
第 7 章 数据库与 ADO.NET 基础

数据库在任何应用程序开发中都非常的重耍，特别在 ASP.NET 应用程序开发中，数据库通常被用来保存用户的信息、文章内容等数据，同时数据库也能够提供用户进行查询、搜索等操作。传统的纯静态 HTML 页面已经不能满足互联网的发展应用，使用数据库能够让网站与用户、新闻、投票等信息进行良好的整合。

7.1 数据库基础

要了解数据库，首先就要掌握数据库基础，数据库就是存放数据的仓库。当开发人员在应用程序的开发中，可以将任何可以抽象成数据的信息存放在数据库中，数据库的特点是数据能够按照数据模型组织进行存取，数据库是高度的结构化并且可以为多个用户共享的。

7.1.1 结构化查询语言

结构化查询语言简称“SQL”，最早的是圣约瑟研究实验室为其关系数据库管理系统 SYSTEM R 开发的一种查询语言。现今的数据库，无论是大型的数据库，如 Oracle、Sybase、Informix、SQL server 这些大型的数据库管理系统，还是 Visual Foxpro、PowerBuilder 这些微机上常用的数据库开发系统，都支持 SQL 语言作为查询语言。

SQL 是高级的非过程化编程语言，允许用户在高层数据结构上工作，它不要求用户指定对数据的存放方法，也不需要用户了解具体的数据存放方式，所以具有完全不同的底层结构的不同数据库系统都可以使用相同的 SQL 语言作为数据输入与管理的接口。它以记录集作为操作对象，所有 SQL 语句接受集合作为输入，返回集合作为输出，这种集合特性允许一条 SQL 语句的输出作为另一条 SQL 语句的输入，所以 SQL 语言可以嵌套，这也使 SQL 语句具有极大的灵活性和强大的功能。在多数情况下，在其他语言中需要一大段程序实现的一个单独事件只需要一个 SQL 语句就可以达到目的，这也意味着用 SQL 语言可以写出非常复杂的语句。下面给出一组例子来演示 SQL 语句的使用方法。

1. 查询表中所有记录

通过使用 select 关键字进行查询，示例代码如下所示。

```
SELECT * FROM NEWS
```

2. 带条件的查询语句

通过使用 where 语句进行带条件的查询，示例代码如下所示

```
SELECT * FROM NEWS WHERE TITLE='新闻'
```

3. 使用函数

语句中也可以使用内置函数，示例代码如下所示。

```
SELECT COUNT(*) AS MYCOUNT FROM NEWS
```

4. 插入数据语句

通过使用 insert 进行插入数据库操作，示例代码如下所示。

```
INSERT INTO NEWS VALUES ('新闻','2008/9/9','新闻内容')
```

5. 删除数据语句

通过使用 delete 关键字删除数据库中的数据，示例代码如下所示。

```
DELETE FROM NEWS WHERE ID=1
```

注意：当 delete 后面的条件没有限定时，则会删除该表的所有数据。

6. 更新数据语句

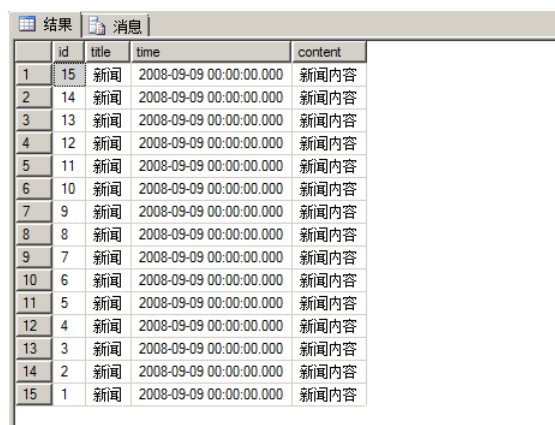
通过使用 update 关键字更新数据，示例代码如下所示。

```
UPDATE NEWS SET TITLE='新闻标题' WHERE ID='1'
```

注意：SQL 并不区分大小写，但是推荐使用大写来书写 SQL 语句，这样能够在应用程序中清晰的辨认。

7.1.2 表和视图

表是关系数据库中最主要的数据库对象，开发人员通过创建表并向表中进行数据操作来存储和操作数据，表是用来存储和操作数据的一种逻辑结构。表通常以二维表形式呈现，在 SQL Server Management Studio 中可以看见表的结构，如图 7-1 所示。



	id	title	time	content
1	15	新闻	2008-09-09 00:00:00.000	新闻内容
2	14	新闻	2008-09-09 00:00:00.000	新闻内容
3	13	新闻	2008-09-09 00:00:00.000	新闻内容
4	12	新闻	2008-09-09 00:00:00.000	新闻内容
5	11	新闻	2008-09-09 00:00:00.000	新闻内容
6	10	新闻	2008-09-09 00:00:00.000	新闻内容
7	9	新闻	2008-09-09 00:00:00.000	新闻内容
8	8	新闻	2008-09-09 00:00:00.000	新闻内容
9	7	新闻	2008-09-09 00:00:00.000	新闻内容
10	6	新闻	2008-09-09 00:00:00.000	新闻内容
11	5	新闻	2008-09-09 00:00:00.000	新闻内容
12	4	新闻	2008-09-09 00:00:00.000	新闻内容
13	3	新闻	2008-09-09 00:00:00.000	新闻内容
14	2	新闻	2008-09-09 00:00:00.000	新闻内容
15	1	新闻	2008-09-09 00:00:00.000	新闻内容

图 7-1 表的表现形式

创建表可以使用 SQL 语句进行创建，下面是创建表的表脚本代码。

```
CREATE TABLE [dbo].[news](  
    [id] [int] IDENTITY(1,1) NOT NULL,  
    [title] [nvarchar](50) NULL,  
    [time] [datetime] NULL,  
    [content] [ntext] NULL,  
)
```

上述代码创建了一个新闻表并且该表具有 4 个字段，这 4 个字段分别为 id、title、time 和 content，表是一个具体的表，用于数据的存放和读取。视图不同于表，视图并不是实际存在的表，视图是一种虚拟的表，视图将存在的表中按照一定的规则读取若干列，组成新的结果集，视图在物理上并不存在。当对视图进行操作时，系统会根据视图的定义去操作与视图相关联的基本表。视图有助于隐藏现有的表中的数据，创建视图代码如下所示。

```
CREATE VIEW myview as
```

```
SELECT title,[time] from news
```

上述代码创建了一个视图，是基于查询语句 `select title,[time] from news` 所查询的集合的。

注意：视图不是一个表，是一个虚拟的表，视图可以是多个表的集合、筛选形成的新表，视图是这些表的一个结果集。

7.1.3 存储过程和触发器

存储过程是一组为了完成特定功能的 SQL 语句集，在编写完成后，系统会编译代码并存储在数据库中。用户只需要指定存储过程的名字并给出传递的参数，就可以使用存储过程。存储过程的概念有点像应用程序开发中的方法。

1. 存储过程

存储过程是数据库中一个非常重要的对象，使用好存储过程能够将数据库应用与程序应用相分离。当维护与数据库相关的功能的时候，只需要维护存储过程即可，另外使用存储过程能够提升性能，存储过程和过程会在运行中被编译，当没有显著的数据更新时，可以直接从编译后的文件中获取相应的结果。存储过程优点如下所示：

- ☐ 存储过程允许标准组件式编程。
- ☐ 存储过程的执行速度较快。
- ☐ 存储过程能够减少网络流量，降低应用程序读取数据库的次数。
- ☐ 存储过程比查询语句更加安全。

存储过程声明语法如下所示：

```
CREATEPROC[EDURE]procedure_name[:number]
  [{@parameterdata_type}
  [VARYING][=default][OUTPUT]
  ][...n]
  [WITH
  {RECOMPILE|ENCRYPTION|RECOMPILE,ENCRYPTION}]
  [FORREPLICATION]
  ASsql_statement[...n]
```

存储过程的各个参数的使用如下所示。

- ☐ **procedure_name**：新存储过程的名称，过程名必须符合标识符规则，且对于及其所有者必须唯一。
- ☐ **number**：是可选的整数，用来对同名的过程分组，以便使用一条 **DROPPROCEDURE** 语句即可将同组的过程一起除去。
- ☐ **@parameter**：过程中的参数。在 **CREATEPROCEDURE** 语句中可以声明一个或多个参数。用户必须在执行过程时提供每个所声明参数的值。
- ☐ **data_type**：参数的数据类型。所有数据类型如 **text**、**ntext** 和 **image** 均可以用作存储过程的参数，而与之不同的是，**cursor** 数据类型只能用于 **OUTPUT** 参数。
- ☐ **VARYING**：指定作为输出参数支持的结果集，其由存储过程动态构造，内容可以变化，**VARYING** 仅适用于游标参数。
- ☐ **default**：参数的默认值。如果定义了默认值，不必指定该参数的值即可执行过程，默认值必须是常量或 **NULL**，如果过程将对该参数使用 **LIKE** 关键字，那么默认值中可以包含通配符（*、_、[]和^）。

- ❑ **OUTPUT**: 表明参数是返回参数。该选项的值可以返回给 EXEC[UTE]。使用 OUTPUT 参数可将信息返回给调用过程。
- ❑ **n**: 表示最多可以指定 2100 个参数的占位符。
- ❑ **{RECOMPILE|ENCRYPTION|RECOMPILE,ENCRYPTION}**: RECOMPILE 表明 SQLSERVER 不会缓存该过程的计划, 该过程将在运行时重新编译; ENCRYPTION 表示 SQLSERVER 加密 syscomments 表中包含 CREATEPROCEDURE 语句文本的条目; 使用 ENCRYPTION 可防止将过程作为 SQLSERVER 复制的一部分发布。

通过以上参数可以声明一个存储过程, 示例代码如下所示。

```
CREATE PROCEDURE UpdatenewsInfo
@ID int,
@title nvarchar(50),
@time datetime,
@content ntext,
AS
UPDATE [newsInfo]
Set News Title=@title,News Datetime=@time
where [ID]=@ID
GO
```

