQL 命令。

第 5 章，“执行存储过程”，介绍存储过程的执行方式，以及 IN、OUT 和 INOUT 参数的用法。

第 6 章，“处理结果集”，本章介绍了几个结果集对象的创建执行以及用法。

第 7 章，“大对象”，介绍了对大对象进行存储和读取等。

第 8 章，“事务特性”，主要介绍了事务机制的特性和使用方法。

第 9 章，“标量函数”，主要介绍各种标量函数的使用方法。

〖相关文档〗

使用本手册时可以参考神通数据库的手册集，手册集包含以下文档：

《神通数据库数据库安装手册》

《神通数据库数据库备份恢复工具使用手册》

《神通数据库数据库 DBA 管理工具使用手册》

《神通数据库数据库系统管理员手册》

《神通数据库数据库嵌入式 SQL 语言手册》

《神通数据库数据库交互式 SQL 查询工具使用手册》

《神通数据库数据库过程语言手册》

《神通数据库数据库 OLEDB/ADO 用户手册》

《神通数据库数据库迁移工具使用手册》

《神通数据库数据库审计管理指南》

《神通数据库数据库 ODBC 程序员开发指南》

《神通数据库数据库 SQL 语言参考手册》

《神通数据库数据库审计工具使用手册》

《神通数据库数据库性能监测工具使用手册》

《神通数据库数据库作业调度工具使用手册》

〖手册约定〗

本手册遵循以下约定：

所有标题均使用黑体字。

如果标题后跟有“〖条件〗”字样，说明该标题下正文所要求的内容只是在一定条件下必须的。

【注意】的意思是请读者注意那些需要注意的事项。

【警告】的意思是请读者千万注意某些事项，否则将造成严重错误。

【提示】的意思是提供给读者一些实用的操作技巧。

对于手册中出现的正文和程序代码，遵循如下约定：

表 0-1 手册正文约定

约定	含义	范例
粗体	表示强调	确保控制文件和数据文件不要驻留在同一个磁盘上。
大写等宽字母	表示由系统提供的元素，如参数、权限、数据类型、关键词、命令、函数名，以及由系统提供的列名、数据库对象和结构等。	利用 BACKUP 命令备份数据库。 在 USER_TABLES 数据字典视图中查询 TABLE_NAME 列
小写等宽字母	表示执行程序、文件名、目录名以及需要由使用者提供的元素，包括计算机名、数据库名、数据库对象和结构、列名、程序单元以及参数等。 【注意】：某些元素要求使用大写或者大小写混合形式。此时，应当根据实际的要求输入。	department_id， department_name 以及 location_id 列在 departments 表中。
小写等宽斜体	表示占位符或者变量	

表 0-2 手册中出现的程序代码的书写约定

约定	含义	范例
[]	表示包含一个或者多个可选项。不需要输入中括号本身。	DECIMAL (digits [, precision])
{}	表示包含两个以上（含两个）的选项，其中有一个是必须的。不需要输入花括号本身。	{ENABLE | DISABLE}
|	分割中括号或者花括号中的两个或者两个以上的选项。不需要输入“|”本身。	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]
…	表示省略 表示重复	CREATE TABLE … AS subquery; SELECT col1, col2, … , coln FROM employees;
. . .	表示省略了若干行	
斜体	表示占位符或者需要提供特定值	CONNECT SYSTEM/system_password

	的变量	DB_NAME = database_name
大写	表示系统提供的元素，主要是为了与使用者定义的元素相互区分。除了出现在方括号中的元素外，应当按照顺序逐字输入。当然，有些元素在系统中是大小写不敏感的，因此使用者也可以根据系统说明以小写的方式输入。	SELECT last_name, employee_id FROM employees; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;
小写	表示由使用者提供的元素。例如：表、列和文件的名字。 【注意】：根据某些具体的要求，有些由使用者提供的元素可能要求使用大写或者大小写混合的形式。此时，应当根据实际的要求输入。	SELECT last_name, employee_id FROM employees; CREATE USER tom IDENTIFIED BY a8M9j7

目录

第1章 概述

本章内容包括：

1. ADO 与 ADO.NET 简介
2. 神通数据库 .Net Data Provider 简介

1.1ADO 与 ADO.NET 简介

ADO 使用 OLE DB 接口并基于微软的 COM 技术，而 ADO.NET 拥有自己的 ADO.NET 接口并且基于微软的.NET 体系架构。ADO.NET 和 ADO 是两种数据访问方式。在开始设计.NET 体系架构时，微软就决定重新设计数据访问模型，以便能够完全的基于 XML 和离线计算模型。两者的区别主要有：

ADO 以 Recordset 存储，而 ADO.NET 则以 DataSet 表示。Recordset 看起来更像单表，如果让 Recordset 以多表的方式表示就必须在 SQL 中进行多表连接。反之，DataSet 可以是多个表的集合。ADO 的运作是一种在线方式，这意味着不论是浏览或更新数据都必须是实时的。ADO.NET 则使用离线方式，在访问数据的时候 ADO.NET 会利用 XML 制作数据的一份幅本，ADO.NET 的数据库连接也只有在这段时间需要在线。

由于 ADO 使用 COM 技术，这就要求所使用的数据类型必须符合 COM 规范，而 ADO.NET 基于 XML 格式，数据类型更为丰富并且不需要再做 COM 编排导致的数据类型转换，从而提高了整体性能。

ADO.NET 提供对诸如 SQL Server 和 XML 这样的数据源以及通过 OLE DB 和 ODBC 公开的数据源的一致访问。通过数据处理将数据访问分解为多个可以单独使用或一前一后使用的不连续组件。 ADO.NET 包含用于连接到数据库、执行命令和检索结果的 .NET Framework 数据提供程序。这些结果或者被直接处理，放在 ADO.NET DataSet 对象中以便以特别的方式向用户公开，并与来自多个源的数据组合；或者在层之间传递。ADO.NET 则使用离线方式，在访问数据的时候 ADO.NET 会利用 XML 制作数据的一份幅本，ADO.NET 的数据库连接也只有在这段时间需要在线。

ADO.NET 成为数据访问技术进化过程的下一个阶段，那么 OLE DB 在 .NET 中又是什么情况呢？在 .NET 中，Web 应用程序主要是断开连接的应用程序，该应用程序使用新设计的特别工具来管理数据。 .NET 框架提供了处理数据的类。这些类（特别是 ADO.NET 和 XML 命名空间）提供了收集、读取和写入功能。ADO.NET 和 XML 子系统最终取代了 ADO 和 OLE DB SDK，因此，现在使用一种与语言无关的方法来获取或设置数据。ADO.NET 类在提取数据源方面甚至比 ADO 做得更好，这是因为它具有以数据为中心的明确设计，与此相反，ADO 却仍然使用以数据库为中心的模型。

ADO.NET 通过数据处理将数据访问分解为多个可以单独使用或一前一后使用的不连续组件。ADO.NET 包含用于连接到数据库、执行命令和检索结果的 .NET Framework 数据提供程序。您可以直接处理检索到的结果，或将其放入 ADO.NET DataSet 对象，以便与来自多个源的数据或在层之间进行远程处理的数据组合在一起，以特殊方式向用户公开。ADO.NET DataSet 对象也可以独立于 .NET Framework 数据提供程序使用，以管理应用程序本地的数据或源自 XML 的数据。

1.2神通数据库.Net Data Provider 简介

通过神通数据库 .NET Framework 数据提供程序(.net Data Provider)，可以使用自己的内部协议访问 神通数据库 6.0 版或更高版本的数据库。该数据提供程序设计的功能与 OLE DB、ODBC 和 Oracle 的 .NET Framework 数据提供程序的功能类似。

ADO.NET 对象模型中提供五个主要的组件，分别是 Connection 对象、Command 对象、

DataReader 对象、DataAdapter 对象以及 DataSet 对象。这些组件中负责建立联机和数据操作的部分我们称为数据操作组件（Managed Providers），分别由 Connection 对象、Command 对象、DataAdapter 对象以及 DataReader 对象所组成。数据操作组件最主要是当作 DataSet 对象以及数据源之间的桥梁，负责将数据源中的数据取出后植入 DataSet 对象中，以及将数据存回数据源的工作。

第2章 神通数据库 PROVIDER DRIVER 的安装与配置

安装神通数据库 PROVIDER DIRVER 有两种方法，用户既可以通过神通数据库安装程序来安装和配置神通数据库 PROVIDER DRIVER，也可以通过手动方式来安装和配置神通数据库 PROVIDER DRIVER。

2.1通过神通数据库安装程序安装驱动程序

运行神通数据库安装程序（服务器或者客户端安装）时，神通数据库的.NET Provider 驱动程序会被安装到神通数据库安装目录的 Provider 目录下，例如在 Windows 平台下，将神通数据库安装到 C:\ ShenTong 的目录下， 则神通数据库 的.Net Data Provider 驱动 System.Data.OscarClient.dll 被安装到 C:\ ShenTong\ dotNetProvider 目录下，同时会有一个环境变量 SZ_OSCAR_HOME 指向产品的安装目录。

2.2手动安装驱动程序

如果用户没有运行神通数据库安装程序，那么就可以通过手动方式来安装驱动程序。

用户只需从神通数据库安装目录中的 dotNetProvider 目录下将 System.Data.OscarClient.dll 文件手动拷贝到某个目录下，例如拷贝到 c:\Provider 目录下，然后在.net 项目的解决方案资源管理器单击右键，添加引用，在.net 选择项下选择 System.Data.OscarClient.dll。在引用文件中引入 “using System.Data.OscarClient” 。

第3章 神通数据库 .Net Data Provider 快速入门

本章内容包括：

1. 神通数据库 .Net Data Provider 的使用
2. 示例：连接、查询、处理结果集
3. 数据库类型的映射
4. 处理 SQL 异常
5. 神通数据库 .Net Data Provider 的存储过程
6. 神通数据库 .Net Data Provider 的配置文件

3.1神通数据库 .Net Data Provider 的使用

本节主要介绍应用程序如何使用神通数据库 .Net Data Provider 驱动来访问数据库服务器。当使用神通数据库 .Net Data Provider 驱动的时候，用户必须在应用程序里设置一系列关于驱动程序的信息。本节将按照步骤来描述如何使应用程序连上神通数据库数据库，并从数据库中获取和存储数据。对于数据的获取和处理采用”在线”(连接数据对象)和离线（非连接数据对象）两种方式。

要使应用程序（客户端）连接上数据库服务器，并从数据库服务器中查询数据或修改数据，通常需要以下几个步骤：

1. 引用 .Net Data Provider 命名空间
2. 与后台数据库建立连接
3. 创建一个 OscarCommand 对象
4. 执行查询语句，并取得 OscarDataReader 对象,或者 DataSet
5. 处理获得的结果集
6. 关闭 OscarDataReader 对象
7. 改变数据库的数据内容
8. 提交(commit)对数据库做的改变
9. 关闭与后台数据库的连接
10. 离线（非连接）数据对象的使用

3.1.1 引入 PROVIDER 命名空间

不管使用哪个厂家的驱动程序，或者哪种类别的驱动程序，只要应用程序需要连接上数据库，就应当在应用程序的开始处引用

```
using System.Data.OscarClient;
```

3.1.2 与后台数据库建立连接

打开一个与后台数据库的连接，取得一个 OscarConnection 类的实例(Instance)，这个实例就在应用程序和后台数据中建立了一个连接，使用这个连接，我们就能操作数据库了。要得到 OscarConnection 的实例，参数是一个连接字符串，连接字符串需要输入用户名、密码等等。连接字符串的格式是使用分号分隔的键/值参数对列表：

keyword 1=value 1; keyword 2=value 2…keyword n=value n;

关键字不区分大小写，并将忽略键/值对之间的空格。不过，根据数据源的不同，值可能是区分大小写的。

有效的连接字符串语法是因提供程序而异的，从早期的 API（如 ODBC）开始，已经过了多年的发展演变，其常见连接字符串的语法更具灵活性。为使用户更加方便，这里为一些关键字添加了一些别名，如 Password，还可以使用人们经常使用 pwd 等。

其中：

参数	参数说明
host	服务器的主机名。
server	服务器的主机名。
Data source	服务器的主机名。
port	服务器监听的端口号。 缺省时是神通数据库标准的端口号（2003）
database	数据库名。此处的数据库名就是在安装时创建的数据库所对应的名字
DB	数据库名。此处的数据库名就是在安装时创建的数据库所对应的名字
Initial catalog	数据库名。此处的数据库名就是在安装时创建的数据库所对应的名字
USERNAME	数据库的登陆用户名
USER	数据库的登陆用户名
USERID	数据库的登陆用户名
UID	数据库的登陆用户名
PASSWORD	相应用户名的登陆密码
PSW	相应用户名的登陆密码
PWD	相应用户名的登陆密码
SSL	布尔值，控制是否尝试安全连接。默认值=False
Pooling	布尔值, 控制是否使用数据库连接池. 默认值= True;
MinPoolSize	数据库连接池最小连接数;
MaxPoolSize	数据库连接池最大连接数
Timeout	连接打开可等待时间。默认值为 15
CommandTimeout	命令执行完成抛出一个例外之前等待时间。在几秒钟内。默认值为 20.
Sslmode	连接 SSL 模式控制。允许（Allow）或禁用（Disable）。默认为禁用。检查用户手册解释。
ConnectionLifeTime	未使用的连接等待多少秒时间后关闭（close）。默认值为秒。
ENCODING	编码方式
StmtRollBack	设置后台事务回滚方式，值为 0 时为事务级别回滚，值为 1 时为语句级别回滚。默认状态下，为后台的置的方式。

比如想要连接数据库，连接字符串：

```
String    connectionSql = "Server=127.0.0.1;Port=2003;
User Id=SYSDBA;Password=szoscar55;Database=OSRDB;Encoding=GBK;";
```

创建连接：

```
OscarConnection con = new OscarConnection(connectionSql);
```

3.1.3 创建一个 OscarCommand 对象

一旦连接上了数据库，也就是说，已经获得了一个 OscarConnection 的实例，那么就可以通过该 OscarConnection 实例创建一个 OscarCommand 对象。通过 OscarCommand 对象来处理不同的 SQL 语句。我们就用前面获得的 OscarConnection 实例 con 来创建一个 OscarCommand 对象，下面就是一个创建 OscarCommand 对象的例子：

```
OscarCommand cmd = con.CreateCommand();
```

也可以直接创建 OscarCommand 对象，然后将 OscarConnection 实例作为 OscarCommand

对象的属性赋值。或者将 sql 语句和 OscarConnection 实例作为参数来创建 OscarCommand 对象：

```
OscarCommand cmd = new OscarCommand();
        comm.Connection = con;
```

或者：

```
OscarCommand cmd = new OscarCommand (sql,con);
```

这个操作是完全遵循.net 标准规范的。

3.1.4 执行查询语句，并取得 OscarDataReader 对象

如果需要查询数据库中的数据，只需要调用 OscarCommand 对象的 ExecuteReader()方法。这个方法有两个重载，无参数和有参数（CommandBehavior 的一个值），方法执行完后返回一个 OscarDataReader 对象。为了保持连续性，我们就使用上面已经创建的 OscarCommand 对象。比如我们要从一张表名为 TEMP 的表中选出所有数据，我们只要简单的调用 ExecuteReader()方法：

```
comm.CommandText = "select * from tablea;"
OscarDataReader dr= comm .ExecuteReader();
```

同上面一样，这个操作也是完全遵循 ADO.NET 标准规范的。

注：假如我们已经在神通数据库中创建了一个名为 tablea 的表，它有两列分别为 FIELD_SERIAL(INT)，FIELD_TEXT(VARCHAR)。后面的章节将多次使用到这个表。

3.1.5 处理获得的结果集

一旦执行了 SQL 语句，获得了 OscarDataReader 对象，那么就可以通过调用 OscarDataReader 对象中的 Read()操作遍历结果集操作，以获得每条记录。如果 Read()方法返回为 true，那么就意味着现在有记录可以读取出来，接着就可以调用 OscarDataReader 对象的 getXXX()方法来取得对应列的值，XXX 代表了要取得的列的类型，该方法是以列序号或列名为参数的，下面就从上面获得的 OscarDataReader 对象中取得数据：

```
while (dr.Read())
        System.Console.WriteLine(dr.getString(2));
```

或者：

```
while (dr.Read())
        System.Console.WriteLine((string)dr[2]);
```

在这里 TEMP 表中第二列的类型 VARCHAR，所以我们使用了 dr 对象的 getString()方法，关于数据库类型和前台类型的对应关系，我们会在后面的章节中加以说明。当然这里也可以通过调用 dr[paramName]来获得值。这个方法也是遵循.Net 标准的。如果一直向下做遍历，当没有记录的时候，Read()就会返回 false。

3.1.6 关闭 OscarDataReader 对象

当不再需要使用 OscarDataReader 对象的结果集数据之后，就必须显式的关闭已经创建的 OscarDataReader 对象。PROVIDER 驱动程序没有自动地释放这些对象的功能，应用程序必须显式的调用 OscarDataReader 的 close()方法。一旦已经显式的关闭了已经创建的对象，就不能再使用这些对象了。如果已经不再使用某些 OscarDataReader 对象，但是却不去释放它，那将会造成严重的内存泄漏。如果创建了大量的 OscarDataReader 对象，但是却不释放它们，应用程序可能最终会造成 OUT OF MEMORY 的后果。

比如应用程序中的 OscarDataReader 对象是 dr，就可以这样关闭它：

```
dr.close();
```

还可以用 using 指令来指定应用范围，（using 语句允许程序员指定使用资源的对象应当何时释放资源。为 using 语句提供的对象必须实现 IDisposable 接口。此接口提供了 Dispose 方法，该方法将释放此对象的资源。）：

```
using(OscarCommand cmd =new OscarCommand())
{    <some code >      }
```

当程序跳出 using 的范围的时候，.Net framework 会自动的将 cmd,以及神通数据库 DataReader 对象自动地关闭。虽然关闭了 OscarCommand 对象时，创建该 OscarCommand 对象的 Connection 仍然与后台数据库保持连接，应用程序仍然可以用它创建其他的 OscarCommand 对象。

3.1.7 改变数据库的数据内容

用户不仅可以从后台数据库中查询数据，也可以往后台数据库插入数据，修改数据。可以简单的用上面已经创建的 OscarCommand 对象（假设它还没有关闭），来插入一条记录。

下面这个例子就是向 TABLEA 表中插入一条记录:

```
String sql = "INSERT INTO TABLEA VALUES(1,'value1')";
cmd.ExecuteNonQuery(sql);
```

当然用户也可以改变表中记录的内容，比如：

```
String sql = "UPDATE TABLEA SET FIELD_SERIAL = 10 WHERE FIELD_SERIAL = 1";
cmd.ExecuteNonQuery(sql);
```

3.1.8 提交(commit)对数据库做的改变

对于缺省情况下，DML 操作（INSERT,UPDATE,DELETE）一旦被执行，将自动被提交。当然可以自己来改变 commit 模式，比如需要执行多条语句，这些语句要求在一个事务里面，那么这时候就必须创建一个 OscarTransaction 对象。可以通过调用 Connection 对象的 BeginTransaction()方法来获得 OscarTransaction 对象。

```
OscarTransaction tran = con.BeginTransaction();
```

上述方法，创建了一个事务(Transaction)，然后就可以提交或者回滚多条语句了。比如插入 3 条记录。

```
String sql = "INSERT INTO TABLEA VALUES(1,'value1')";
cmd.ExecuteNonQuery(sql);
sql = "INSERT INTO TABLEA VALUES(2,'value2')";
cmd.ExecuteNonQuery(sql);
sql = "INSERT INTO TABLEA VALUES(3,'value3')";
cmd.ExecuteNonQuery(sql);
```

当执行完上述 SQL 语句之后，千万别忘了 commit：

```
tran.Commit();
```

如果想回退事务，就需要 rollback 了：

```
tran.Rollback();
```

3.1.9 关闭与后台数据库的连接

最后，当不再需要与数据库的连接时，就需要关闭神 OscarConnection 对象。调用 OscarConnection 的 close()方法，就会关闭连接，释放网络上的资源。

```
con.close();
```

关闭了 OscarConnection 对象，同时也会关闭由它创建的 OscarCommand 对象，这些 OscarCommand 对象不能再执行 SQL 语句了。

3.1.10 离线（非连接）数据对象的使用

首先引入 DataSet 的概念，DataSet 对象是.NET 框架提供的数据一种基于内存的关系表示，并且是主要的非连接数据对象。可以将 DataSet 视为一种内存中的关系数据库，它仅仅缓存数据库而不提供任何对于当前关系数据库的事务属性。DataSet 对象包含一个 DataTable 对象集合 DataTable 对象可以包含唯一外键约束，以保证数据的完整性。

.Net Data Provider 提供 OscarDataAdapter 类用于离线结果集的操作。OscarDataAdapter 对象用于获取和更新 DataSet 对象的 DataTable 对象。OscarDataAdapter 类继承 DbDataAdapter 类，OscarDataAdapter 对象拥有 Command 对象属性，（SelectCommand 属性，InsertCommand 属性，UpdateCommand 属性，DeleteCommand 属性）。

通常采用 Fill 方法将数据从数据存储区移动到作为该方法参数的 DataSet 或者 DataTable 对象中，如果 Fill 方法时，没有指定源 DataTable 对象，则会在该 DataSet 对象中创建一个新的 DataTable 对象。

除 Fill 方法外还会经常使用 Update 方法将修改结果保存到数据源中。

下面的代码片段演示了如何使用 OscarDataAdapter 对象：

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.OscarClient;
using System.Data;
using System.Data.SqlClient;
namespace TestProviderForHelpfiles
{
    class Program
    {
        static void Main(string[] args)
        {
            UpdateWithDataSet();
        }

        public static void UpdateWithDataSet()
{
            string connectString = "Server=10.0.5.226;Port=2003;User Id=SYSDBA;Password=szoscar55;Database=OSRDB;Encoding=GBK;";
            OscarConnection TheConnection = new OscarConnection(connectString);
            TheConnection.Open();
           OscarCommand command = new OscarCommand("insert into tablea(field_text) values ('text');", TheConnection);
            command.ExecuteNonQuery();

            DataSet ds = new DataSet();

            OscarDataAdapter da = new OscarDataAdapter("select * from tablea where field_serial = (select max(field_serial) from tablea)", TheConnection);

            OscarCommandBuilder cb = new OscarCommandBuilder(da);
            //获得数据源的本地缓存 DataSet
            da.Fill(ds);
            //获得 DataTable 对象
            DataTable dt = ds.Tables[0];
            //活动 DataRow 对象
           DataRow dr = ds.Tables[0].Rows[ds.Tables[0].Rows.Count - 1];
```

```
                    //更改本地缓存数据
                    dr["field_text"] = "ChangeText";
                    //调用更改的 updateCommand
                    da.Update(ds);
                    //提交更改
                    ds.AcceptChanges();
                    //读取更改后数据
    OscarDataReader dreader = new OscarCommand("select * from tablea where field_serial =
(select max(field_serial) from tablea)", TheConnection).ExecuteReader();
                    dreader .Read();
                    //Assert.AreEqual("ChangeText", dr2["field_text"]);
                    Console.Write(dreader["field_text"].ToString());
                    dreader .Close();
                    command.Dispose();
                    TheConnection.Close();
                }
            }
        }
```

3.2示例：连接、查询、处理结果集

这一节将整合前面的例子，给出一个完整的例子，便于用户参考。