上述代码创建了一个名为“Updatenewsinfo”的存储过程, 该存储过程作用是修改新闻表中的相应的字段的值。

2. 触发器

触发器实际上也是一种存储过程, 不过触发器是一种特殊的存储过程, 当使用 UPDATE, INSERT 或 DELETE 的一种或多种对指定的数据库的相关表进行操作时, 会触发触发器。触发器的语法格式如下所示。

```
CREATE TRIGGER trigger_name
ON { table | view }
[ WITH ENCRYPTION ]
{
{ { FOR | AFTER | INSTEAD OF } { [ INSERT ] [ , ] [ UPDATE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS
[ { IF UPDATE ( column )
[ { AND | OR } UPDATE ( column ) ]
[ ...n ]
| IF ( COLUMNS_UPDATED ( ) { bitwise_operator } updated_bitmask )
{ comparison_operator } column_bitmask [ ...n ]
} ]
sql_statement [ ...n ]
}
}
```

其中, 触发器的各个参数的使用如下所示。

- ❑ **trigger_name**: 是触发器的名称。触发器名称必须符合标识符规则, 并且在数据库中必须惟一, 开发人员可以选择是否指定触发器所有者名称。
- ❑ **Table | view**: 是在其上执行触发器的表或视图, 有时称为触发器表或触发器视图, 可以选择是否指定表或视图的所有者名称。

-
- ❑ **WITH ENCRYPTION**: 加密 syscomments 表中包含 CREATE TRIGGER 语句文本的条目。使用 WITH ENCRYPTION 可防止将触发器作为 SQL Server 复制的一部分发布。
 - ❑ **AFTER**: 指定触发器只有在触发 SQL 语句中指定的所有操作都已成功执行后才激发, 所有的引用级联操作和约束检查也必须成功完成后, 才能执行此触发器。
 - ❑ **INSTEAD OF**: 指定执行触发器而不是执行触发 SQL 语句, 从而替代触发语句的操作。
 - ❑ **{ [DELETE] [,] [INSERT] [,] [UPDATE] }**: 是指定在表或视图上执行哪些数据修改语句时将激活触发器的关键字, 必须至少指定一个选项。在触发器定义中允许使用以任意顺序组合的这些关键字。如果指定的选项多于一个, 需用逗号分隔这些选项。
 - ❑ **WITH APPEND**: 指定应该添加现有类型的其他触发器, 只有当兼容级别是 65 或更低时, 才需要使用该可选子句。
 - ❑ **NOT FOR REPLICATION**: 表示当复制进程更改触发器所涉及的表时, 不应执行该触发器。
 - ❑ **AS**: 是触发器要执行的操作。
 - ❑ **sql_statement**: 是触发器的条件和操作, 触发器条件指定其他准则, 以确定 DELETE、INSERT 或 UPDATE 语句是否导致执行触发器操作。

触发器可以包含复杂的 SQL 语句, 主要用于强制复杂的业务规则或要求。同时, 触发器也能够维持数据库的完整性, 当执行插入、更新或删除操作时, 触发器会根据表与表之间的关系, 强制保持其数据的完整性。

7.2 使用 SQL Server 2005 管理数据库

SQL Server 2005 是微软继 SQL Server 2000 后 5 年发布的一款新的数据库产品。SQL Server 2005 不仅增加了许多功能, 同时也在 UI、管理工具、性能上做了很多的优化。使用 SQL Server 2005 管理网站数据库, 不仅提高了开发中数据的存储和读写的效率, 也更加方便了数据的管理。

7.2.1 初步认识 SQL Server 2005

相比于 SQL Server 2000, SQL Server 2005 在安装上更加的简单, 基本上无需手动配置任何事情即可安装。在安装之前, SQL Server 2005 会检查宿主机器的配置是否适合安装 SQL Server 2005, 如果机器的配置适合安装 SQL Server 2005, 则会进入安装主界面。SQL Server 2005 的安装向导是基于 Windows 的安装程序, 用户使用起来更加友好, 并且在安装过程中为用户提供了可选方案, 让用户选择自己需要的组件安装。

当安装完毕后, 用户可以打开 SQL Server 2005 软件体系中的 SQL Server Management 来配置和管理 SQL Server 2005。并进行数据操作。在进入 SQL Server Management 时, 对每个连接 SQL Server 2005 都要求一个连接实例, 进行身份验证, 如图 7-2 所示。

用户可以以 Windows 身份验证的方式登录到 SQL Server 2005 管理工具中, 也可以使用 SQL Server 身份验证的方式登录到 SQL Server 2005 管理工具, 相比之下, SQL Server 身份验证的方式更加安全。登入后 SQL Server Management 管理工具界面如图 7-3 所示。



图 7-2 SQL Server 2005 身份验证

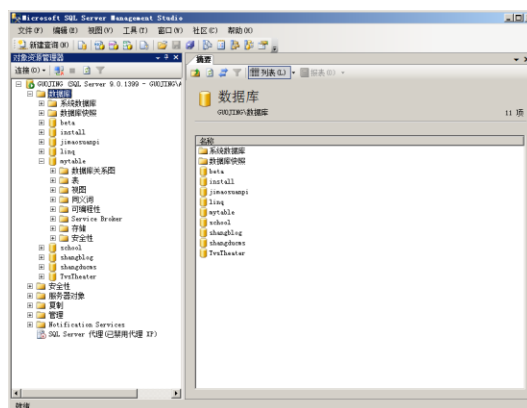


图 7-3 SQL Server Management 管理工具界面

在 SQL Server Management 管理工具中，表的操作与 SQL Server 2000 中并没有太大的差别，但是 SQL Server 2005 中没有了查询分析器，取而代之的是在 SQL Server 2005 中，可以直接在同一个窗口进行查询和数据操作，只需要单击导航栏上的【新建查询】按钮即可，如图 7-3 所示。

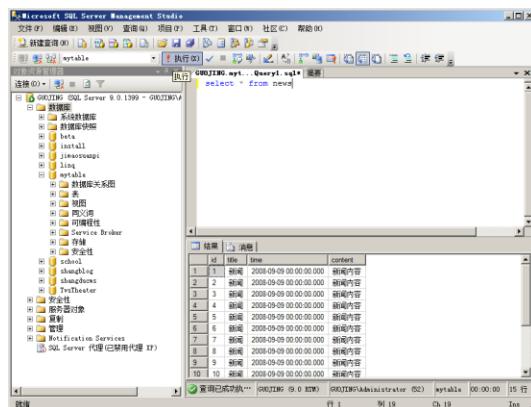


图 7-4 SQL Server Management 进行查询

对于普通的应用而言，SQL Server 2005 与 SQL Server 2000 并没有太大的区别。而对于高级的应，SQL Server 2005 做了相应的优化，SQL Server 2005 的操作更加友好，在数据的存储等性能上也有较大的提升。

7.2.2 创建数据库

使用 SQL Server Management 管理工具可以快速的创建数据库，在 SQL Server Management 管理工具中左侧的【对象资源管理器】选项中单击【数据库】选项，右击相应数据库，在下拉菜单中选择【新建数据库】。选择后，系统会显示一个创建数据库的向导，如图 7-5 所示。

通常来说，对于一般的应用，只需要填写数据库的名称，而数据和日志逻辑名称系统会自己填写。当有其他需求时，用户也可以更改逻辑名称，以及数据库存放的物理地址。在数据库的创建过程中，可以选择数据库的初始大小，最大值为多少，并且设置增量。当单击【确定】按钮后系统就创建完成数据库“mytable”，如图 7-6 所示。

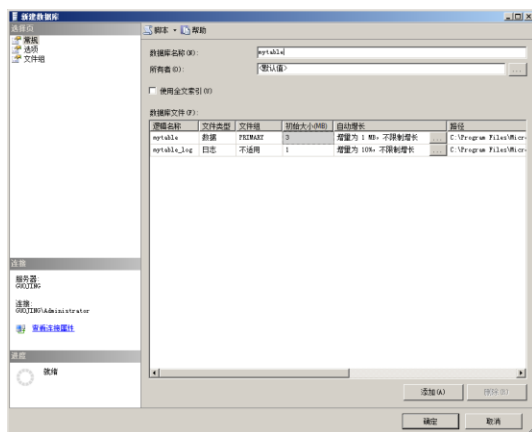


图 7-5 创建数据库

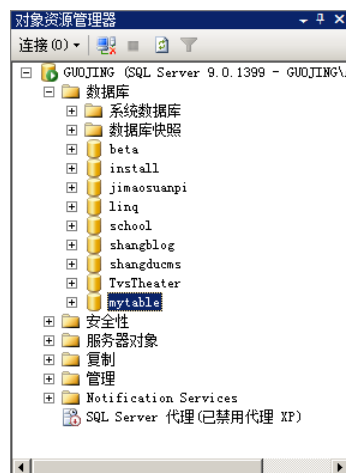


图 7-6 完成数据库的创建

对于任何可以使用 SQL Server Management 管理工具执行的操作，都可以通过 SQL 结构化查询语句来实现，同样，创建表的过程能够通过 SQL 语句来实现，示例代码如下所示。

```
CREATE DATABASE mytable
GO
```

在 SQL Server Management 管理工具中，新建查询，并将上述代码复制到代码块中，单击【执行】按钮，则会创建一个表 mytable。上述代码只是创建了一个简单的没有任何约束或功能的表，在 SQL 语句创建表语句中，使用 ON 子句可以设置数据库文件的属性，ON 子句的参数如下所示。

- ❑ PRIMARY: 设置主文件，ON 子句中只能出现一个 PRIMARY。
- ❑ NAME: 指定文件的逻辑名称。
- ❑ FILENAME: 指定文件的物理路径和名称。
- ❑ SIZE: 指定文件的初始大小。
- ❑ MAXSIZE: 指定文件大小的最大值。
- ❑ UNLIMITED: 指定文件将增长到磁盘变满位置。如果不指定此参数，当文件大小达到了 MAXSIZE 时，将存储为另外一个数据文件。
- ❑ FILEGROWTH: 定义文件的生长量。

当不指定以上参数时，系统会以默认方式创建数据库。若需要通过使用语句来自定义创建数据库，则可以使用 ON 子句并附上参数。示例代码如下所示。

```
CREATE DATABASE mytable
ON
PRIMARY (NAME=table1,
FILENAME='C:\PROGRAM FILES\MICROSOFT SQL SERVER\MSSQL\DATA\MYTABLEDAT1.MDF',
SIZE=100MB,MAXSIZE=200,FILEGROWTH=20)
GO
```

上述代码创建了一个 mytable 表，并指定了主文件为 table1，文件路径为 C:\PROGRAM FILES\MICROSOFT SQL SERVER\MSSQL\DATA\MYTABLEDAT1.MDF，并指定了初始大小为 100m，最大大小为 200m。

7.2.3 删除数据库

在 SQL Server Management 管理工具中，可以直接对数据库进行删除操作。在对象资源管理器中，

选中需要删除的数据库，右击选中的数据库，在下拉菜单中选择【删除】选项，SQL Server Management 管理工具出现一个删除向导，如图 7-7 所示。

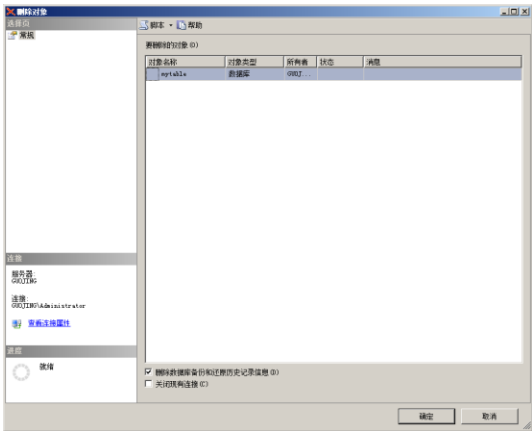


图 7-7 删除数据库

通常情况下，删除功能能够快速并安全的执行删除，但是有的时候，如数据库的连接正在被打开或数据库中的信息正被使用，那么就无法执行删除，必须勾选【关闭现有连接】复选框关闭现有连接。与创建数据库相同的是，删除数据库也可以使用 SQL 语句执行，删除数据库的 SQL 语法如下所示。

```
DROP DATABASE <数据库名>
```

当需要删除 mytable 数据库时，可以编写相应的 SQL 删除语句执行删除操作，示例代码如下所示。

```
DROP DATABASE mytable
GO
```

7.2.4 备份数据库

在数据库的使用中，通常会造成一些不可抗力或灾难性的损坏，如人工的操作失误，不小心删除了数据库，或出现了断电等情况，造成数据库异常或丢失。为了避免数据库中重要数据的丢失，就需要使用 SQL Server Management 管理工具来备份数据库。

SQL Server Management 管理工具备份数据库也非常的简单，在对象资源管理器中选择需要备份的数据库，右击需要备份的数据库，选择【任务】菜单，在【任务】菜单中单击【备份】按钮。单击【备份】按钮后，系统会出现一个备份向导，如图 7-8 所示。

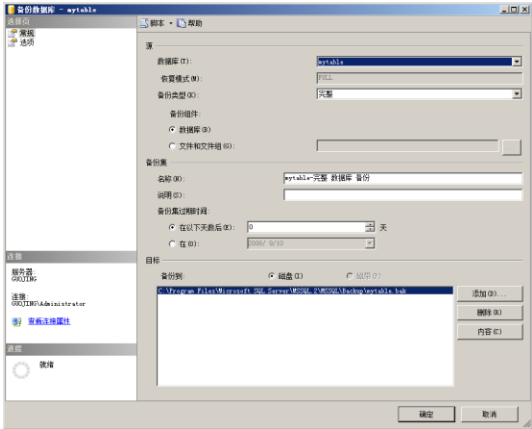


图 7-8 备份数据库

- ❑ 数据库：需要备份的数据库。
- ❑ 恢复模式：数据库的恢复模式。
- ❑ 备份类型：数据库的备份类型，通常有完全备份、差异备份、事物日志。
- ❑ 备份组件：通常可选数据库类型和文件类型。
- ❑ 名称：备份的名称。
- ❑ 说明：备份数据库所说的说明。
- ❑ 备份集过期时间：备份集过期的事件，可以设置过期时间。
- ❑ 备份到：选择备份的物理路径，可以选择备份到磁盘或磁带中。

7.2.5 还原数据库

注意：这里的“数据库”是所有数据库的统称的，并不是某个数据库，是数据库的集合。

[illegible]

指定备份

指定还原操作的备份媒体及其位置。

备份媒体 (B): 文件

备份位置 (L):

添加 (A)

删除 (D)

内容 (C)

确定 (O) 取消 帮助

单击【添加】按钮选择备份文件，如图 7-11 所示。

186

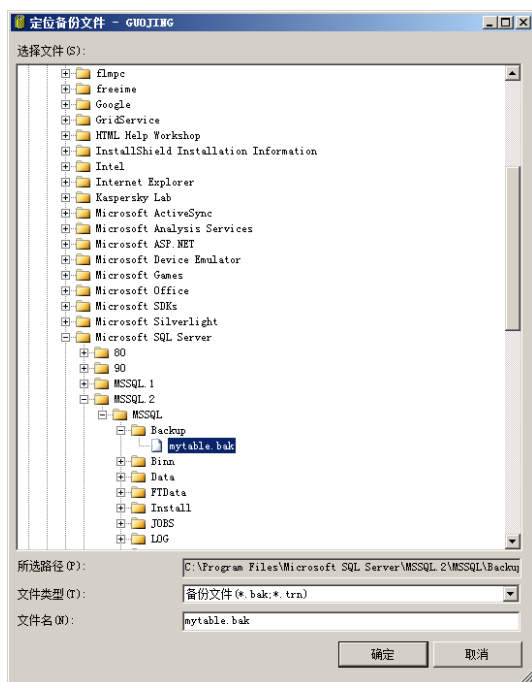


图 7-11 选择备份文件

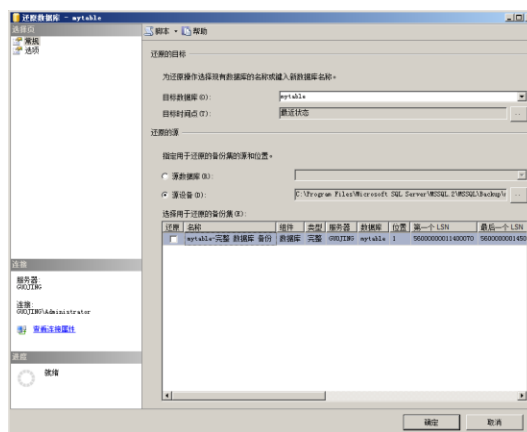


图 7-12 备份向导完成

单击【确定】按钮即可完成数据库的恢复，可以看见在对象资源管理器中，mytable 数据库又恢复了。备份数据库是一个非常良好的习惯，因为数据库保存着应用程序的所有信息，一旦数据丢失就会造成无法挽回的影响或亏损，经常备份数据库能够在数据丢失时进行数据的恢复，将应用程序的影响降低到最小。

7.2.6 创建表

在创建了数据库之后，就需要创建表来保存数据，SQL Server Management 管理工具可以可视化的为用户创建表操作。在定义表的结构中，需要说明表由哪些列组成，并且需要指定这些列的名称和数据类型。通过 SQL Server Management 管理工具可以可视化的创建表结构。在对象资源管理器中，选择相应数据库，右击相应的数据库，在下拉菜单中选择【新建表】选项，单击【新建表】按钮，系统会弹出一个新的 TAB 窗口，该窗口可以可视化的让用户创建表，如图 7-13 所示。



图 7-13 创建表

创建表中的列时，必须指定名称和数据类型。在上述创建表的过程中，创建了 `int` 数据类型的字段 `id` 和 `nvarchar` 数据类型的字段 `title`。

在表的结构中，有的列可以被设置为惟一的标识，如学生表中的学号，当设置了惟一的标识后，此列的数据在表中必须是惟一的、不能重复的。通常情况下，将表中的 `ID` 标识设置为主键。主键可以有效的约束添加到表中的值，被称为主键约束。为了保证约束主键和数据的完整性，定义的主键的字段将不允许插入空值。

技巧：在设计器中，在相应的字段上单击右键，选择“设置主键”即可将该字段设置为主键。

在应用程序开发中，通常需要将数据库中的编号设置为主键，通过编号来筛选内容。例如，当开发一个新闻系统中，新闻系统的编号是不应该重复的，所以可以设置为主键。同时，对 `int` 类型的主键可以设置为自动增长，当插入数据时，系统会根据相应的 `id` 号自动增长而不需要通过编程实现。在设计器中，可以为 `int` 类型的字段设置为自动增长，如图 7-14 所示。

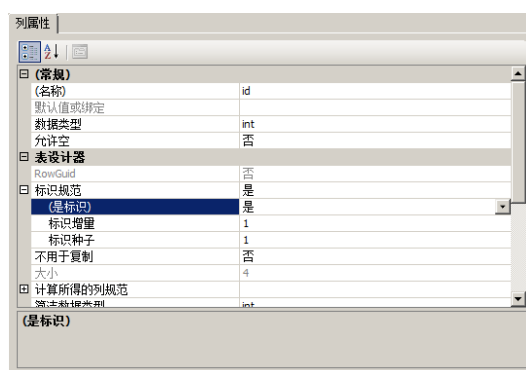


图 7-14 设置自动增长

将相应的字段设置了自动增长时，当插入一条数据，如果该表中没有任何数据，则表中的该字段为 1，当再次插入数据时，该字段则会自动增长到 2。在应用程序开发中，自动增长的字段经常被使用。

注意：当设置了主键，并配置了标识规范，就是配置自动增长，则在编写 SQL 的 `INSERT` 语句时无需向自动增长的列插入数据。

通过 SQL Server Management 管理工具可以创建表，也可以通过 SQL 语句创建表，创建表的语法结构如下所示。

```
CREATE TABLE 表名
(
    列名 数据类型,
    列名 1 数据类型
    ....
)
```

在创建表中的字段时，也可以使用关键字来约定字段。例如可以使用 **IDENTITY** 关键字来定义一个字段为自动增长列。也可以使用 **PRIMARY KEY** 关键字定义当前列为为主键。同样，当规定用户插入一个列时，必须填写字段，则可以使用 **NOT NULL** 关键字。示例代码如下所示。

```
CREATE TABLE mynews
(
    ID INT IDENTITY PRIMARY KEY,
    TITLE NVARCHAR(50)
)
```