```
    using System;
    using System.Collections.Generic;
    using System.Text;
    using System.Data.OscarClient;
    using System.Data;
    using System.Data.SqlClient;
    namespace TestProviderForHelpfiles
    {
        class Program
        {
            static void Main(string[] args)
            {
                TestPROVIDER();
    }
            public static void TestPROVIDER()
            {
                string connectString = "Server=10.0.5.226;Port=2003;User
Id=SYSDBA;Password=szoscar55;Database=OSRDB;Encoding=GBK;";
                OscarConnection con = new OscarConnection(connectString);
                con.Open();
                OscarCommand cmd = new OscarCommand("SELECT * FROM TABLEA",
con);
                OscarDataReader dr = cmd.ExecuteReader();
                while (dr.Read())
                {
                    System.Console.WriteLine(dr.GetString(1));
                }
                Console.Read();
                dr.Close();
                con.Close();
            }
        }
    }
```

3.3数据类型的映射

神通数据库 .NET DATA PROVIDER 驱动支持大多数由.net framwork 规范所要求的类型。在类型映射中，我们将讨论.NET 类型、PROVIDER 类型以及后台数据库类型是如何进行相互映射的。比如前台的 String 对应于 PROVIDER 类型中的那种，又对应于后台数据库中的那种类型。

表 3-1 是神通数据库数据类型。

序号	类型	描述	有效范围
1	tinyint	整型数据	[-128,127]
2	smallint	整型数据	[-32768, 32767]
3	int	整型数据	[-2147483648, 2147483647]
4	bigint	整型数据	[-9223372036854775808, 9223372036854775807]
5	bit	固定长度的位串数据	1，0 最大长度限制是 8*8000bit
6	numeric	固定精度和小数位的数字数据	
7	decimal	固定精度和小数位的数字数据	
8	real	浮点精度数字	[-3.4E+38, 3.4E+38]
9	double precision	浮点双精度数	[2.2250738585072014e-308,1.7976931348623158e+308]
10	float	浮点精度数字	[-1E+308, 1E+308]
11	char	固定长度的字符数据	最大长度为 8000 个字符
12	varchar	可变长度的字符数据	最大长度为 8000 个字符
13	text	可变长度的字符数据	最大长度为 8000 个字符
14	binary	定长的二进制字串	最大长度为 8000
15	varbinary	变长的二进制字串	最大长度为 8000
16	boolean	布尔类型	
17	date	日期	
18	time	时间	
19	timestamp	时间戳	
20	clob	字符型大对象类型	理论上没有限制，可以达到 4 个 G
21	blob	二进制大对象类型	理论上没有限制，可以达到 4 个 G

神通数据库数据类型这一列列举了在数据库中的数据类型。

PROVIDER 类型这一列列举 PROVIDER 标准支持的，并在.net.sql.Types 类中有定义的类型。

标准.NET 类型这一列列举了在.NET 语言中定义的标准类型。

表 3-2 神通数据库数据类型、PROVIDER 类型和标准.NET 类型之间的映射。

序号	神通数据库数据类型	PROVIDER System.DbType 类型	标准.Net System Type 类型
1	tinyint	DbType.SByte	Byte
2	smallint	DbType.Int16	Int16
3	int	DbType.Int32	Int32
4	bigint	DbType.Int64	Long/Int64
5	bit	DbType.Boolean	Boolean
6	numeric	DbType.Decimal	Decimal
7	decimal	DbType.Decimal	.Decimal

8	real	DbType.Single	Single
9	double precision	DbType.Double	double
10	float	DbType.Double	double
11	char	DbType.Single	String
12	varchar	DbType.Single	String
13	text	DbType.Single	String
14	binary	DbType.Binary	Byte[]
15	varbinary	DbType.Binary	Byte[]
16	boolean	DbType.Boolean	Bool
17	date	DbType.DATE	DateTime
18	time	DbType.TIME	DateTime
19	timestamp	DbType.DateTime	DateTime
20	clob	DbType.String	String
21	blob	DbType.Binary	Byte[]

备注::

关于 binary 类型，如果用户插入的数据没有达到定义的长度，后台数据库会自动补 0，故最后读出的 byte 数组的长度为定义时的长度。

关于 char 类型，如果用户插入的数据没有达到定义的长度，后台数据库会自动补空格，故最后读出的 String 对象的长度为定义时的长度。

关于 bit 类型，对应于 DbType.Boolean，故最后读出的 bit 也为 Boolean 类型。比如在数据库中插入的为 B’01110’，通过 PROVIDER 读的时候将判断是否为 0，如果不为 0，就返回 true，否则返回 false，对上述例子就返回 true。

3.4处理 SQL 异常

在运行应用程序的时候，可能会抛出很多异常。可能由于数据库服务器没有打开导致连接不上，插入的数据类型不符，想要删除的表不存在等等。由.Net Data Provider 抛出的异常大多是 OscarException 对象。这些异常有可能是从后台数据库跑出来的，也有可能是从前台的.Net Data Provider 驱动抛出的。比如上面提到的数据库服务器连接不上就是由.Net Data Provider 驱动抛出的异常，想要删除的表不存在就是由后台数据库抛出的错误。

那么如何处理这些由.Net Data Provider 抛出的异常呢？由.Net Data Provider 抛出的 DbException 和 OscarException，一般都会包括错误描述。

通过调用 OscarException 的 ToString()方法或者 Message 属性，可以获得由驱动或者后台数据库报出的错误。通过这个错误描述，就可以知道错误的原因。

下面这例子打印出调用 Message 属性所得到的错误信息。

```
try
{
    <some code>
}
catch(OscarException e)
{
    System.Console.WriteLine("错误信息:"+e.Message);
}
```

3.5神通数据库 .NET DATA PROVIDER 中的存储过程

应用程序可以通过 PROVIDER 驱动直接调用后台的存储过程。执行存储过程可以提高程序执行的效率。尤其是极大的减少了网络数据的开销。比如有 20 个 INSERT SQL 语句，如果每次都通过 OscarCommand 对象的 ExecuteNonQuery()方法去执行，那么每次都要传送一遍，也就是说要跟后台数据库交互 20 次，而用存储过程则只需一次交互。关于存储过程

的详细内容，包括如何创建、调用存储过程，我们将在存储过程这一章中给出详细的说明。

3.6神通数据库 .NET DATA PROVIDER 中的配置文件

神通数据库.NET DATA PROVIDER 驱动的配置文件是 oscarconfig.ini，该文件存放于.NET DATA PROVIDER 驱动的同级目录下，即：神通数据库安装目录的 dotNetProvider 文件夹下。oscarconfig.ini 的内容格式如下：

```
[globalparam]
# 批量执行的缓存大小（单位：MB）
#    最大值 1024
#    最小值 1
#    默认值 10
batch_buffer=10

# 普通 execute 语句一次执行次数达到多少可以走 batch 协议
#    最大值 5000
#    最小值 1，表示不走 Batch 协议
#    默认值 100
execute_batch_size=100

# loglevel：日志级别，分为：0，1，2，3，4 四级，0 表示不打印日志，1 表示 SQL
#语句级，2 表示调用接口级，3 表示轻量协议级，4 表示详细协议级。
loglevel=0

#logFilePath:日志文件路径，默认在 c:盘根目录下
logFilePath=c:\log.txt

# maxFileSize:日志文件最大大小，单位为 MB，达到最大大小后，会使用新的日志文件
maxFileSize=100

# 数据格式是采用字符串方式传输还是采用协议的方式传输
#    默认值 off，即采用协议方式传输
net_data_by_str=off

# 小数的类型，如 float、double、numeric、decimal 是否按字符串格式传输
#    默认值 off，即采用协议格式传输
send_floatingnumber_keep_precision=off

# prepareCacheSize:preparedStatement 缓存，当 prepareCacheSize>0 时，
#该功能将启用，缓存中 preparedStatement 个数为 prepareCacheSize 个
prepareCacheSize=0

# 一次从数据库获取的结果集行数
#    最大值 5000
#    最小值 0，表示一次性获取所有结果集
#    默认值 16
fetchsize=16

# 不带参数的 prepare 语句是否可以直接走 query 协议执行 sql 语句
#    默认值 on,即直接执行 sql 语句
prepare_simple_execute=on

#驱动是否直接使用旧协议
```

```
#默认值 false ， 即使用新协议
compatible_old_protocal=false
```

配置文件已经对各参数进行了具体说明，这里不一一赘述。

【注意】：使用.NET DATA PROVIDER 驱动的配置文件 oscarconfig.ini 时，需确保该文件与应用程序使用的驱动 System.Data.OscarClient.dll 位于同一目录下，否则，应用程序无法正确获取配置文件 oscarconfig.ini 中的参数值。

第4章 执行 SQL 语句和处理结果集

神通数据库 .Net data provider 驱动提供了 OscarCommand 对象用于发送 SQL 语句到数据服务器。OscarCommand 对象的 CommandText 属性可以执行所有的 Sql 语句，包括标准的 SELECT、UPDATE、DELETE 语句。例如执行 OscarCommand 对象中的 SQL 语句，创建表，外键，主键等。

OscarCommand 对象提供了多个重载的 Execute 方法以执行希望的操作。当返回的结果为一个数据流时，可以使用 ExecuteReader 返回一个 OscarDataReader 对象；返回一个单指时可以使用 ExecuteScalar。

4.1 OscarCommand 对象可配置属性

FetchSize：

该属性在执行 select 语句，读取大量数据时可用配置服务器端游标每次获取数据量，默认情况是每次从数据库中获取 1000 条到本地内存中。

CommandText：

该属性为要执行的文本命令。默认值为空字符串 ("")。 当将 CommandType 设置为 StoredProcedure 时，应将 CommandText 属性设置为存储过程的名称。当调用 Execute 方法之一时，该命令将执行此存储过程。

CommandTimeout：

该属性设置等待命令执行的时间（以秒为单位）。如果不进行手动设置，则取该 Command 对象所在的 Connection 中连接字符串中设置的 CommandTimeOut 值。

CommandType：

该属性的值为 CommandType 值之一，默认为 text。当将 CommandType 设置为 StoredProcedure 时，应将 CommandText 属性设置为存储过程的名称。当调用 Execute 方法之一时，该命令将执行此存储过程。

CommandType 属性设置为 TableDirect 时，应将 CommandText 属性设置为要访问的表的名称。如果已命名的任何表包含任何特殊字符，那么用户可能需要使用转义符语法或包括限定字符。当您调用“执行”(Execute) 方法之一时，将返回命名表的所有行和列。

Connection：

该属性为与该 OscarCommand 对象关联的与数据源的连接。

4.2 执行命令

命令 OscarCommand 对象构造好之后，就要传递给后台执行，但是不同命令的执行方式是不同的，这是根据命令的目标决定的，以下就是几种不同的命令执行方式：

命令	返回值
ExecuteReader	返回一个 DataReader 对象。
ExecuteScalar	返回一个标量值。
ExecuteNonQuery	执行不返回任何行的命令。
ExecuteQuery	执行 SQL 语句，接收服务器返回的结果，以上几种命令均引用，该方法实体在连接模块中，但逻辑上属于命令模块，故在这里说明

4.2.1 ExecuteReader()

将解析后的 CommandText 发送到 Connection ，调用 ExecuteQuery 方法，得到结果，

使用含有结果的 command 对象以及 CommandBehavior 并生成一个神通数据库 DataReader。

例如：

```
OscarCommand command = new OscarCommand("select * from tablea where field_serial = 1;", TheConnection);
OscarDataReader dr = command.ExecuteReader();
dr.Read();
Char[] result = new Char[6];
dr.GetChars(1, 0, result, 0, 6);
```

4.2.2 ExecuteScalar()

将解析后的 CommandText 发送到 Connection ，调用 ExecuteQuery 方法，得到结果，并返回查询所得结果集中第一行的第一列。忽略其他列或行。这个方法看似有些突兀，但是主要针对于可能需要返回只是单个值的数据库信息，而不需要返回表或数据流形式的数据库信息，这在性能上是一个提升。

使用 ExecuteScalar 方法处理 SQL 数据查询语句：

```
OscarCommand command = new OscarCommand( "select count(*) from tablea where field_text is null", con);
int result = command.ExecuteScalar();
```

这里 result 得到第一行第一列的值。

4.2.3 ExecuteNonQuery()

将解析后的 CommandText 发送到 Connection ，调用 ExecuteQuery 方法，得到结果，返回受影响的行数，对于 UPDATE、INSERT 和 DELETE 语句，返回值为该命令所影响的行数。对于所有其他类型的语句，返回值为-1。如果发生回滚，返回值也为-1。

下面给了一个示例：

```
OscarCommand command = new OscarCommand("insert into tablea(field_text) values (:p0)", conn);
command.Parameters.Add(new OscarParameter("p0", OscarDbType.Text));
command.Parameters["p0"].Value = @"\test";
Object result = command.ExecuteNonQuery();
```

这里 result 得到的是一个 Int 型的值，值为 1。

此方法+2 次重载，也可以将要执行的 sql 语句作为 ExecuteNonQuery 的参数来执行。

下面的代码与上面的代码同效：