注意：当使用语句创建数据库时，必须在导航栏中选择相应的数据库，默认的数据库为 **master**，在执行 SQL 语句前需选择相应操作的目标数据库。

7.2.7 删除表

使用 SQL Server Management 管理工具能够快速的删除表。在对象资源管理器中，右击相应表，在下拉菜单中选择【删除】选项即可，如图 7-15 所示。

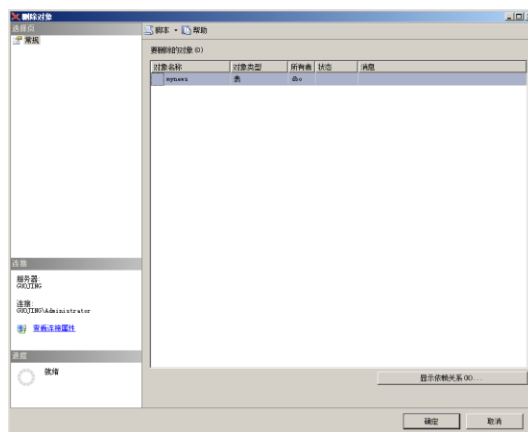


图 7-15 删除表

删除表与前面的删除数据库的操作非常的像，只需单击【确定】按钮即可删除该表。同样，在删除表时，也可以使用 SQL 语句删除表，语法结构如下所示。

```
DROP TABLE 表名称
```

当需要删除 **mynews** 表时，可以使用 **DROP** 语句来删除，示例代码如下所示。

```
DROP TABLE mynews
```

7.2.8 创建数据库关系图

在大型关系型数据库中，数据表很多，关系非常复杂。通过关系图，可以很清楚的分析数据库中表的关系。同时，通过这个关系图，你也可以对这些关系进行操作，可以说是一个图形化的关系操作入口。

在 SQL Server Management 管理工具中，单击相应的数据库，选择数据库关系图，单击右键，在下拉菜单中选择【新建数据库关系图】选项，系统会提示选择表来创建数据库关系图，如图 7-16 所示。选择需要的数据库中的表，单击【添加】按钮，则会出现关系图，如图 7-17 所示。

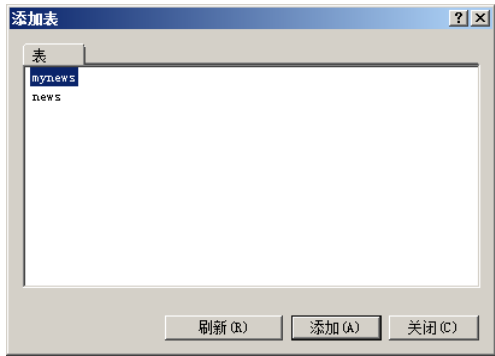


图 7-16 创建数据库关系图

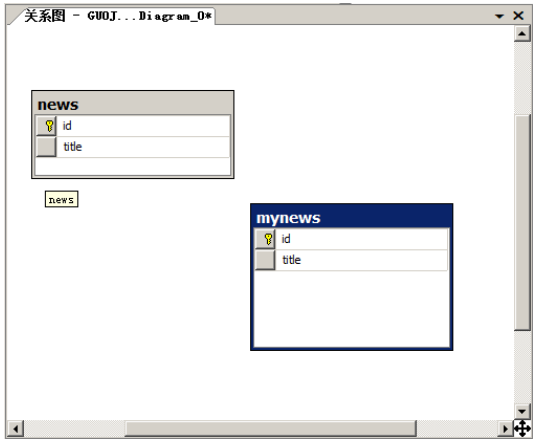


图 7-17 数据库关系图

在数据库关系图中，可以设置表与表之间的约束，可以拖动关系图中的字段，并建立关系，系统会自动建立表和列的关系，如图 7-18 所示。

选择了相应的列值之后，系统会提示填写表和列的规范，来规范外键的约束，建立约束，规范表与表之间的关系，如图 7-19 所示。

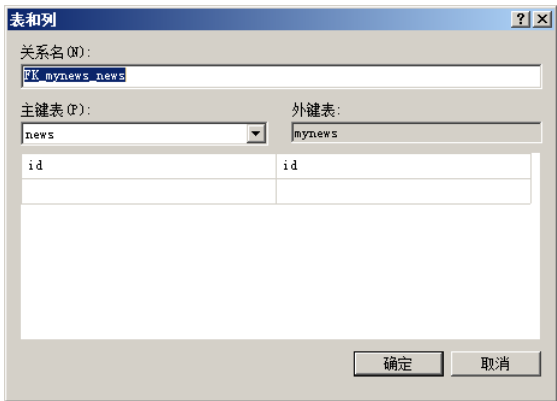


图 7-18 创建约束

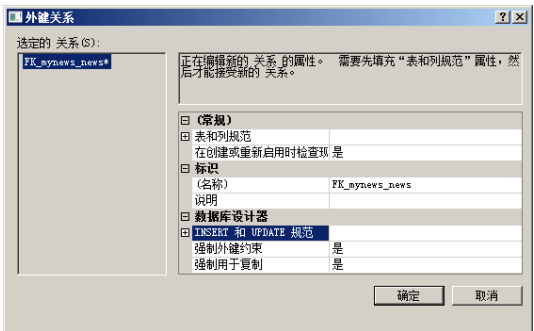


图 7-19 建立规范

单击【确定】按钮即可保存关系图。数据库关系图通过使用外键加强两个表数据之间的连接。外键（FOREIGN KEY）是用于建立和加强两个表数据之间的链接的一列或多列。通过将保存表中主键值的一列或多列添加到另一个表中，可创建两个表之间的链接，这个列就成为第二个表的外键。FOREIGN KEY 约束的主要目的是控制存储在外键表中的数据，但它还可以控制对主键表中数据的修改。

例如，当有一个学生管理系统，这个系统中有一个学生的学号为 20051183049，那么这个学生就会有一系列的数据，包括成绩、选修等。当这个学生毕业，注销该学生，或者这个学生经常不上课，要删除该学生的信息时，如果只删除这个学生的信息，而不删除这个学生的成绩、选修等信息就会造成其他的表中的数据的数据的完整性被破坏，在这时就需要使用外键约束即在删除学生信息时一同删除学生的其他信息以保证数据库的完整性。另外，通过数据库关系图，也可以良好的表达和操作表与表之间的关系。

7.3 ADO.NET 连接 SQL 数据库

ADO.NET 是 .NET Framework 中的一系列类库，它能够让开发人员更加方便的在应用程序中使用和操作数据。在 ADO.NET 中，大量的复杂的数据操作的代码被封装起来，所以当开发人员在 ASP.NET 应用程序开发中，只需要编写少量的代码即可处理大量的操作。ADO.NET 和 C#.NET、VB.NET 不同的是，ADO.NET 并不是一种语言，而是对象的集合。

7.3.1 ADO.NET 基础

ADO.NET 是由微软编写代码，提供了在 .NET 开发中数据库所需要的操作的类。在 .NET 应用程序开发中，C# 和 VB.NET 都可以使用 ADO.NET。

ADO.NET 可以被看作是一个介于数据源和数据使用者之间的转换器。ADO.NET 接受使用者语言中的命令，如连接数据库、返回数据集之类，然后将这些命令转换成在数据源中可以正确执行的语句。在传统的应用程序开发中，应用程序可以连接 ODBC 来访问数据库，虽然微软提供的类库非常的丰富，但是开发过程却并不简单。ADO.NET 在另一方面，可以说简化了这个过程。用户无需了解数据库产品的 API 或接口，也可以使用 ADO.NET 对数据进行了操作。ADO.NET 中常用的对象有：

- ❑ **SqlConnection**：该对象表示与数据库服务器进行连接。
- ❑ **SqlCommand**：该对象表示要执行的 SQL 命令。
- ❑ **SqlParameter**：该对象代表了一个将被命令中标记代替的值。
- ❑ **SqlDataAdapter**：该对象表示填充命令中的 DataSet 对象的能力。
- ❑ **DataSet**：表示命令的结果，可以是数据集，并且可以同 **BulletedList** 进行绑定。

通过使用上述的对象，可以轻松的连接数据库并对数据库中的数据进行操作。对开发人员而言，可以使用 ADO.NET 对数据库进行操作，在 ASP.NET 中，还提供了高效的控件，这些控件同样使用了 ADO.NET 让开发人员能够连接、绑定数据集并进行相应的数据操作。

7.3.2 连接 SQL 数据库

ADO.NET 通过 **ADOConnection** 连接到数据库。和 ADO 的 **Connection** 对象相似的是，**ADOConnection** 同样包括 **Open** 和 **Close** 方法。**Open** 表示打开数据库连接，**Close** 表示关闭数据库连接。在每次打开数据库连接后，都需要关闭数据库连接。

1. 建立连接

在 SQL 数据库的连接中，需要使用 .NET 提供的 **SqlConnection** 对象来对数据库进行连接。在连接数据库前，需要为连接设置连接串，连接串就相当于告诉应用程序怎样找到数据库去进行连接，然后程序才能正确的与 SQL 建立连接，连接字串示例代码如下所示。

```
server='服务器地址';database='数据库名称';uid='数据库用户名';pwd='数据库密码';
```

上述代码说明了数据库连接字串的基本格式，如果需要连接到本地的 mytable 数据库，则编写相应的 SQL 连接字串进行数据库的连接，示例代码如下所示。

```
string strcon; //声明连接字串
strcon = "server=(local);database=mytable;uid='sa';pwd='sa';"; //设置连接字串
```

上述代码声明了一个数据库连接字串，**SqlConnection** 类将会通过此字串来进行数据库的连接。其中，**server** 是 SQL 服务器的地址，如果相对于应用程序而言数据库服务器是本地服务器，则只需要配置为

(localhost)即可，而如果是远程服务器，则需要填写具体的 ip。另外，uid 是数据库登录时的用户名，pwd 是数据库登录时使用的密码。在声明了数据库连接字符串后，可以使用 SqlConnection 类进行连接，示例代码如下所示。

```
string strcon;                                //声明连接字符串
strcon = "server=(local);database=mytable;uid=sa;pwd=sa;"; //编写连接字符串
SqlConnection con = new SqlConnection(strcon); //新建 SQL 连接
try
{
    con.Open();                                //打开 SQL 连接
    Label1.Text = "连接数据库成功";           //提示成功信息
}
catch
{
    Label1.Text = "无法连接数据库";           //提示失败信息
}
```

上述代码连接了本地数据库服务器中的 mytable 数据库，如果连接成功，则提示“连接数据库成功”，出现异常时，则提示“无法连接数据库”。

注意：在使用 SqlConnection 类时，需要使用命名空间 using System.Data.SqlClient;而连接 Access 数据库时，需要使用命名空间 using System.Data.OleDb。

2. 填充 DataSet 数据集

DataSet 数据集表示来自一个或多个数据源数据的本地副本，是数据的集合，也可以看作是一个虚拟的表。DataSet 对象允许 Web 窗体半独立于数据源运行。DataSet 能够提高程序性能，因为 DataSet 从数据源中加载数据后，就会断开与数据源的连接，开发人员可以直接使用和处理这些数据，当数据发生变化并要更新时，则可以使用 DataAdapter 重新连接并更新数据源。DataAdapter 可以进行数据集的填充，创建 DataAdapter 对象的代码如下所示。

```
SqlDataAdapter da=new SqlDataAdapter("select * from news",con); //创建适配器
```

上述代码创建了一个 DataAdapter 对象并初始化 DataAdapter 对象，DataAdapter 对象的构造函数允许传递两个参数初始化，第一个参数为 SQL 查询语句，第二个参数为数据库连接的 SqlConnection 对象。初始化 DataAdapter 后，就需要将返回的数据的集合存放到数据集中，示例代码如下所示。

```
DataSet ds = new DataSet(); //创建数据集
da.Fill(ds, "tablename");    //Fill 方法填充
```

上述代码创建了一个 DataSet 对象并初始化 DataSet 对象，通过 DataAdapter 对象的 Fill 方法，可以将返回的数据存放到数据集 DataSet 中。DataSet 可以被看作是一个虚拟的表或表的集合，这个表的名称在 Fill 方法中被命名为 tablename。

3. 显式 DataSet

当返回的数据被存放到数据集中后，可以通过循环语句遍历和显示数据集中的信息。当需要显示表中某一行字段的值时，可以通过 DataSet 对象获取相应行的某一列的值，示例代码如下所示。

```
ds.Tables["tablename"].Rows[0]["title"].ToString(); //获取数据集
```

上述代码从 DataSet 对象中的虚表 tablename 中的第 0 行中获取 title 列的值，当需要遍历 DataSet 时，可以使用 DataSet 对象中的 Count 来获取行数，示例代码如下所示。

```
for (int i = 0; i < ds.Tables["tablename"].Rows.Count; i++) //遍历 DataSet 数据集
{
    Response.Write(ds.Tables["tablename"].Rows[i]["title"].ToString()+"<br/>");
}
```

DataSet 不仅可以通过编程的方法来实现显示,也可以使用 **ASP.NET** 中提供的控件来绑定数据集并显示。**ASP.NET** 中提供了常用的显示 **DataSet** 数据集的控件,包括 **Repeater**、**DataList**、**GridView** 等数据绑定控件。将 **DataSet** 数据集绑定到 **DataList** 控件中可以方便的在控件中显示数据库中的数据并实现分页操作,示例代码如下所示。

```
DataList1.DataSource = ds; //绑定数据集
DataList1.DataMember = "tablename";
DataList1.DataBind(); //绑定数据
```

上述代码就能够将数据集 **ds** 中的数据绑定到 **DataList** 控件中。**DataList** 控件还能够实现分页、自定义模板等操作,非常方便开发人员对数据开发。

7.3.3 ADO.NET 过程

从上一节中可以看出,在 **ADO.NET** 中对数据库的操作基本上需要三个步骤,即创建一个连接、执行命令对象并显式,最后再关闭连接。使用 **ADO.NET** 的对象,不仅能够通过控件绑定数据源,也可以通过程序实现数据源的访问。**ADO.NET** 的过程如图 7-20 所示。



图 7-20 ADO.NET 规范步骤

从上图中可以归纳出, **ADO.NET** 的规范步骤如下:

- ☐ 创建一个连接对象。
- ☐ 使用对象的 **Open** 方法打开连接。
- ☐ 创建一个封装 **SQL** 命令的对象。
- ☐ 调用执行命令的对象。
- ☐ 执行数据库操作。
- ☐ 执行完毕, 释放连接。

掌握了这些初步的知识, 就能够使用 **ADO.NET** 进行数据库开发。

7.4 ADO 与 ADO.NET

ADO.NET 相比于 **ADO** 有很大的改进。使用 **ADO.NET**, 能够更加容易的进行数据库的开发, 其中, 一部分是针对开发人员做出的更改, 包括易用性、适用性等, 其次的更改让 **ADO.NET** 相比于 **ADO**, 更加灵活、强大、易于升级使用。

7.4.1 ADO 概述

微软公司的 ADO (ActiveX Data Objects) 是一个用于存取数据源的 COM 组件。它提供了编程语言和统一数据访问方式 OLEDB 的一个中间层。允许开发人员编写访问数据的代码而不用关心数据库是如何实现的, 而只关心到数据库的连接。访问数据库的时候, 关于 SQL 的知识不是必要的, 但是特定数据库支持的 SQL 命令仍可以通过 ADO 中的命令对象来执行。

在开发过程中, ADO 为 OLEDB 数据提供给予 COM 的应用程序级别的接口, 对数据库的操作进行了封装, ADO 支持各种开发者的需要。同时, ADO 也能够像 ADO.NET 一样构建客户端记录集, 使用松耦合记录集, 并处理 OLEDB 的数据整形集合。

相比于 ADO.NET, ADO 还支持一些特殊的方法, 例如可滚动的服务器端游标 MOVENEXT。但是, 使用服务器游标需要使用和保存数据库资源, 所以当大量的游标在服务器端被使用时, 则可能对应用程序的性能和可缩放性产生极大的负面影响。使用 ADO, 还需要对防火墙进行配置以启用 COM 的发送请求才能够进行数据交互, 这样可能造成一定的安全问题。ADO 编程模型如下所示:

- ☐ 连接数据源(Connection), 可选择开始事务。
- ☐ 可选择创建表示 SQL 命令的对象(Command)。
- ☐ 可选择指定列、表, 以及 SQL 命令中的值作为变量参数(Parameter)。
- ☐ 执行命令 (Command、Connection 或 Recordset)。
- ☐ 如果命令以行返回, 将行存储在存储对象中(Recordset)。
- ☐ 可选择创建存储对象的视图以便进行排序、筛选和定位数据(Recordset)。
- ☐ 编辑数据。可以添加、删除或更改行、列(Recordset)。
- ☐ 在适当情况下, 可以使用存储对象中的变更对数据源进行更新(Recordset)。
- ☐ 在使用事务之后, 可以接受或拒绝在事务中所做的更改。结束事务(Connection)。

从上述的编程模型可以看出, ADO.NET 在很多方面和 ADO 比较相近, 但是 ADO.NET 并不是 ADO 的 .NET 版本, ADO 和 ADO.NET 是两种不同的数据访问方式。

7.4.2 ADO.NET 与 ADO

ADO.NET 的名称起源于 ADO (ActiveX Data Objects), ADO 用于在以往的 Microsoft 技术中进行数据的访问。所以微软希望通过使用 ADO.NET 名称来向开发人员表明, 这是在 .NET 编程环境和 Windows 环境中优先使用的数据访问接口。

ADO.NET 提供了平台互用性和可伸缩的数据访问, ADO.NET 增强了对非连接编程模式的支持, 并支持 RICH XML。由于传送的数据都是 XML 格式的, 因此任何能够读取 XML 格式的应用程序都可以进行数据处理。事实上, 接受数据的组件不一定是 ADO.NET 组件, 它可以是基于一个 Microsoft Visual Studio 的解决方案, 也可以是任何运行在其他平台上的任何应用程序。

可以说传统的 ADO 和 ADO.NET 是两种不同的数据访问方式, 无论是在内存中保存数据, 还是打开和关闭数据库的操作模式都不尽相同。

1. 数据在内存中的存在形式

使用 ADO.NET 时, 数据保存在内存中并且是以 DataSet 数据集的形式存放在内存中, 而 ADO 中, 则是以 RecordSet 记录集的形式存放在内存中。

在 ASP 的开发过程中, 连接数据库后, 通常将查询操作的数据的集合保存在记录集中, 并使用 MOVENEXT 等方法进行遍历, 也可以使用其他的方法进行查询, 提取任意行。而在 ASP.NET 中,

ADO.NET 提供的 `DataRelation` 对象维护有关主记录和详细资料记录的信息，并提供方法使用户可以获取与正在操作的记录相关的记录。

2. 数据的表现形式

在 ADO 中，记录集的表现形式像一个表。如果需要包含来自多个数据库的表的数据，就必须使用复杂的 SQL 语句中的 JOIN 查询将各个数据库的表的数据组合到单个记录集中。

而在 ADO.NET 中，数据集本身是一个表或多个表的集合。相对于记录集而言，数据集可以保存多个独立的表并维护有关表之间关系的信息。因此，ADO.NET 能够维护和保存数据结构复杂的表，模仿数据库的结构，例如表的自关联，以及有一对多或多对多的关系表。

3. 数据的连接和断开

ADO 是为连接的访问而设计的，相比之下，ADO.NET 打开连接的时间仅仅足够数据库的操作。数据集可以将行读入，然后断开与数据库的连接，再对数据集中记录进行更改。当需要将数据集中的资源更新到数据库时，ADO.NET 再与数据库连接并更新。

4. 数据共享

再 ADO 中，如果需要实现 ADO.NET 中断开数据连接传送数据的功能，则必须在 COM 组件之间互相传送，这样就造成了安全性问题。在杀毒软件中，一些杀毒软件可能会默认禁止 COM 组件之间的通信，这样就造成了开发人员的维护困难。而 ADO.NET 能够使用 `DataSet` 传送数据而不需要考虑防火墙的限制，是因为 `DataSet` 传送的数据集会被转换成 XML 流来传送。ADO.NET 相对于 ADO 再数据共享上，有如下优点：