```
OscarCommand command = new OscarCommand();
Command.Connection=con;
Object result = command.ExecuteNonQuery("insert into tablea(field_text) values ('test')");
```

4.3Prepare()方法

Prepare() 过程其实是一个将用户输入的命令语句经过语法解析最终转化为服务器可以执行的语句的复杂过程。当用户将命令语句发送到数据提供程序，神通数据库 .net data provider 会构造一个 prepare 后的执行语句，在数据源上创建该命令的准备好已编译过的版本，可以重复使用，而后台仅编译了一次，减少后台编译的成本。

由于神通数据库.NET Data Provider 使用 Prepare 方式对数据库进行数据存取访问，而实际用户在使用这个框架时，不能直接操纵 Prepare 过的语句，只需要执行命令即可。

下面给出一个完整的示例：

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.OscarClient;
using System.Data;
using System.Data.SqlClient;
namespace TestProviderForHelpfiles
{
    class Program
    {
        static void Main(string[] args)
        {
            PrepareTest();
        }
public static void PrepareTest()
        {
            string connectString = "Server=10.0.5.226;Port=2003;User Id=SYSDBA;Password=szoscar55;Database=OSRDB;Encoding=GBK;";
            OscarConnection TheConnection = new OscarConnection(connectString);
            TheConnection.Open();
            // 在数据源建立一个新的表
    try
    {
    OscarCommand create = new OscarCommand("create table PrepareTest(c varchar(20), b boolean);", TheConnection);
        create.ExecuteNonQuery();
        create.Dispose();
      }
      catch
      {
      }
         // 用 prepare 方法向表插入数据
        OscarCommand command = new OscarCommand(
                "insert   into PrepareTest(c, b) values(@c, @b);",
                    TheConnection);
         command.Parameters.Add("@c", OscarDbType.VarChar);
         command.Parameters.Add("@b", OscarDbType.VarBinary);
         //   prepare 语句到后台
         command.Prepare();
        for (int i = 0; i < 5; i++)
        {
      if (i % 2 != 0)
        {    command.Parameters[1].Value = true;       }
        else
        {    command.Parameters[1].Value = null;       }
        command.Parameters[0].Value = i.ToString();
        // 执行编译好的 sql 语句，参数更改
        command.ExecuteNonQuery();
      }
        command.Dispose();
        OscarDataReader reader = null;
        OscarCommand select = null;
    try
    {
                    // Check that data is correct
select = new OscarCommand("( select * from PrepareTest)", TheConnection);
```

```
                reader = select.ExecuteReader();
                while (reader.Read())
              {
                  if (!reader.IsDBNull(0))
                   {    Console.Write(reader.GetString(0).ToString() + "\t");    }
                    else
                   {    Console.Write("null\t");      }
                    if (!reader.IsDBNull(1))
                   {    Console.WriteLine(reader.GetString(1).ToString());    }
                   else
                   {    Console.Write("null\t");      }
            }
        }
             catch (Exception ex)
            {    throw ex;          }
             finally
             {
                if (reader != null)
                { reader.Close();    }
                if (select != null)
                { select.Dispose();    }
                  // Drop      table
                  OscarCommand drop = new OscarCommand("drop table PrepareTest",
TheConnection);
                   drop.ExecuteNonQuery();
                   drop.Dispose();
                }
             }
         }
    }
```

4.4命令参数绑定配置

首先介绍下参数的命名方式，我们在 Sql 语句中写入参数名，也可以用“?”来缺省参数名，有参数名的参数为‘具名参数’，驱动会根据 OscarCommand 对象的参数名与 sql 语句的参数名进行匹配，在 sql 语句中，以“:”（冒号）或者“@”符号开始的字段被驱动程序视为参数名。参数名的命名规则包含数字、字母或者下划线并且以字母或者下划线为首个字符的字段。

命令对象可使用参数来将值传递给 SQL 语句或存储过程。与命令文本不同，参数输入被视为文本值，而不是可执行代码。这样可帮助抵御“SQL 注入”攻击，这种攻击的攻击者会将命令插入 SQL 语句，从而危及服务器的安全。

除具备安全优势外，这里使用一种参数化的命令设计，参数绑定 OscarCommand 对象的 Parameters 属性。Parameters 属性为 OscarParameterCollection 对象，OscarParameterCollection 对象的 ADD()方法来添加参数，ADD()方法有六个重载方法，可选择的提供参数名，参数类型，参数长度等等，也可以部分缺省。这个参数可以存在于 sql 语句中也可以是存储过程中，ADD()方法最终要在 OscarParameterCollection 对象中添加一个 OscarParameter 对象。

```
    OscarParameter param= new OscarParameter("Param_Name",Param_Type);
    param="The Value";
    OscarCommand cmd = new OscarCommand("insert into tablea(field_text) values
(:Param_Name) "，conn);
    cmd.Parameters.Add(param);
```

上面的代码片段就是一个带有参数的命令 OscarCommand 的示例，OscarParameter 构造方法第一个参数即为参数的名，此名称应与 OscarCommand 的 Sql 语句中参数名相符。在有

in 参数的存储过程的操作中，添加参数，在命令中可以不写出参数，也就是说，直接写出存储过程名即可，记得设定 CommandType=CommandType.StoredProcedure；

例如，一个用户自定义存储过程”FunctionCaseSensitive”，该存储过程有两个 in 参数，没有 out 参数。

```
OscarCommand command = new OscarCommand("\"SYSDBA\".\"FunctionCaseSensitive\"", con);
command.CommandType = CommandType.StoredProcedure;
// 添加新的参数
command.Parameters.Add(new OscarParameter("p1", OscarDbType.Integer));
// 为新的参数赋值
command.Parameters[0].Value = 1;
// 再次添加参数
command.Parameters.Add(new OscarParameter("p2", OscarDbType.Text));
// 为新添加的参数赋值
command.Parameters[1].Value = 2;
// 执行
Object result = command.ExecuteScalar();

Assert.AreEqual(0, result);
```

这个例子里 FunctionCaseSensitive 要加引号是因为在后台区分大小写，当执行后，驱动会将拼接好的调用存储过程语句发送到后台。

关于存储过程，在下面章节中详细介绍。

第5章 批量导入数据

神通数据库 .Net data provider 驱动提供了 OscarImportCommand 对象和 OscarImportHandler 对象，用于提供批量导入数据协议使用。

OscarImportCommand 对象执行的是旧的 import 协议，OscarImportHandler 对象执行的是新的 insert bulk 协议。在性能与健壮性方面，insert bulk 协议比旧的 import 协议有了很大的提高，并且执行 insert bulk 协议导入的数据可以自动维护索引，检查约束等。在此，建议用户优先使用 OscarImportHandler 对象来进行大容量数据的导入。

5.1 OscarImportCommand

OscarImportCommand 对象提供了快速的数据导入功能。可以将数据以文件流的形式导入到数据库中。该对象执行的是 import 协议，不同于 OscarCommand 对象执行的 sql 语句，不同于 INSERT 语句。在执行批量导入时，驱动程序并不负责检查数据的完整性。所以对于异常的容错能力比较有限。

OscarImportCommand 对象提供了三个重载的 ExecuteImport 方法以执行导入数据的操作。返回结果为导入数据的行数。三个执行方法分别执行的是内存中构造的数据，字符串格式的数据，流（通常是文件流）形式的数据。

构造函数中可以通过给定表名，模式名，或者 TABLEOID 进行指定数据表来操作。

5.1.1 构造数据

导入数据的构造方式：

序号	数据构造方式	数据缓存形式	数据执行方法	调用函数
1	依据数据类型按列添加数据	缓存到内存（或外存临时文件）中	ExecuteImport()	SetXXX(int,object) EndRow（）；
2	以数组的形式添加行数据	缓存到内存（或外存临时文件）中	ExecuteImport()	AddRow(object[])
3	按照协议格式化好的字符串添加表数据	缓存到内存中	ExecuteImport(String)	无
4	以协议格式的数据流的方式添加表数据	不缓存	ExecuteImport(Stream)	无

表 5.1 构造数据方式

5.1.2 公开属性

BufferSize,可设置属性，内存中缓存的大小值。

Connection，只读属性，该命令所用的连接对象。

5.1.3 执行命令

执行命令有三个重载方法，针对于不同的类型缓存数据或者用户提供数据，将数据导入到数据库中

执行方法 1

名称、标识符	ExecuteImport(String data)
参数描述	data: 要插入的数据行列表
功能描述	data 为用户按照协议构造好的数据集的字符串，直接通过 import 协议发送给后台
返回值	Int，影响行数

执行方法 2

名称、标识符	ExecuteImport(Stream stream)
参数描述	stream: 外存中按照协议写好的流文件
功能描述	用户按照协议构造好的数据集的字符流，直接通过 import 协议发送给后台
返回值	Int，影响行数

执行方法 3

名称、标识符	ExecuteImport()
参数描述	
功能描述	将缓存的协议数据集，通过 import 协议发送给后台
返回值	Int，影响行数

对于构造数据方式 1 和 2，采用执行方法 3 来执行。构造数据方式 3，采用执行方法 1 执行。构造数据方式 4，采用执行方法 2 执行。

5.1.4 示例

示例一：

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.OscarClient;
using System.Data;
using System.Data.SqlClient;
namespace TestProviderForHelpfiles
{
    class Program
    {
        static void Main(string[] args)
        {
           testImport1();
        }
public static void testImport1()
        {
            string connectString = "Server=10.0.5.226;Port=2003;User Id=SYSDBA;Password=szoscar55;Database=OSRDB;Encoding=GBK;";
            OscarConnection TheConnection = new OscarConnection(connectString);
            TheConnection.Open();
            //构造
            OscarImportCommand commd = TheConnection.CreateImportCommand("tablea");
            //    设置一行数据
            commd.SetInt(0, 128);
            commd.SetString(1, "xiaochun");
            //commd.SetInt(2, 9);
            //commd.SetInt(3, 512);
            //commd.SetBool(4, true);
            commd.EndRow();//设置结束
```

```
                    //     设置另一行
                    commd.SetInt(0, 129);
                    commd.SetString(1, "xiaochun1");
                   // commd.SetInt(2, 9);
                   //commd.SetInt(3, 512);
                   // commd.SetBool(4, true);
                    commd.EndRow();
                    commd.ExecuteImport(); //  执行操作
                    commd.Dispose();
                    TheConnection.Close();
                }
            }
        }
```

示例二：

```
    using System;
    using System.Collections.Generic;
    using System.Text;
    using System.Data.OscarClient;
    using System.Data;
    using System.Data.SqlClient;
    namespace TestProviderForHelpfiles
    {
        class Program
        {
            static void Main(string[] args)
            {
               testImport2();
            }
    public static void testImport2()
            {
                string connectString = "Server=10.0.5.226;Port=2003;User
Id=SYSDBA;Password=szoscar55;Database=OSRDB;Encoding=GBK;";
                OscarConnection TheConnection = new OscarConnection(connectString);
                TheConnection.Open();
                OscarImportCommand commd =
TheConnection.CreateImportCommand("tablee");
                System.IO.FileStream fstream = new System.IO.FileStream(@"D:\6.jpg",
System.IO.FileMode.Open);
                // 添加一行数据，其中一列为大对象
                commd.AddRow(2, fstream);
                commd.ExecuteImport();
                fstream.Close();
            }
        }
    }
```

示例三：

```
    using System;
    using System.Collections.Generic;
    using System.Text;
    using System.Data.OscarClient;
    using System.Data;
    using System.Data.SqlClient;
    namespace TestProviderForHelpfiles
    {
        class Program
```

```
        {
            static void Main(string[] args)
            {
               testImport3();
            }
    public static void testImport3()
            {
                string connectString = "Server=10.0.5.226;Port=2003;User Id=SYSDBA;Password=szoscar55;Database=OSRDB;Encoding=GBK;";
                OscarConnection TheConnection = new OscarConnection(connectString);
                TheConnection.Open();
               OscarImportCommand commd = TheConnection.CreateImportCommand("tablea");
                string data = "127,'abcd'\n";
                commd.ExecuteImport(data);
                commd.Dispose();
                TheConnection.Close();
     }
        }
    }
```

示例四：

```
    using System;
    using System.Collections.Generic;
    using System.Text;
    using System.Data.OscarClient;
    using System.Data;
    using System.Data.SqlClient;

    namespace TestProviderForHelpfiles
    {
        class Program
        {
            static void Main(string[] args)
            {
                testImport4();
            }
    public static void testImport4()
            {
                string connectString = "Server=10.0.5.226;Port=2003;User Id=SYSDBA;Password=szoscar55;Database=OSRDB;Encoding=GBK;";
                OscarConnection TheConnection = new OscarConnection(connectString);
                TheConnection.Open();
                OscarImportCommand commd = TheConnection.CreateImportCommand("test3");
                // 该文件为数据文件，为用户构造好的导入文件
                System.IO.FileStream fstream = new System.IO.FileStream("../../../testsuite/testFile/aa.txt", System.IO.FileMode.Open);
                try
                {
                    commd.ExecuteImport(fstream);
                }
                catch
                {
                    throw;
                }
                finally
```

```
                {
                    fstream.Close();
                }
            }
    }
    }
```

5.2 OscarImportHandler

OscarImportHandler 对象提供了快速的大容量数据导入功能。不同于 OscarImportCommand 对象，该对象执行的是 insert bulk 协议，可以将数据以二进制流的形式导入到神通数据库中。在性能与健壮性方面，insert bulk 协议比旧的 import 协议有了很大的提高，并且执行 insert bulk 协议导入的数据可以自动维护索引，检查约束等。建议用户优先使用 ImportHandler 对象来进行大容量数据的导入。

OscarImportHandler 对象提供了三个重载的 ExecuteImport 方法以执行导入数据的操作。返回结果为导入数据的行数。

构造函数中可以通过给定表名，模式名，或者 TABLEOID 进行指定数据表来操作。

5.2.1 构造数据

导入数据的构造方式：

序号	数据构造方式	数据缓存形式	数据执行方法	调用函数
1	依据数据类型按列添加数据	缓存到内存（或外存临时文件）中	ExecuteImport()	SetXXX(int,object) EndRow（）；
2	以协议格式的数据流的方式添加表数据	缓存到内存中	ExecuteImport(String，String，String)	无
3	以协议格式的数据流的方式添加表数据	缓存到内存中	ExecuteImport(String，String，String,Char)	无

构造数据方式表 5.2

5.2.2 公开属性

IsByte,可设置属性，设置是否通过字节方式导入数据。

Connection，只读属性，该命令所用的连接对象。

5.2.3 执行命令

执行命令有三个重载方法，针对于不同的类型缓存数据或者用户提供数据，将数据导入到数据库中

执行方法 1

名称、标识符	ExecuteImport()
参数描述	
功能描述	将缓存的协议数据集，通过 insert bulk 协议发送给后台
返回值	Int，影响行数

执行方法 2

名称、标识符	ExecuteImport(string filePath, string columnSep, string rowSep)

参数描述	filePath:数据文件的路径,columnSep：列间隔符,rowSep：行间隔符
功能描述	用户按照协议构造好的数据集的二进制流，通过 insert bulk 协议发送给后台
返回值	Int，影响行数

执行方法 3

名称、标识符	ExecuteImport(string filePath, string columnSep, string rowSep, char esc)
参数描述	filePath:数据文件的路径,columnSep：列间隔符,rowSep：行间隔符, esc:转义符
功能描述	用户按照协议构造好的数据集的二进制流，通过 insert bulk 协议发送给后台
返回值	Int，影响行数

对于构造数据方式 5，采用执行方法 1 来执行。构造数据方式 6，采用执行方法 2 执行。构造数据方式 7，采用执行方法 3 执行。

5.2.4 示例

示例一：

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.OscarClient;
using System.Data;
using System.Data.SqlClient;

namespace TestProviderForHelpfiles
{
    class Program
    {
        static void Main(string[] args)
        {
                String connectionSql = "Server=10.0.5.226;Port=2003;User Id=SYSDBA;Password=szoscar55;Database=OSRDB;Encoding=GBK;";
                OscarConnection con = new OscarConnection(connectionSql);
                con.Open();
                 OscarImportHandler icmd_handler = new OscarImportHandler(con, "test");
    System.IO.FileStream fstream = new System.IO.FileStream("../../../myPhoto.jpg", System.IO.FileMode.Open);

                    icmd_handler.SetBigInt(0, 132132234455666666);
                    icmd_handler.SetLob(1, fstream);
                    icmd_handler.SetString(2, "北京市海淀区永丰路");
                     icmd_handler.SetSmallInt(3,1);
                     icmd_handler.EndRow();

                     icmd_handler.SetBigInt(0, 132132233336116666);
                     icmd_handler.SetLob(1, fstream);
                     icmd_handler.SetString(2, "北京市昌平区北环路");
                     icmd_handler.SetSmallInt(3, 11);
                     icmd_handler.EndRow();
                     icmd_handler.ExecuteImport();
    }
        }
}
```

示例二：

```
using System;
using System.Collections.Generic;
```

```
using System.Text;
using System.Data.OscarClient;
using System.Data;
using System.Data.SqlClient;

namespace TestProviderForHelpfiles
{
    class Program
    {
        static void Main(string[] args)
        {
            String connectionSql = "Server=10.0.5.226;Port=2003;User Id=SYSDBA;Password=szoscar55;Database=OSRDB;Encoding=GBK;";
            OscarConnection con = new OscarConnection(connectionSql);
            con.Open();
            OscarImportHandler icmd_handler = new OscarImportHandler(con, "test");
System.IO.FileStream fstream = new System.IO.FileStream("../../../myPhoto.jpg", System.IO.FileMode.Open);
try
            {
icmd_handler.ExecuteImport("../../../test_first.txt", ",", "\r\n");
}
            catch (OscarException ex)
            {
                throw ex;
            }
            finally
            {
                con.Close();
            }
}
        }
}
```

示例三：

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.OscarClient;
using System.Data;
using System.Data.SqlClient;

namespace TestProviderForHelpfiles
{
            class Program
    {
        static void Main(string[] args)
        {
            String connectionSql = "Server=10.0.5.226;Port=2003;User Id=SYSDBA;Password=szoscar55;Database=OSRDB;Encoding=GBK;";
            OscarConnection con = new OscarConnection(connectionSql);
            con.Open();
            OscarImportHandler icmd_handler = new OscarImportHandler(con, "test");
System.IO.FileStream fstream = new System.IO.FileStream("../../../myPhoto.jpg", System.IO.FileMode.Open);
try
            {
```

```
icmd_handler.ExecuteImport("../../../test_first.txt", ",", "\r\n",'/');
}
                catch (OscarException ex)
                {
                    throw ex;
                }
                finally
                {
                    con.Close();
                }
}
            }
}
```

第6章 执行存储过程

存储过程是由流控制和 SQL 语句书写的过程，这个过程经编译和优化后存储在数据库服务器中，应用程序使用时只要调用即可；相对于 SQL 语句，它为用户提供了对数据更为方便的检索和更新方式。存储过程一般都是多条语句一起执行，所以会减少网络的开销。

存储过程和其他 SQL 语句的使用类似，只需要在执行命令 OscarCommand 之前，设置命令类型 CommandType 为 CommandType.StoredProcedure。这是必须的。这样驱动程序才会将这个 SQL 语句作为存储过程名来处理。

在 PROVIDER 中调用已储存过程的语法如下所示。

{过程名[(@ParametersName1, @ParametersName2, ...)]}

不带参数的储存过程的语法类似：

{过程名()}

这里无论带不带参数的存储过程的命令调用时，过程名后的圆括号不是必须的，例如：有一个名为 funcC 的存储过程名，带有一个 inputoutput 参数，并且返回值为参数值减 3。我们在创建命令对象 OscarCommand 对象时，可以采用如下的方法创建：

```
OscarCommand command = new OscarCommand("funcC", TheConnection);
OscarCommand command = new OscarCommand("funcC(@a)", TheConnection);
OscarCommand command = new OscarCommand("funcC(:a)", TheConnection);
```

以上三种方式创建的存储过程命令都是合法的，其中，第一种没有括号，也没有给出参数名，但是需要在后面的代码中，添加一个参数给命令对象，否则会抛出异常。如下：

```
 OscarParameter p = new 神通数据库 Parameter("I", OscarDbType.Integer);
command.parameters.Add(p);
```

用户创建好 OscarCommand 对象的时候,驱动并不知道该命令是存储过程的命令。所以在命令执行前，必须给定命令类型，如下：

```
command.CommandType = CommandType.StoredProcedure;
```

驱动命令 OscarCommand 对象的参数 OscarParameter 对象的默认 ParameterDirection 的值为 Input，所以如果应用要得到 Output 或者其他类型的 ParameterDirection 属性值时，应该显示的指定，如下

```
 OscarParameter p = new OscarParameter("I", OscarDbType.Integer);
 p.Direction = ParameterDirection.InputOutput;
```

下面给出了一个通过 PROVIDER 驱动程序在数据库中调用存储过程的完整方法。

```
 public void PreparedFunctionCallWithImplicitParameters()
 {
    OscarCommand command = new OscarCommand("funcC", TheConnection);
    command.CommandType = CommandType.StoredProcedure;
OscarParameter p = new OscarParameter("I", OscarDbType.Integer);
    p.Direction = ParameterDirection.InputOutput;
    p.Value = 4;

      command.Parameters.Add(p);

      command.Prepare();
      int result = (Int32) command.ExecuteScalar();
}
```

这个存储过程执行的结果为 1。

6.1存储过程的参数类型配置

存储过程的参数的 Direction 属性为枚举值 ParameterDirection，ParameterDirection 有四个值，分别为 Input，Output，InputOutput，ReturnValue。

Input 类型的参数：参数值通过命令对象，传递给存储过程，驱动不会更改该参数的值。

Output 类型的参数：参数的值可以为 Null,也可以是任何其他类型的值，驱动不会将该类型的值给存储过程，而是将相同位置的参数在存储过程中获得值，返回给用户。

InputOutput 类型的参数：该类型的参数的值传递给存储过程，并且接收改变后的值。

ReturnValue 类型的参数：参数获得存储过程执行后得到的值。

6.1.1 Input 参数

Input 参数在将值传递给后台后遍失去了作用。

下面是一个示例，该示例与上面的示例是一样的，只不过参数类型为 Input：

```
public void PreparedFunctionCallWithImplicitParameters()
  {
     OscarCommand command = new OscarCommand("funcC", TheConnection);
     command.CommandType = CommandType.StoredProcedure;
OscarParameter p = new OscarParameter("I", OscarDbType.Integer);
     p.Direction = ParameterDirection.Input;
     p.Value = 4;

       command.Parameters.Add(p);

       command.Prepare();
       int result = (Int32) command.ExecuteScalar();
       Console.WriteLine("The param value is " +    p.Value);
}
```

在执行之后参数 p 的值仍然是 4，result 值为 1。最后输出打印为：“The param value is 4”也就是说参数的值不变。

6.1.2 InputOutput 参数

既支持输入又接受输出的参数（InputOutput 参数）。

这里我们仍沿用上面的示例来说明，参数类型为 InputOutput：

```
public void PreparedFunctionCallWithImplicitParameters()
  {
     OscarCommand command = new OscarCommand("funcC", TheConnection);
     command.CommandType = CommandType.StoredProcedure;
OscarParameter p = new OscarParameter("I", OscarDbType.Integer);
     p.Direction = ParameterDirection.InputOutput;
     p.Value = 4;

       command.Parameters.Add(p);

       command.Prepare();
       int result = (Int32) command.ExecuteScalar();
  Console.WriteLine("The param value is " +    p.Value);
}
```

在执行之后参数 p 的值发生改变，是 1，result 值为 1。最后输出打印为：“The param value

is 1”。这里要说明的是，如果没有 ReturnValue 类型的参数，那么存储过程的返回值可以视为 Output 类型的参数。这里，存储过程的返回值会被付给 Output 类型或者 ReturnValue 类型的参数。

6.1.3 Output 参数

Output 类型的参数的值会在命令执行时忽略掉，也就是说该类型的参数赋值是无用的。该类型的参数只被用来返回 out 类型的参数。下面的示例中，存储过程名为“testreturnrecord”，存储过程有两个 out 类型的参数，第一个参数的默认值为 4，第二个默认值为 5。执行后，参数应该被存储过程赋值。我们不必为 Output 参数赋值。

```
 public void ProcedureOutputSupport()
{
    OscarCommand command = new OscarCommand("SYSDBA.testreturnrecord()", TheConnection);
    command.CommandType = CommandType.StoredProcedure;
    command.Parameters.Add(new OscarParameter(":a", OscarDbType.Integer));
    command.Parameters[0].Direction = ParameterDirection.Output;
    command.Parameters.Add(new OscarParameter(":b", OscarDbType.Integer));
    command.Parameters[1].Direction = ParameterDirection.Output;
    command.ExecuteNonQuery();
    //第一个参数的值为 4
   // Assert.AreEqual(4, command.Parameters[0].Value);
Console.WriteLine(command.Parameters[0].Value);
    //第二个参数的值为 5
  //   Assert.AreEqual(5, command.Parameters[1].Value);
Console.WriteLine(command.Parameters[1].Value);
 }
```

6.1.4 ReturnValue 参数

该类型的参数比较好理解，它只用来得到存储过程返回值，类似于得到函数执行后的返回值一样。如下示例，存储过程名为“funcd”，存储过程的定义中没有参数，但是有返回值，返回值为 6。如下示例：

```
public void FunctionCallReturnValueParameter()
{
    OscarCommand command = new OscarCommand("funcd", TheConnection);
    command.CommandType = CommandType.StoredProcedure;

    OscarParameter p = new OscarParameter("@a", DbType.Int32);
    p.Direction = ParameterDirection.ReturnValue;
    p.Value = -1;
    //  该参数的值为-1
    command.Parameters.Add(p);

    command.ExecuteNonQuery();
    //  该参数的值为 6
    //Assert.AreEqual(6, command.Parameters["@a"].Value);
Console.WriteLine(command.Parameters["@a"].Value);
}
```

该参数在执行之后值为 6。

第7章 处理结果集

本章介绍了以下结果集对象的创建执行以及用法。

1. OscarDataReader 对象
2. OscarDataAdapter 对象
3. OscarCommandBuilder 对象
4. DataSet 对象

7.1 OscarDataReader 对象

OscarDataReader 对象提供一种获取数据存储区中数据的高性能方法。它提供一个只进只读的服务器端游标。这使得 OscarDataReader 对象成为填充 ListBox 对象和 DropDownList 对象的理想选择。在编写一个修改数据的操作，且该操作不必将修改结果返回给数据库时，最好不要使用 OscarDataReader 对象。对于修改数据的操作，最好使用 OscarDataAdapter 对象，我们将在下一节讨论该对象。

OscarDataReader 对象包含一个将数据读入其缓冲区的 Read 方法，该方法一次只能读取一行数据，这意味着在处理数据之前，不必将所有的数据都读入应用程序中。下面的代码直接使用 tablea 表中的数据填充一个新的 DataTable 对象。