- ❑ 突破 COM 数据类型的限制：由于 ADO.NET 基于 XML 流传送数据，所以对数据类型没有限制。
- ❑ 减少数据类型的转换：ADO.NET 相对于 ADO 而言，减少了大量的数据类型的转换，提高了性能。
- ❑ 可以穿透防火墙：基于 XML 流传送数据的方法能够轻松穿透防火墙。

5. 构架设计

在构架设计上 ADO.NET 与 ADO 也是不同的，ADO.NET 相对于 ADO 更加方便和简洁，从设计的角度来说，ADO.NET 设计的更加完善。

在 ADO 中，通过使用 `ADORecordSet` 对象进行数据的连接和操作，`ADORecordSet` 对象是一个庞大的对象，它提供了多种类型的游标能力，例如快速的即时的游标到无连接的客户端游标。但是使用 `ADORecordSet` 对象，很难对数据的操作的方法进行自定义。而在 ADO.NET 中，ADO.NET 将 ADO 中 `ADORecordSet` 对象的方法进行了一个拆分，将其中的若干功能分成多个类，通过类之间的调用来实现。这样就方便了开发人员自定义数据的连接和操作。

注意：在应用程序中，ADO 与 ADO.NET 是可以共存的，因为在 .NET 中同样可以使用 COM 互操作服务使用 ADO。

7.5 ADO.NET 常用对象

ADO.NET 提供了一些常用对象来方便开发人员进行数据库的操作，这些常用的对象通常会使用在应用程序开发中，对于中级的开发人员而言，熟练的掌握这些常用的 ADO.NET 对象，能够自行封装数据库操作类，来简化开发。ADO.NET 的常用对象包括：

- ❑ `Connection` 对象。

- ❑ DataAdapter 对象。
- ❑ Command 对象。
- ❑ DataSet 对象。
- ❑ DataReader 对象。

上面的对象在.NET 应用程序操作数据中是非常重要的，它们不仅提供了数据操作的便利，同时，还提供了高级的功能给开发人员。为开发人员解决特定的需求。

7.6 Connection 连接对象

在.NET 开发中，通常情况下开发人员被推荐使用 Access 或者 SQL 作为数据源，若需要连接 Access 数据库，可以使用 System.Data.OleDb.OleDbConnection 对象来连接；若需要连接 SQL 数据库，则可以使用 System.Data.SqlClient.SqlConnection 对象来连接。使用 System.Data.Odbc.OdbcConnection 可以连接 ODBC 数据源，而 System.Data.OracleClient.OracleConnecton 提供了连接 Oracle 的一些方法。本章主要讨论连接 Access 和 SQL 数据库。

7.6.1 连接 SQL 数据库

如需要连接 SQL 数据库，则需要使用命名空间 System.Data.SqlClient 和 System.Data.OleDb。使用 System.Data.SqlClient 和 System.Data.OleDb 能够快速连接 SQL 数据库，因为 System.Data.SqlClient 和 System.Data.OleDb 都分别为开发人员提供了连接方法，示例代码如下所示。

```
using System.Data.SqlClient;           //使用 SQL 命名空间
using System.Data.OleDb                //使用 OleDb 命名空间
```

1. 使用 System.Data.SqlClient

连接 SQL 数据库，则需要创建 SqlConnection 对象，SqlConnection 对象创建代码如下所示。

```
SqlConnection con = new SqlConnection();           //创建连接对象
con.ConnectionString = "server=(local);database=mytable;uid=sa'pwd=sa"; //设置连接字符串
```

上述代码创建了一个 SqlConnection 对象，并且配置了连接字符串。SqlConnection 对象专门定义了一个专门接受连接字符串的变量 ConnectionString，当配置了 ConnectionString 变量后，就可以使用 Open() 方法来打开数据库连接，示例代码如下所示。

```
SqlConnection con = new SqlConnection();           //创建连接对象
con.ConnectionString = "server=(local);database=mytable;uid=sa'pwd=sa";
try
{
    con.Open();                                     //尝试打开连接
    Label1.Text = "连接成功";                       //提示打开成功
    con.Close();                                    //关闭连接
}
catch
{
    Label1.Text = "连接失败";                       //提示打开失败
}
```

上述代码尝试判断是否数据库连接被打开，使用 Open 方法能够建立应用程序与数据库之间的连接。与之相同的是，可以使用默认的构造函数来对数据库连接对象进行初始化，示例代码如下所示。

```
string str = "server=(local);database=mytable;uid=sa;pwd=Sa"; //设置连接字符串
SqlConnection con = new SqlConnection(str); //默认构造函数
```

上述代码与使用 `ConnectionString` 变量的方法等价，其默认的构造函数中已经为 `ConnectionString` 变量进行了初始化。

2. 使用 `System.Data.OleDb`

ADO.NET 中，具有相同功能的函数一般具有相同的参数和字段以及方法。所以，在 .NET 开发中，开发人员能够很快的适应新的操作。同样 `System.Data.OleDb` 也提供了 `Open` 方法以及 `ConnectionString` 字段，示例代码如下所示。

```
OleDbConnection con = new OleDbConnection(); //创建连接对象
con.ConnectionString = "Provider=SQLOLEDB;Data Source=(local);Initial Catalog=mytable;uid=sa;pwd=sa"; //初始化连接字符串
try
{
    con.Open(); //尝试打开连接
    Label1.Text = "连接成功"; //提示连接成功
    con.Close(); //关闭连接
}
catch
{
    Label1.Text = "连接失败"; //提示连接失败
}
```

同样，`OleDbConnection` 也提供默认的构造函数来初始化连接变量，示例代码如下所示。

```
string str =
"Provider=SQLOLEDB;Data Source=(local);Initial Catalog=mytable;uid=sa;pwd=sa";
OleDbConnection con = new OleDbConnection(str);
```

上述代码通过使用构造函数初始化连接变量进行相应的 `ConnectionString` 变量的配置。值得注意的是，从上面代码可以看出，连接字符串一般都通过使用用户名和密码的形式连接，这样保证了连接的安全性。另外，连接字符串还可以使用 `Trusted_Connection=Yes` 来声明这是一个值得信任的连接字符串，而不需要输入用户名和密码，示例代码如下所示。

```
string str2 =
"Provider=SQLOLEDB;Data Source=(local);Initial Catalog=mytable;Trusted_Connection=Yes";
OleDbConnection con = new OleDbConnection(str2);
```

7.6.2 连接 Access 数据库

Access 是一种桌面级数据库，虽然与 SQL 相比，Access 数据库的性能和功能都不强大，但是 Access 却是最常用的数据库之一。对于小型应用和小型企业来说，Access 数据库也是开发中小型软件的最佳选择。

1. 创建 Access 数据库

Access 是 Office 组件之一，当安装了 Office 后，就可以新建 Access 数据库，在桌面或任何文件夹中单击右键就能够创建 Access 数据库。创建完成后，双击数据库文件就能够打开数据库并建立表和字段，如图 7-21 所示。

同样，Access 数据库也需要创建表和字段，基本方法与 SQL 数据库相同，但是在数据类型上，自动增长编号作为单独的数据类型而存在。开发人员能够在表窗口中创建表 `mytable` 和相应字段，如图 7-22

所示。

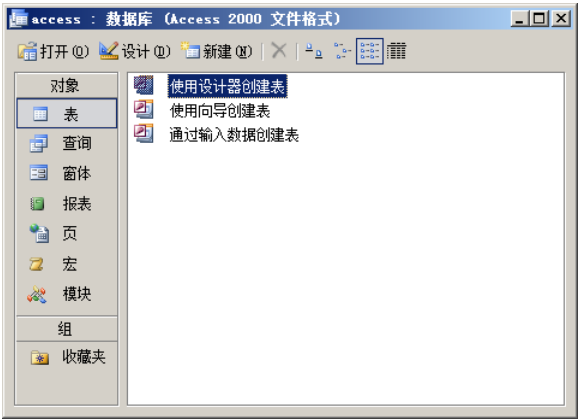


图 7-21 创建 Access 数据库

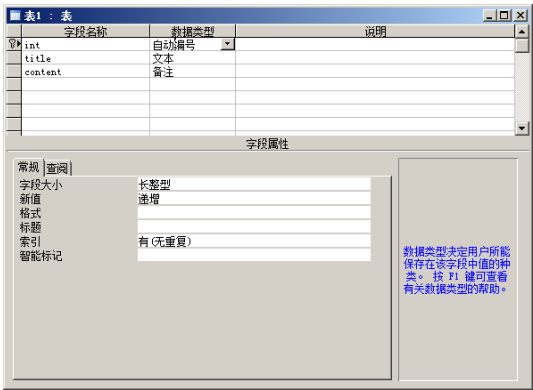


图 7-22 创建 Access 数据库的表

创建完成后可以使用 System.Data.OleDb 的对象进行数据库的连接和数据操作。

注意：Access 数据库是一个桌面级的数据库，其数据都会存放在一个文件中而不是存放在数据库服务器中。

2. 使用 System.Data.OleDb

在使用 System.Data.OleDb 时，只需要修改连接字符串即可。在这里需要强调一点的是，Access 数据库是一种桌面级的数据库，同文件类型的数据库类似，所以连接 Access 数据库时，必须指定数据库文件的路径，或者使用 Server.MapPath 来确定数据库文件的相对位置。示例代码如下所示。

```
string str = "provider=Microsoft.Jet.OLEDB.4.0 ;Data Source="
+ Server.MapPath("access.mdb") + "; //使用相对路径
OleDbConnection con = new OleDbConnection(str); //构造连接对象
try
{
    con.Open(); //打开连接
    Label1.Text = "连接成功"; //提示连接成功
    con.Close();
}
catch(Exception ee) //抛出异常
{
    Label1.Text = "连接失败";
}
```

Server.MapPath 能够确定文件相对于当前目录的路径，如果不使用 Server.MapPath，则需要指定文件在计算机的路径，如“D:\服务器\文件夹\数据库路径”。但是这样会暴露数据库的物理路径，让程序长期处于不安全的状态。