```
public void ExampleA()
{
  OscarCommand command = new OscarCommand("select * from tablea;", TheConnection);
  command.Prepare();
  OscarDataReader dr = command.ExecuteReader(CommandBehavior.Default);
  DataTable table = new DataTable();
  table.Load(dr,LoadOption.Upsert);
  dr.Close();
}
```

注意，DataTable 对象的 Load 方法包含一个 LoadOption 参数。LoadOption 提供了确定哪个 DataRowVersion 应获取输入数据的选项。例如，如果加载一个 DataTalbe 对象，并修改数据，然后将修改结果保存到数据库中，这时如果有人在你获取数据后和试图保存数据之前修改了数据，则可能出现同步错误。一种解决方法是再次加载 DataTable 对象，并使用默认的 PreserveCurrentValues 枚举值加载原先保存数据库数据的 DataRowVersion，而使当前的 DataRowVersion 保持不变。接着，只需再次执行 Update 方法，就能成功地执行更新操作了。

关于更多 DataTable 对象，请参考 MSDN。

使用 OscarDataReader 对象是获取数据库中数据的最快方法之一，但其中一个问题是在遍历查询结果时，

提示：到底使用 OscarDataReader 还是使用 DataSet。这个题的答案取决于性能要求。如果需要高性能，并且对获取的数据只访问一次，则使用 OscarDataReader；如果要多次访问同一个数据，或者要在内存中模拟一种复杂的关系，则使用 DataSet。通常，在确定使用哪一个之前，最好全面测试每一种方法。

OscarDataReader 对象的 Read 方法可以获取查询结果中的某一行。把列名或者列的序号易用传递给 OscarDataReader,就可以返回行中的第一列进行访问。为了使性能最优，OscarDataReader 提供了一系列的方法(GetDateTime,GetDouble,GetInt32 等)，以访问基本数据类型的列值。在知道基本数据类型的情况下，使用这些类型访问器方法可以建设在获取列值时所需的大量类型转换（从 Object 类型转换过来）。

OscarDataReader 是一个无缓冲的数据流，该 允许过程逻辑有效地按顺序处理来自数据源的结果。在获取大量的数据时，最好选择 OscarDataReader。在使用完 OscarDataReader

对象时，必须调用 Close 方法并关闭 OscarDataReader 对象的数据库连接；否则连接只有在垃圾回收器收集该对象时才关闭。

在调用 OscarCommand 的 ExecuteReader 方法时，我们可以使用 CommandBehavior 的枚举值作为参数。例如 CommandBehavior.CloseConnection 枚举值作为参数，在调用 OscarDataReader 的 Close 方法时要自动关闭数据库连接。

7.2 OscarDataAdapter 对象

和.Net Framwork 中的其他数据提供程序一样，神通数据库 也提供一个 DataAdapter 对象：OscarDataAdapter 对象。OscarDataAdapter 用于获取数据源中的数据并填充 DataSet 中的 DataTable 对象和约束，还可以将 DataSet 产生的改变解析回数据源。它使用.Net data provider 的 OscarConnection 对象连接数据源，使用 OscarCommand 对象获取 DataSet 对象中的数据并将 DataSet 对象中数据的变化返回给数据源。DataSet 与 OscarDataReader 的区别在于 OscarDataReader 对象使用 OscarConnection 直接访问数据，不必使用 OscarDataAdapter。OscarDataAdapter 实际上将 DataSet 对象与实际数据源断开。

OscarDataAdapter 的 SelectCommand 属性是一个从数据源中获取数据的 Command 对象。神通数据库 DataAdapter 的 InsertCommand、UpdateCommand 和 DeleteCommand 属性也是 Command 对象，他们根据对 DataSet 中数据的修改来更新数据源中的数据。OscarDataAdapter 的 Fill 方法用于将 OscarDataAdapter 中的 SelectCommand 的结果填充到 DataSet，它也用于添加或者刷新 DataSet 中的行以匹配数据源中的行。OscarDataAdapter 的 FillSchema 方法用于将 OscarDataAdapter 中的 SelectCommand 的结果中的模式信息填充 DataSet。下面的示例说明如何填充 DataSet 对象：

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.OscarClient;
using System.Data;
using System.Data.SqlClient;
namespace TestProviderForHelpfiles
{
    class Program
    {
        static void Main(string[] args)
        {
            String connectionSql = "Server=10.0.5.226;Port=2003;User Id=SYSDBA;Password=szoscar55;Database=OSRDB;Encoding=GBK;";
            OscarConnection TheConnection = new OscarConnection(connectionSql);
            TheConnection.Open();
             // TheConnection 是连接到 Oscar 数据源的 OscarConnection 连接对象
            // 建立一个新的 OscarDataAdapter 对象。
            OscarDataAdapter da = new OscarDataAdapter("select * from tablea;", TheConnection);
            // 建立一个 DataSet 对象。以获得缓存的数据。
            DataSet ds = new DataSet();
            // 填充 DataSet 对象
            da.Fill(ds);
            // 得到 DataSet 对象的第一个 Table（DataTable 对象）
            // 遍历该 Table 的每一个 DataRow 对象，并输出第一列和第二列内容
            foreach (DataRow row in ds.Tables[0].Rows)
            {
                Console.WriteLine(row[0] + " " + row[1]);
            }
```

```
                    // 释放 OscarDataAdapter 对象占用资源
                    da.Dispose();
                    // 关闭连接，并释放该连接所占用资源
                    TheConnection.Close();
    }
            }
    }
```

注意这里是如何使用 OscarDataAdapter 的构造函数来传递和设置 SelectCommand，又是如何传递连接字符串以代替一个包含已初始化 Connection 对象。我们调用了 OscarDataAdapter 对象的 Fill 方法，并传递一个已经初始化的 DataSet 对象。如果 DataSet 对象没有经过初始化，则 Fill 方法将会引发异常（System.ArgumentNullException）。

现在我们通过一个高级点的示例来说明如何使用 OscarDataAdapter 对 DataTable 进行插入、更新和删除操作，并把改变返回到数据源。

这个示例是 ADO.NET 1.0 的做法，我们需要手动设置 UpdateCommand 等属性。

```
    using System;
    using System.Collections.Generic;
    using System.Text;
    using System.Data.OscarClient;
    using System.Data;
    using System.Data.SqlClient;
    namespace TestProviderForHelpfiles
    {
        class Program
        {
            static void Main(string[] args)
            {
                string connectString = "Server=10.0.5.226;Port=2003;User Id=SYSDBA;Password=szoscar55;Database=OSRDB;Encoding=GBK;";
                OscarConnection TheConnection = new OscarConnection(connectString);
                TheConnection.Open();
                OscarCommand command = new OscarCommand("insert into tableb(field_int2) values (2)", TheConnection);
                command.ExecuteNonQuery();
                DataSet ds = new DataSet();
                OscarDataAdapter da = new OscarDataAdapter("select * from tableb where field_serial = (select max(field_serial) from tableb)", TheConnection);
                da.InsertCommand = new OscarCommand(";", TheConnection);
                da.UpdateCommand = new OscarCommand("update tableb set field_int2 = :a, field_timestamp = :b, field_numeric = :c where field_serial = :d", TheConnection);
                da.UpdateCommand.Parameters.Add(new OscarParameter("a", DbType.Int16));
                da.UpdateCommand.Parameters.Add(new OscarParameter("b", DbType.DateTime));
                da.UpdateCommand.Parameters.Add(new OscarParameter("c", DbType.Decimal));
                da.UpdateCommand.Parameters.Add(new OscarParameter("d", OscarDbType.BigInt));
                da.UpdateCommand.Parameters[0].Direction = ParameterDirection.Input;
                da.UpdateCommand.Parameters[1].Direction = ParameterDirection.Input;
                da.UpdateCommand.Parameters[2].Direction = ParameterDirection.Input;
                da.UpdateCommand.Parameters[3].Direction = ParameterDirection.Input;
                da.UpdateCommand.Parameters[0].SourceColumn = "field_int2";
                da.UpdateCommand.Parameters[1].SourceColumn = "field_timestamp";
                da.UpdateCommand.Parameters[2].SourceColumn = "field_numeric";
```

```
                    da.UpdateCommand.Parameters[3].SourceColumn = "field_serial";

                    da.Fill(ds);
                    DataTable dt = ds.Tables[0];
                    DataRow dr = ds.Tables[0].Rows[ds.Tables[0].Rows.Count - 1];
                    dr["field_int2"] = 4;
                    DataSet ds2 = ds.GetChanges();
                    da.Update(ds2);
                    ds.Merge(ds2);
                    ds.AcceptChanges();
                    OscarDataReader dr2 = new OscarCommand("select * from tableb where
field_serial = (select max(field_serial) from tableb)", TheConnection).ExecuteReader();
                    dr2.Read();
                    dr2.Close();
    }
            }
    }
```

tableb 的结构如下，FIELD_SERIAL（int），FIELD_INT2（smallint）FIELD_TIMESTAMP（timestamp），FIELD_NUMERIC（decimal）。这个用例里没有用 ADO.NET 2.0 的新特性神通数据库 CommandBuilder。下一节介绍该对象的使用方法。

7.3 OscarCommandBuilder 对象

OscarCommandBuilder 是 ADO.NET2.0 新增的类新增的概念。ADO.NET2.0 改变了 DbDataAdapter 更新数据库的方式，从而提升了性能。批处理更新支持被加入到 DbDataAdapter 中。这就意味着在调用 Update 方法时，DbDataAdapter 把 DataSet 中的所有更新一次全部返回到数据库中。

每当设置了 DataAdapter 属性，OscarCommandBuilder 就将其本身注册为 RowUpdating 事件的侦听器。一次只能将一个 OscarDataAdapter 与一个 OscarCommandBuilder 对象互相联系。

为了生成 Insert,Update 或 Delete 语句，OscarCommandBuilder 会自动使用 SelectCommand 属性来检索所需的元数据集。如果在检索元数据后（例如在第一次更新后）更改 SelectCommand，则应调用 RefreshSchema 方法来更新元数据。

SelectCommand 还必须至少返回一个主键列或唯一的列。如果什么都没有返回，就会产生 InvalidOperationException 异常，不生成命令。

OscarCommandBuilder 还使用由 SelectCommand 引用的 OscarConnection、CommandTimeout 和 OscarTransaction 属性。如果修改了任何这些属性或者替换了 SelectCommand 本身，用户则应调用 RefreshSchema。否则，InsertCommand、UpdateCommand 和 DeleteCommand 属性都保留它们以前的值。

如果调用 Dispose，则会解除 OscarCommandBuilder 与 OscarDataAdapter 的关联，并且不再使用生成的命令。(更多请参考 MSDN)

如果，上一个示例采用 OscarCommandBuilder 对象来操作，则不必手动注册 UpdateCommand。

如下的示例介绍该特性的用法：

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.OscarClient;
using System.Data;
using System.Data.SqlClient;
```

```
namespace TestProviderForHelpfiles
{
    class Program
    {
        static void Main(string[] args)
        {
            string connectString = "Server=10.0.5.226;Port=2003;User Id=SYSDBA;Password=szoscar55;Database=OSRDB;Encoding=GBK;";
            OscarConnection TheConnection = new OscarConnection(connectString);
            TheConnection.Open();