7.6.3 打开和关闭连接

无论是使用 System.Data.SqlClient 还是 System.Data.OleDb 创建数据库连接对象，都可以使用 Open 方法来打开连接。同样，也可以使用 Close 方法来关闭连接，示例代码如下所示。

```
SqlConnection con = new SqlConnection(str); //创建连接对象
OleDbConnection con2 = new OleDbConnection(str2); //创建连接对象
```

con.Open();	//打开连接
con.Close();	//关闭连接
con2.Open();	//打开连接
con2.Close();	//关闭连接

注意：如果使用了连接池，虽然显式的关闭了连接对象，其实并不会真正的关闭与数据库之间的连接，这样能够保证再次进行连接时的连接性能。

7.7 DataAdapter 适配器对象

在创建了数据库连接后，就需要对数据集 DataSet 进行填充，在这里就需要使用 DataAdapter 对象。在没有数据源时，DataSet 对象对保存在 Web 窗体可访问的本地数据库是非常实用的，这样降低了应用程序和数据库之间的通信次数。然而 DataSet 必须要与一个或多个数据源进行交互，DataAdapter 就提供 DataSet 对象和数据源之间的连接。

为了实现这种交互，微软提供了 SqlDataAdapter 类和 OleDbDataAdapter 类。SqlDataAdapter 类和 OleDbDataAdapter 类各自适用情况如下。

- ❑ **SqlDataAdapter**：该类专用于 SQL 数据库，在 SQL 数据库中使用该类能够提高性能，SqlDataAdapter 与 OleDbDataAdapter 相比，无需适用 OLEDB 提供程序层，可直接在 SQL Server 上使用。
- ❑ **OleDbDataAdapter**：该类适用于由 OLEDB 数据提供程序公开的任何数据源，包括 SQL 数据库和 Access 数据库。

若要使一个使用 DataAdapter 对象的 DataSet 要能够和一个数据源之间交换数据，则可以使用 DataAdapter 属性来指定需要执行的操作，这个属性可以是一条 SQL 语句或者是存储过程，示例代码如下所示。

string str = "server=(local);database=mytable;uid='sa';pwd='sa'";	//创建连接字符串
SqlConnection con = new SqlConnection(str);	
con.Open();	//打开连接
SqlDataAdapter da = new SqlDataAdapter("select * from news", con);	//DataAdapter 对象
con.Close();	//关闭连接

上述代码创建了一个 DataAdapter 对象，DataSet 对象可以使用该对象的 Fill 方法来填充数据集。

7.8 Command 执行对象

Command 对象可以使用数据命令直接与数据源进行通信。例如，当需要执行一条插入语句，或者删除数据库中的某条数据的时候，就需要使用到 Command 对象。Command 对象的属性包括了数据库在执行某个语句的所有必要的信息，这些信息如下所示：

- ❑ **Name**：Command 的程序化名称。
- ❑ **Connection**：对 Connection 对象的引用。
- ❑ **CommandType**：指定是使用 SQL 语句或存储过程，默认情况下是 SQL 语句。
- ❑ **CommandText**：命令对象包含的 SQL 语句或存储过程名。
- ❑ **Parameters**：命令对象的参数。

通常情况下，Command 对象用于数据的操作，例如执行数据的插入和删除，也可以执行数据库结

构的更改，包括表和数据库。示例代码如下所示。

```
string str = "server=(local);database=mytable;uid='sa';pwd='sa'";           //创建数据库连接字符串
SqlConnection con = new SqlConnection(str);
con.Open();                                                                //打开数据库连接
SqlCommand cmd = new SqlCommand("insert into news values ('title')",con); //建立 Command 对象
```

上述代码使用了可用的构造函数并指定了查询字符串和 `Connection` 对象来初始化 `Command` 对象 `cmd`。通过指定 `Command` 对象的方法可以对数据执行具体的操作。

7.8.1 ExecuteNonQuery 方法

当指定了一个 SQL 语句，就可以通过 `ExecuteNonQuery` 方法来执行语句的操作。`ExecuteNonQuery` 不仅可以执行 SQL 语句，开发人员也可以执行存储过程或数据定义语言语句来对数据库或目录执行构架操作。而使用 `ExecuteNonQuery` 时，`ExecuteNonQuery` 并不返回行，但是可以通过 `Command` 对象和 `Parameters` 进行参数传递。示例代码如下所示。

```
string str = "server=(local);database=mytable;uid='sa';pwd='sa'";           //创建数据库连接字符串
SqlConnection con = new SqlConnection(str);
con.Open();
SqlCommand cmd = new SqlCommand("insert into news values ('title')",con);
cmd.ExecuteNonQuery();                                                     //执行 SQL 语句
```

运行上述代码后，会执行 “insert into news values (‘title’)” 这条 SQL 语句并向数据库中插入数据。值得注意的是，修改数据库的 SQL 语句，例如常用的 `INSERT`、`UPDATE` 以及 `DELETE` 并不返回行。同样，很多存储过程同样不返回任何行。当执行这些不返回任何行的语句或存储过程时，可以使用 `ExecuteNonQuery`。但是 `ExecuteNonQuery` 语句也会返回一个整数，表示受已执行的 SQL 语句或存储过程影响的行数，示例代码如下所示。

```
string str = "server=(local);database=mytable;uid='sa';pwd='sa'";           //创建连接对象
SqlConnection con = new SqlConnection(str);
con.Open();                                                                //打开连接
SqlCommand cmd = new SqlCommand("delete from mynews", con);                //构造 Command 对象
Response.Write("该操作影响了("+cmd.ExecuteNonQuery().ToString()+") 行");//执行 SQL 语句
```

上述代码执行了语句 “delete from mynews” 并将影响的行数输出到字符串中。开发人员能够使用 `ExecuteNonQuery` 语句进行数据库操作和数据库操作所影响行数的统计。

7.8.2 ExecuteNonQuery 执行存储过程

`ExecuteNonQuery` 不仅能够执行 SQL 语句，同样可以执行存储过程和数据定义语言来对数据库或目录执行构架操作如 `CREATE TABLE` 等。在执行存储过程之前，必须先创建一个存储过程，然后在 `SqlCommand` 方法中使用存储过程。在 SQL Server 管理器中可以新建查询创建存储过程，示例代码如下所示。

```
CREATE PROC getdetail
(
    @id int,
    @title varchar(50) OUTPUT
)
AS
SET NOCOUNT ON
```

```

DECLARE @newscount int
SELECT @title=mynews.title,@newscount=COUNT(mynews.id)
FROM mynews
WHERE (id=@id)
GROUP BY mynews.title
RETURN @newscount

```

上述存储过程返回了数据库中新闻的标题内容。“@id”表示新闻的 id，“@title”表示新闻的标题，此存储过程将返回“@title”的值，并且返回新闻的总数。上述代码可以直接在 SQL 管理器中菜单栏中单击【新建查询】后创建的 TAB 中使用，同样也可以使用 SqlCommand 对象进行存储过程的创建，示例代码如下所示。

```

string str = "CREATE PROC getdetail" +
"(" +
"@id int," +
"@title varchar(50) OUTPUT" +
")" +
"AS" +
"SET NOCOUNT ON" +
"DECLARE @newscount int" +
"SELECT @title=mynews.title,@newscount=COUNT(mynews.id)" +
"FROM mynews" +
"WHERE (id=@id)" +
"GROUP BY mynews.title" +
"RETURN @newscount";
SqlCommand cmd = new SqlCommand(str, con);
cmd.ExecuteNonQuery();           //使用 cmd 的 ExecuteNonQuery 方法创建存储过程

```

创建存储过程后，就可以使用 SqlParameter 调用命令对象 Parameters 参数的集合的 Add 方法进行参数传递，并指定相应的参数，示例代码如下所示。

```

string str = "server='(local)';database='mytable';uid='sa';pwd='Sa'";
SqlConnection con = new SqlConnection(str);
con.Open();                               //打开连接
SqlCommand cmd = new SqlCommand("getdetail", con);           //使用存储过程
cmd.CommandType = CommandType.StoredProcedure;           //设置 Command 对象的类型
SqlParameter spr;                                           //表示执行一个存储过程
spr = cmd.Parameters.Add("@id", SqlDbType.Int);           //增加参数 id
spr = cmd.Parameters.Add("@title", SqlDbType.NChar,50);           //增加参数 title
spr.Direction = ParameterDirection.Output;           //该参数是输出参数
spr = cmd.Parameters.Add("@count", SqlDbType.Int);           //增加 count 参数
spr.Direction = ParameterDirection.ReturnValue;           //该参数是返回值
cmd.Parameters["@id"].Value = 1;           //为参数初始化
cmd.Parameters["@title"].Value = null;           //为参数初始化
cmd.ExecuteNonQuery();           //执行存储过程
Label1.Text = cmd.Parameters["@count"].Value.ToString();           //获取返回值

```

上述代码使用了现有的存储过程，并为存储过程传递了参数，当参数被存储过程接受并运行后，会返回一个存储过程中指定的返回值。当执行完毕后，开发人员可以通过 cmd.Parameters 来获取其中一个变量的值。

7.8.3 ExecuteScalar 方法

Command 的 Execute 方法提供了返回单个值的功能。在很多时候，开发人员需要获取刚刚插入的数据的 ID 值，或者可能需要返回 Count(*), Sum(Money)等聚合函数的结果，则可以使用 ExecuteScalar 方法。示例代码如下所示。

```
string str = "server=(local);database=mytable;uid='sa';pwd='sa'"; //设置连接字符串
SqlConnection con = new SqlConnection(str); //创建连接
con.Open(); //打开连接
SqlCommand cmd = new SqlCommand("select count(*) from mynews", con); //创建 Command
Label1.Text = cmd.ExecuteScalar().ToString(); //使用 ExecuteScalar 执行
```

上述代码创建了一个连接，并创建了一个 Command 对象，使用了可用的构造函数来初始化对象。当使用 ExecuteScalar 执行方法时，会返回单个值。ExecuteScalar 方法同样可以执行 SQL 语句，但是与 ExecuteNonQuery 方法不同的是，当语句不为 SELECT 时，则返回一个没有任何数据的 System.Data.SqlClient.SqlDataReader 类型的集合。