            DataSet ds = new DataSet();
            OscarDataAdapter da = new OscarDataAdapter("select * from tableb where field_serial = (select max(field_serial) from tableb)", TheConnection);
            OscarCommandBuilder cb = new OscarCommandBuilder(da);
            da.Fill(ds);
            DataTable dt = ds.Tables[0];
            DataRow dr = ds.Tables[0].Rows[ds.Tables[0].Rows.Count - 1];
            dr["field_int2"] = 4;
            DataSet ds2 = ds.GetChanges();
            da.Update(ds2);
            ds.Merge(ds2);
            ds.AcceptChanges();
            OscarDataReader dr2 = new OscarCommand("select * from tableb where field_serial = (select max(field_serial) from tableb)", TheConnection).ExecuteReader();
            dr2.Read();
            Console.WriteLine(dr2["field_int2"].ToString());
            dr2.Close();
    }
        }
    }
```

7.4DataSet 对象

这一节中详细介绍 DataSet 对象。在 ADO.NET 中，DataSet 对象对于需要支持断开连接的数据和分布式数据的应用程序非常重要。DataSet 是一个内存驻留的数据表示方式，它提供一个与数据源无关的一致性关系编程模型。DataSet 表示一个完整的数据集，该数据集包含相关的表、约束以及表之间的关系，就像一个驻留在内存中的关系数据库。

DataSet 把大量的元数据存放在内存中，所以在使用它之前必须清楚具体需要填充多少数据。

DataSet 中的方法和对象与关系数据库模型中的对象和方法是一致的。DataSet 也可以用 XML 格式保存和重新加载它的内容，并把其架构表现为 XSD。DataSet 与数据库连接是完全断开的，使用内存中的什么数据对它进行填充完全取决于用户。

7.4.1 DataTableCollection 对象

DataSet 包含一个集合，该集合中可以没有表，也可以有由 DataTable 对象表示的多个表。DataTableCollection 包含 DataSet 中所有的 DataTable 对象。

DataTable 在 System.Data 命名空间中定义，它代表一个由内存驻留数据组成的表。DataTable 包含一个由 DataColumnCollection 表示的列集合，定义了表的模式和行；包含一个由 DataRowCollection 所示的行集合，包含了表中的数据。除了当前的状态外，DataRow 还保留了其原始状态。

7.4.2 DataRelationCollection 对象

DataSet 在 DataRelationCollection 对象中存储其关系。关系将 DataTable 中的行和另一个 DataTable 中的行关联起来。在 DataSet 中的关系可以包含由 UniqueConstraint 和 ForeignKeyConstraint 对象表示的约束，这与关系数据库中主键列和外键列间的 Join 路径类似。

关系说明了用来连接两个表间信息的内容。DataRelation 的基本元素是关系名称、两个相关联的表和每一个表中的相关列。创建关系时允许使用同一个表中的多个列，但是需要一个 DataColumn 数组指定列。当把关系添加到 DataRelationCollection 时，可以选择性地添加 ForeignKeyConstraints，不允许发送使关系无效的改变。

7.4.3 ADO.NET DataTable 对象

ADO.NET 使用 DataTable 对象表示 DataSet 中的表。DataTable 是一个表示内存中关系数据的表。该数据对于驻留它的.NET 应用程序来说是本地的，但可以通过 OscarDataAdapter 使用数据源填充它。

DataTable 类是.NET Framework 类库中 System.Data 命名空间的一个成员。它可以单独创建和使用，也可以作为 DataSet 的一个成员。DataTable 对象可以被其他的.NET Framework 对象使用，包括 Data View 对象。通过 DataSet 对象的 Tables 属性可以访问 DataSet 中表的集合。

如果使用数据库填充 DataTable，DataTable 就会自动继承该数据库的约束，也就是说不必手动进行使用的工作。DataTable 必须包含一些行，以存储和排序数据。DataRow 类用于表示表中实际的数据，使用 DataRow 和其属性与方法可以获取、估算和处理表中的数据。

关于 DataSet 可以参考神通数据库 DataAdapter 的示例。

第8章 大对象

ADO.NET 2.0 标准对大数据量对象的操作，没有相应的接口。对于二进制数据（Blob）的处理，和字符串数据(Clob)的数据(Character)处理，与其他形式的数据处理相同。除此之外，神通数据库目前支持 Oracle 兼容的 BFile 类型。

8.1BLOB

在.NET DATA PROVIDER 接口中没有创建一个 Blob 大对象的标准接口，因此在神通数据库的.NET DATA PROVIDER 驱动中，我们处理 Blob 大对象与其他 sql 的参数处理是相同的。方法如下：

8.1.1 插入 BLOB

对于 Blob 型的大对象，我们提供了两种方式来进行操作，byte[]数组形式作为参数和流(Stream)类型形式作为参数。对于相对较大的大对象，应该选择后者（流 stream 形式）。创建完 Blob 对象，如何将这个 Blob 对象插入到表中去呢?比如存在表 TABLEE，它有三个字段，第一个为 FIELD_SERIAL(INT)，第二个为 FIELD_BLOB(BLOB)，第三个为 FIELD_CLOB(CLOB)。括号内为该字段的数据类型。对于 Blob 我们可以有两种方法可以使用。

方法一（byte[]数组形式）：

```
OscarCommand commd = TheConnection.CreateCommand();
// 创建一个大对象的流对象
System.IO.FileStream fstream = new System.IO.FileStream("../../../testsuite/testFile/g.wmv",
System.IO.FileMode.Open);
// byte[]数组作为大对象的表述方式
byte[] buffer = new byte[fstream.Length];
fstream.Read(buffer, 0, (int)fstream.Length);
fstream.Close();

// 插入大对象的 sql 语句。
  commd.CommandText = "insert into tablee(FIELD_BLOB) values(@value);";
// 将大对象的 byte[]数组值付给这个参数
  commd.Parameters.Add("@value", OscarDbType.Blob).Value = buffer;
// 执行插入操作
commd.ExecuteNonQuery();
```

方法二（stream 流形式）：

```
OscarCommand commd = TheConnection.CreateCommand();
// 创建一个大对象的流对象
System.IO.FileStream fstream = new System.IO.FileStream("../../../testsuite/testFile/g.wmv",
System.IO.FileMode.Open);

// 插入大对象的 sql 语句。
commd.CommandText = "insert into tablee(FIELD_BLOB) values(@value);";
// 将大对象的流对象值付给这个参数
commd.Parameters.Add("@value", OscarDbType.Blob).Value = fstream ;
// 执行插入操作
commd.ExecuteNonQuery();
```

8.1.2 获取 BLOB

获取 Blob 对象非常简单，与其他类型字段相同。获取 Blob 对象转换成 byte[]数据。

比如上一例子中的表，读取 Blob 对象。那么采用如下方法就可以获得 Blob 对象：

```
// 等待接收大对象 Blob 的 byte[]数组
byte[] buffer = null;
// 读取 FIELD_BLOB 字段的 sql
commd.CommandText = "select FIELD_BLOB from tablee where field_serial=(select max(field_serial) from tablee);";
// 执行命令，获得结果集
OscarDataReader reader = commd.ExecuteReader();
if (reader.Read())
{
// 获得大对象的 byte[]形式表述
   buffer = (byte[])(reader[0]);
}
```

8.1.3 删除 BLOB

删除 Blob 对象非常简单，只要删除该条记录，数据库服务器会自动删除该 Blob 对象。对于在上面插入的大对象，只要删除该大对象所在的记录就行了：

```
String sql = "DELETE FROM tablee WHEREfield_serial=(select max(field_serial) from tablee);";
//执行 SQL 语句，删除 field_serial 为最大的记录，Blob 对象也自动被删除了
commd.ExecuteNonQuery(sql);
```

8.1.4 BLOB 应用程序示例

在这一节中我们将给出一个应用 Blob 对象的完整例子。在示例中，我们读取文件，写到数据库中，然后从数据库中将文件内容读出来，写到另外一个文件里。

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.OscarClient;
using System.Data;
using System.Data.SqlClient;
namespace TestProviderForHelpfiles
{
    class Program
     {
          static void Main(string[] args)
          {
              string connectString = "Server=10.0.5.226;Port=2003;User Id=SYSDBA;Password=szoscar55;Database=OSRDB;Encoding=GBK;";
                OscarConnection TheConnection = new OscarConnection(connectString);
                TheConnection.Open();
                bool testBlob = true;
                OscarCommand commd = TheConnection.CreateCommand();
                System.IO.FileStream fstream = new System.IO.FileStream(@"d:/6.jpg", System.IO.FileMode.Open);
                try
                {
                     commd.CommandText = "insert into tablee(FIELD_BLOB)
```

```
values(@value);";
                        commd.Parameters.Add("@value", OscarDbType.Blob).Value = fstream;
                        commd.ExecuteNonQuery();
                        byte[] buffer = null;
                        commd.CommandText = "select FIELD_BLOB from tablee where
ID=(select max(ID) from tablee);";

                        OscarDataReader reader = commd.ExecuteReader();
                        if (reader.Read())
                        {
                            buffer = (byte[])(reader[0]);
                        }
                        System.IO.FileStream Wstream = new System.IO.FileStream("d:/6.jgp",
System.IO.FileMode.OpenOrCreate);
                        Wstream.Write(buffer, 0, buffer.Length);
                        Wstream.Flush();
                        Wstream.Close();
                        reader.Close();
                        OscarDataAdapter adapter = new OscarDataAdapter(commd);
                        DataTable dataTable = new DataTable();
                        adapter.Fill(dataTable);
                        buffer = (byte[])(dataTable.Rows[0][0]);
                    }
                    catch (Exception e)
                    {
                        Console.WriteLine("异常：" + e.Message);
                        Console.Read();
                     }
    }
            }
    }
```

8.2 CLOB

在.NET DATA PROVIDER 接口中没有创建一个 Clob 大对象的标准接口，因此在神通数据库的.NET DATA PROVIDER 驱动中，我们处理 Clob 大对象与其他 sql 的参数处理是相同的。方法如下：这里使用与上面相同的表和连接对象 TheConnection

8.2.1 插入 CLOB

对于 Clob 型的大对象，我们目前提供一种方式进行操作，就是把 Clob 对象视为长度较大的 String 对象。具体的操作与 Blob 操作相似。

```
    commd.CommandText = "insert into tablee(FIELD_CLOB) values(@value);";
    StringBuilder sb = new StringBuilder();
     for (int i = 0; i < 1000000; i++)
     {
     sb.Append("gagasgagagdengdenggjgljlsjglsg");
       }
    commd.Parameters.Add("@value", OscarDbType.Clob).Value = sb.ToString();
    commd.ExecuteNonQuery();
```

从上边的例子看，对于参数的操作与其他类型的数据操作没有不同。

8.2.2 获取 CLOB

获取 Clob 对象非常简单，获得 OscarDataReader 或者调用 OscarCommand 对象的 ExecuteScalar 方法就能获取 Clob 对象。

```
commd.CommandText = "select FIELD_CLOB from tablee where filed_serial= (select max(filed_serial) from tablee)";
OscarDataReader reader = commd.ExecuteReader();
 if (reader.Read())
 {
      Console.WriteLine((String)reader[0]);
 }
```

8.2.3 删除 CLOB

删除 Clob 对象非常简单，只要删除该条记录，数据库服务器会自动删除该 Clob 对象。对于上面操作中插入的大对象，只要删除该大对象所在的记录就行了：

```
String sql = "DELETE FROM tablee WHEREfield_serial=(select max(field_serial) from tablee);";
//执行 SQL 语句，删除 field_serial 为最大的记录，Clob 对象也自动被删除了
commd.ExecuteNonQuery(sql);
```

8.3 BFILE

神通的 BFILE 数据类型是一种 LOB 数据类型，包含对最大 2GB 的二进制数据的引用。神通的 BFILE 不同于其他 LOB 数据类型的主要特征在于它的数据存储是在数据库服务器所在机器的操作系统的物理文件中，而不是数据库服务器中。BFILE 数据类型应该用于引用尺寸较大的 LOB，因此要将该数据类型存储在数据库中非常不切实际。另外与使用 LOB 数据类型相比，使用 BFILE 数据类型会有更多的客户端、服务器的通信开销。如果只是获取整个对象的少量数据，则以 BFILE 方式访问会更加有效。否则如需获取整个对象，则以访问驻留在数据库中的 LOB 更为有效。

8.3.1 操作方法

神通的 BFILE 类型在程序中被称为 OscarBFile。

每个非空 OscarBFile 对象都与两个定义物理文件位置的实体关联：

一个 DIRECTORY 对象，它是文件系统中一个目录的数据库别名。

基础物理文件的文件名，位于 DIRECTORY 所指定的目录中。

创建了 BFILE 之后，就可以使用 ExecuteReader 或 ExecuteScalar 方法，通过调用 GetOscarBFile 方法获取 OscarBFile 对象，并检索对应的引用定位器。任何使用 Read 或 Seek 方法访问已经关闭的 OscarBFile 的操作都会自动重新打开一个 OscarBFile 流。

8.3.2 注意事项

由于 BFILE 类型是只读的，因此不支持 WRITE 方法，包括从 System.IO.Stream 继承的 BeginWrite，EndWrite 和 WriteByte 方法。

8.3.3 程序示例

下面的示例使用了 C#语言调用.NET Data Provider 方式实现。说明了如何在 Oscar 表中创建 BFILE，然后使用 OscarBFile 对象形式对 BFIle 进行获取和检索。

```
//Create and execute the commands.
OscarCommand command = connection.CreateCommand();
command.CommandText = "CREATE OR REPLACE DIRECTORY TestDir AS 'c:\\bfiles'";
command.ExecuteNonQuery();
command.CommandText = "CREATE TABLE TestTable(col1 number, col2 BFILE)";
```

```
      command.ExecuteNonQuery();
       command.CommandText = "INSERT INTO TestTable VALUES ('2',
BFILENAME('TESTDIR', 'File.jpg'))";
       command.ExecuteNonQuery();
       command.CommandText = "SELECT * FROM TestTable";

       //Read the BFile data.
       byte[] buffer = new byte[100];
       OscarDataReader dataReader = command.ExecuteReader();
       using (dataReader)
       {
              if (dataReader.Read())
              {
                     OscarBFile BFile = dataReader.GetOscarBFile(1);
                     using (BFile)
                     {
                            BFile.Seek(0, SeekOrigin.Begin);
                            BFile.Read(buffer, 0, 100);
                     }
               }
       }
```

第9章 事务特性

事务为保证数据的完整性，数据的一致性和程序语义的正确性提供了一种机制。事务机制中主要包括提交模式、事务隔离级别和保存点三部分。神通数据库和神通数据库 .NET DATA PROVIDER 驱动程序，对这三部分给出了较好的实现方法。在本章中，我们将依次介绍它们的特性和使用方法。

9.1提交模式

应用程序创建一个连接时，该连接默认事务自动提交，即每执行一次 SQL 语句，该 SQL 语句就被认为是一个事务，而被自动提交了。如果用户需要将多条语句包含在一个事务中，那么用户就要改变自动提交模式。用户可以在 Connection 对象中开始一个事务，并且可以提供事务提交模式，方法如下：

```
OscarConnection conn = new OscarConnection(ConnectionString);
OscarTransaction trans = conn.BeginTransaction(IsolationLevel.Value);
……<some code>
……trans.Commit();   //或者是回滚事务  trans.Rollback();
```

如果开始了一个事务，则得到一个事务对象 OscarTransaction 对象。在事务结束时，应该调用事务的 Commit 方法将事务提交或者 Rollback 方法将事务回滚。OscarConnection 的 BeginTransaction 方法可以不带参数，则默认的事务隔离级别为 IsolationLevel.ReadCommitted; 也可以在 BeginTransaction 方法时作为参数给出。

在非自动提交模式下，用户在一个事务执行完成之后需要显式调用 OscarTransaction 对象的 commit()方法来提交事务。方法如下：

```
trans.Commit();
```

当该事务需要回滚时，则需要显式调用 OscarTransaction 对象的 Rollback()方法。

```
trans.Rollback();
```

当事务执行完被提交或回滚后，驱动程序将自动地释放事务对象，如果要再开启一个事务，则需要重复上面的步骤。如果事务对象没有显示的调用 Commit 方法或者 Rollback 方法，在该 Connection 关闭的时候（Close 方法的时候），会自动调用事务对象的 Commit 方法进行提交。

9.2隔离级别

神通数据库支持的事务隔离级别有：

```
TRANSACTION_READUNCOMMITTED
TRANSACTION_READCOMMITTED
TRANSACTION_REPEATABLE_READ
TRANSACTION_SERIALIZABLE
```

默认的事务隔离级别为 TRANSACTION_READ_COMMITTED。用户可以通过通过 Connection 接口中的 BeginTransaction 方法来开始一个事务，并设置事务隔离级别。

由于事务隔离级别升高时，数据库系统为了保证语义的正确性，会加入更多的锁。当锁加得过多时，会降低程序的性能。因此在确定事务隔离级别的时候，用户应充分考虑数据一致性和性能要求之间的平衡。

9.3保存点(savepoint)

保存点（Savepoint）提供了对事务更小粒度的控制方法，它代表一个逻辑事务点。在非自动提交模式下，一个事务中可以设置多个保存点，这样在回滚的时候，可以回滚到指定的

保存点，从事务开始到该保存点之间的操作依然有效。这就为事务处理提供了更多的灵活性。保存点方法为事务对象的 Save 方法。

保存点方法必须为保存点命名。下面的例子演示了保存点的使用：

```
// Rollback 方法用于回滚当前事务，Rollback 方法既可以回滚整个事务，也可以回滚到某个 savepoint。

// 使用 rollback 方法回滚整个事务
conn = new OscarConnection(connStr);
conn.Open();
txn = conn.BeginTransaction();
txn.Rollback();
txn.Dispose();

// 使用 rollback 方法回滚到某个特定的 savepoint
txn = conn.BeginTransaction();
txn.Save("SavePoint1");
txn.Save("SavePoint2");
txn.Rollback("SavePoint2");
txn.Rollback("SavePoint1");
txn.Rollback();
txn.Dispose();

// 离开活动事务范围后调用 Rollback 会报错
txn = conn.BeginTransaction();
txn.Rollback();

gotIt = false;
try{
txn.Rollback();
}catch(OscarException e){
gotIt = true;
}
// 这里 gotIt 的值会为 true
//Assert.IsTrue(gotIt);
Console.WriteLine(gotlt.ToString());
txn.Dispose();

// 指定错误的 savepoint，后台报错
txn = conn.BeginTransaction();
txn.Save("SavePoint3");

gotIt = false;
try{
txn.Rollback("SavePoint4");
}catch(OscarException e){
gotIt = true;
}
//Assert.IsTrue(gotIt);
Console.WriteLine(gotlt.ToString());
txn.Dispose();

conn.Close();
```

这里应用的保存点和 Rollback 方法（以保存点名为参数）配合使用，可以使事务回滚到某一个点上。

用户也可以调用 Rollback 方法（无参数）来回滚全事务（无视保存点的存在）。一旦回滚，事务对象立即变为 null，所有保存点也将无效，试图引用也会导致异常。这里要注意另外一种情况，比如按顺序设置了保存点 sp1，sp2，sp3，那么当用户回滚到 sp2 时，那么 sp3 保存点将不复存在，对 sp3 的释放或回滚操作将导致 PROVIDER 驱动程序抛出异常。

第10章　标量函数

神通数据库 .NET DATA PROVIDER 驱动支持规范要求的标量函数，包括数量函数、字符串函数、时间和日期的函数、系统函数、转换函数。比如要调用一个 ABS(number)的标量函数，就像下面这样直接调用，并返回结果。

```
    OscarCommand cmd = new OscarCommand("SELECT    ABS(-5)",
TheConnection);
    OscarDataReader rd = cmd.ExecuteReader();
    if (rd.Read())
    {
    int value = rd.GetInt32(0);
    }
```

执行上述代码后，变量 value 的值就为 5。

10.1　数量函数(Numeric Functions）

ABS(number) 求一个数的绝对值

ACOS(float) 反余弦函数，弧度用浮点数表示

ASIN(float) 反正弦函数，弧度用浮点数表示

ATAN(float) 反正切函数，弧度用浮点数表示

ATAN2(float1，float2) 反正切函数，float1 表示对边，float2 表示邻边

CEILING(number) 返回一个大于等于 number 的最小整数

COS(float) 余弦函数，float 表示弧度数

COT(float) 余切函数，float 表示弧度数

DEGREES(number) 弧度 number 对应的角度值

EXP(float) float 的自然指数

FLOOR(number) 返回一个小于等于 number 的最大整数

LOG(float) 返回以 e 为底 float 的对数

LOG10(float) 返回以 10 为底 float 的对数

MOD(integer1, integer2) 整数 integer1 除以 integer2 的余数

PI() 返回常数 π

POWER(number, power) 返回 number(整数)的 power 次幂

RADIANS(number) 角度 number 对应的弧度值

RAND(integer) 根据种子 integer 产生的随机浮点数

ROUND(number, places) 将 number 根据 places 规定的位置进行舍入

SIGN(number) 符号函数

SIN(float) 正弦函数，弧度用浮点数 float 表示

SQRT(float) 浮点数 float 的平方根

TAN(float) 正切函数，弧度为浮点数 float 表示

TRUNCATE(number, places) 根据 places 指定的位置将 number 去尾

10.2 字符串函数(String Functions）

ASCII(string) 返回字符串 string 最左边的字符对应的 ASCII 码

CHAR(code) 返回 ASCII 码 code 对应的字符，code 介于 0 到 255

CONCAT(string1, string2) 将字符串 string2 连接到字符串 string1 后，如果其中一个字符串为空，则结果为空

DIFFERENCE(string1, string2) 返回一个整数，表明 string1 和 string2 由函数 SOUNDEX 返回值的差别。

INSERT(string1,start,length,string2) 将字符串 string1 从 start 的位置开始删除 length 个字符，并将 string2 插入到 string1 的开头。

LCASE(string) 将 string 中的所有大写字母转换为小写。

LEFT(string, count) 字符串 string 最左边的 count 个字符组成的字符串。

LENGTH(string) 字符串 string 去掉多余的空格后的长度。

LOCATE(string1, string2[, start]) 返回 string2 在 string1 中首次出现的位置。如果没有指定 start 位置，则从 string2 的最左边开始查找，如果指定了 start 位置，则从该位置开始查找。如果没有找到，返回 0；如果找到，Position 1 对应 string2 中的第一个字符。

LTRIM(string) 去掉 string 中前面的空格后得到的字符串。

REPEAT(string, count) 返回一个将字符串 string 重复 count 次后形成的一个新字符串。

REPLACE(string1,string2,string3) 将 string1 中的所有 string2 用 string3 代替。

RIGHT(string, count) 字符串 string 最右边的 count 个字符组成的字符串。

RTRIM(string) 将字符串 string 去掉多余的空格后得到的字符串。

SOUNDEX(string) 返回一个表示 string 中单词发音的音标字符串，是一个四位的 SOUNDEX 码，一个音标对应一个单词。

SPACE(count) 返回一个包含 count 个空格的字符串。

SUBSTRING（string, start, length） 从字符串 string 中抽取 start 开始的长度为 length 的子串。

UCASE（string）将 string 中的字符转换成大写

10.3 时间和日期的函数(Time and Date Functions)

CURDATE() 返回当前日期的 date 型值。

CURTIME() 返回当前本地时间的 time 类型值。

DAYNAME(date) 返回一个代表 date 中 day 分量的字符串。

DAYOFMONTH(date) 返回一个 1－31 之间的整数，代表 date 所处一月中的某一天。

DAYOFWEEK(date) 返回一个 1－7 间的整数，代表 date 所处一周中的某一天。1 表示星期天。

DAYOFYEAR(date) 返回一个 1－366 间的整数，代表 date 所处一年中的某一天。

HOUR(time) 返回一个 1－23 间的整数值，表示 time 中的小时分量。

MINUTE(time) 返回一个 0－59 间的整数值，表示 time 中的分钟分量。

MONTH(date) 返回一个 1－12 之间的整数值，表示 date 中的月分量。

MONTHNAME(date) 返回一个字符串，表示 date 中的月分量。

NOW() 返回一个代表当前时间和日期的 timestamp 类型的值。

QUARTER(date) 返回一个 1－4 之间的整数，表示 date 所处的一年中的某一个季节，1 表示 1 月 1 日至 3 月 31 日。

SECOND(time) 返回一个 0－59 的整数，代表 time 中的秒分量。

TIMESTAMPADD(interval, timestamp) 将 interval 加到 timestamp，得到一个新的 timestamp 值，interval 可以是如下一些数据类型： SQL_TSI_FRAC_SECOND，SQL_TSI_SECOND， SQL_TSI_MINUTE， SQL_TSI_HOUR， SQL_TSI_DAY，SQL_TSI_WEEK， SQL_TSI_MONTH， SQL_TSI_QUARTER， or SQL_TSI_YEAR。

TIMESTAMPDIFF(interval, timestamp1, timestamp2) 返回一个整数，该整数表明 timestamp2 比 timestamp1 之间的时间间隔，间隔可以是以下这些数据类型：SQL_TSI_FRAC_SECOND， SQL_TSI_SECOND， SQL_TSI_MINUTE， SQL_TSI_HOUR，SQL_TSI_DAY， SQL_TSI_WEEK， SQL_TSI_MONTH， SQL_TSI_QUARTER， or SQL_TSI_YEAR。

WEEK（date）用一个 1－53 之间的整数来表明 date 对应的一年中的某一个周。

YEAR(date) 返回 date 中的年对应的整数。

10.4 系统函数(System Functions)

DATABASE( ) 返回数据库的名字。

IFNULL(expression, value) 如果 expression 为空，返回 value；如果 expression 非空，返回 expression。

USER() 返回 DBMS 系统的用户名。

10.5 转换函数(Conversion Functions)

CONVERT(value,SQLtype) 将数据从一种类型转换为另一种类型的值，SQLtype 的类型表示可以为下列这些值 BIGINT，BINARY， BIT，CHAR， DATE， DECIMAL， DOUBLE，FLOAT，INTEGER，LONGVARBINARY，LONGVARCHAR，REAL，SMALLINT，TIME，TIMESTAMP，TINYINT，VARBINARY，VARCHAR。

第11章　A 参考资料

关于更详细的.NET DATA PROVIDER 的信息，可以参考 MSDN 的 ADO.NET 2.0 规范.