ExecuteScalar 方法执行 SQL 语句通常是用来执行具有返回值的功能的 SQL 语句，例如上面所说的当插入一条新数据时，返回刚刚插入的数值的 ID 号。这种功能在自动增长类型的数据库设计中，经常被使用到，示例代码如下所示。

```
string str = "server=(local);database=mytable;uid='sa';pwd='sa'"; //设置连接字符串
SqlConnection con = new SqlConnection(str); //创建连接
con.Open(); //打开连接
SqlCommand cmd = new SqlCommand("insert into mynews values ('this is a new title!')
SELECT @@IDENTITY as 'bh'", con); //打开连接
Label2.Text = cmd.ExecuteScalar().ToString(); //获取返回的 ID 值
```

上述代码使用了“SELECT @@IDENTITY as”语法，“SELECT @@IDENTITY”语法会自动获取刚刚插入的自动增长类型的值，例如，当表中有 100 条数据时，插入一条数据后数据量就成 101，为了不需要再次查询就获得 101 这个值，则可以使用“SELECT @@IDENTITY as”语法。运行结果如图 7-23 所示。

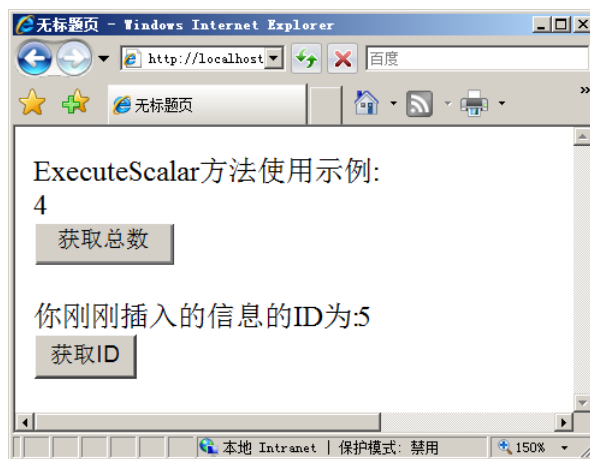


图 7-23 ExecuteScalar 方法示例

当使用了“SELECT @@IDENTITY as”语法进行数据操作时，ExecuteScalar 方法会返回刚刚插入的数据的 ID，这样就无需再次查询进行刚刚插入的数据的信息的获取。

7.9 DataSet 数据集对象

DataSet 是 ADO.NET 中的核心概念，作为初学者，可以把 DataSet 想象成虚拟的表，但是这个表不能用简单的表来表示，这个表可以想象成具有数据库结构的表，并且这个表是存放在内存中的。由于 ADO.NET 中 DataSet 的存在，开发人员能够屏蔽数据库与数据库之间的差异，从而获得一致的编程模型。

7.9.1 DataSet 数据集基本对象

DataSet 能够支持多表、表间关系、数据库约束等，可以模拟一个简单的数据库模型。DataSet 对象模型如图 7-24 所示。

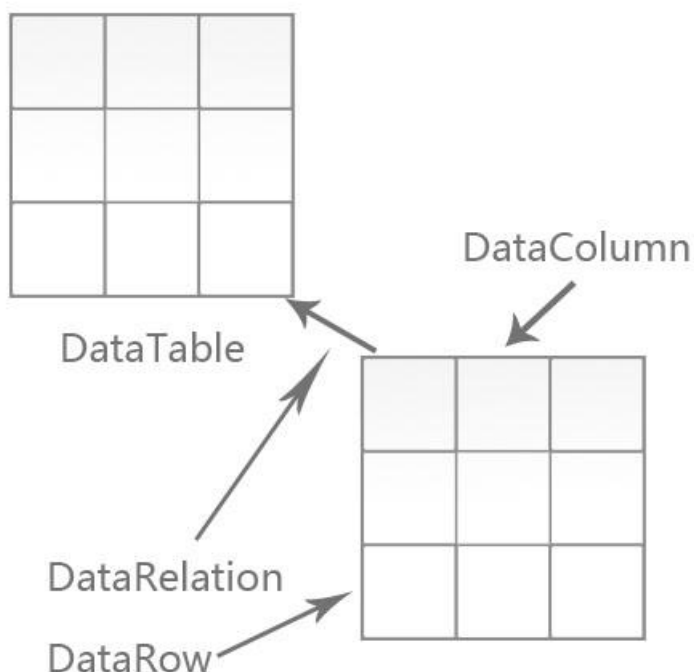


图 7-24 DataSet 对象模型

上图简要的介绍了常用对象之间的构架关系。在 DataSet 中，主要包括 TablesCollection、RelationsCollection、ExtendedProperties 几个重要对象：

1. TablesCollection 对象

在 DataSet 中，表的概念是用 DataTable 来表示的。DataTable 在 System.Data 中定义，它能够表示存储在内从中的一张表。它包含一个 ColumnsCollection 的对象，代表数据表的各个列的定义。同时，它也包含 RowsCollection 对象，这个对象包含 DataTable 中的所有数据。

2. RelationsCollection 对象

在各个 DataTable 对象之间，是通过使用 RelationsCollection 来表达各个 DataTable 对象之间的关系。RelationsCollection 对象可以模拟数据库中的约束的关系。例如当一个包含外键的表被更新时，如果不满足主键-外键约束，这个更新操作就会失败，系统会抛出异常。

3. ExtendedProperties 对象

ExtendedProperties 对象能够配置特定的信息，例如 DataTable 的密码，更新时间等等。

7.9.2 DataTable 数据表对象

DataTable 是 DataSet 中的常用的对象，它和数据库中的表的概念十分相似。开发人员能够将 DataTable 想象成一个表。并且可以通过编程的方式创建一个 DataTable 表。示例代码如下所示。

```
DataTable Table = new DataTable("mytable");           //创建一个 DataTable 对象
Table.CaseSensitive = false;                         //设置不区分大小写
Table.MinimumCapacity = 100;                         //设置 DataTable 初始大小
Table.TableName = "newtable";                       //设置 DataTable 的名称
```

上述代码创建了一个 DataTable 对象，并为 DataTable 对象设置了若干属性，这些属性都是常用的属性，其作用分别如下所示。

- ❑ CaseSensitive: 此属性设置表中的字符串是否区分大小写，若无特殊情况，一般设置为 false，该属性对于查找，排序，过滤等操作有很大的影响。
- ❑ MinimumCapacity: 设置创建的数据表的最小的记录空间。
- ❑ TableName: 指定数据表的名称。

一个表必须有一个列，而 DataTable 必须包含列。当创建了一个 DataTable 后，就必须向 DataTable 中增加列的。表中列的集合形成了二维表的数据结构。开发人员可以使用 Columns 集合的 Add 方法向 DataTable 中增加列，Add 方法带有两个参数，一个是表列的名称，一个是该列的数据类型。示例代码如下所示。

```
DataTable Table = new DataTable("mytable");           //创建一个 DataTable
Table.CaseSensitive = false;                         //设置不区分大小写
Table.MinimumCapacity = 100;                         //设置 DataTable 初始大小
Table.TableName = "newtable";                       //设置 DataTable 的名称
DataColumn Column = new DataColumn();                //创建一个 DataColumn
Column = Table.Columns.Add("id", typeof(int));       //增加一个列
Column = Table.Columns.Add("title", typeof(string)); //增加一个列
```

上述代码创建了一个 DataTable 和一个 DataColumn 对象，并通过 DataTable 的 Columns.Add 方法增加 DataTable 的列，这两列的列名和数据类型如下：

- ❑ 新闻 ID: 整型，用于描述新闻的编号。
- ❑ 新闻标题 TITLE: 字符型，用于描述新闻发布的标题。

注意：上述代码中，DataTable 的列的数据类型使用的只能是 .net 中数据类型，因为其并不是真实的数据库，所以不能直接使用数据库类型，必须使用 typeof 方法把 .net 中的数据类型转换成数据库类型。

7.9.3 DataRow 数据行对象

在创建了表和表中列的集合，并使用约束定义表的结构后，可以使用 DataRow 对象向表中添加新的数据库行，这一操作同数据库中的 INSERT 语句的概念类似。插入一个新行，首先要声明一个 DataRow 类型的变量。使用 NewRow 方法能够返回一个新的 DataRow 对象。DataTable 会根据 DataColumnCollection 定义的表的结构来创建 DataRow 对象。示例代码如下所示。

```
DataRow Row = Table.NewRow(); //使用 DataTable 的 NewRow 方法创建一个新 DataRow 对象
```

上述代码使用 DataTable 的 NewRow 方法创建一个新 DataRow 对象，当使用该对象添加了新行之后，必须使用索引或者列名来操作新行，示例代码如下所示。

```
Row[0] = 1; //使用索引赋值列
Row[1] = "datarow"; //使用索引赋值列
```

上述代码通过索引来为一行中个各列赋值。从数组的语法可以知道，索引都是从第 0 个位置开始。将 DataTable 想象成一个表，从左到右从 0 开始索引，直到数值等于列数减 1 为止。为了提高代码的可读性，也可以通过直接使用列名来添加新行，示例代码如下所示。

```
Row["bh"] = 1; //使用列名赋值列
Row["title"] = "datarow"; //使用列名赋值列
```

通过直接使用列名来添加新行与使用索引添加新行的效果相同，但是通过使用列名能够让代码更加可读，便于理解，但是也暴露了一些机密内容（如列值）。在数据插入到新行后，使用 Add 方法将该行添加到 DataRowCollection 中，示例代码如下所示。

```
Table.Rows.Add(Row); //增加列
```

7.9.4 DataView 数据视图对象

当需要显示 DataRow 对象中的数据时，可以使用 DataView 对象来显示 DataSet 中的数据。在显示 DataSet 中的数据之前，需要将 DataTable 中的数据填充到 DataSet。值得注意的是，DataSet 是 DataTable 的集合，可以使用 DataSet 的 Add 方法将多个 DataTable 填充到 DataSet 中去，示例代码如下所示。

```
DataSet ds = new DataSet(); //创建数据集
ds.Tables.Add(Table); //增加表
```

填充完成后，可以通过 DataView 对象来显示 DataSet 数据集中的内容，示例代码如下所示。

```
dv = ds.Tables["newtable"].DefaultView; //设置默认视图
```

DataSet 对象中的每个 DataTable 对象都有一个 DefaultView 属性，该属性返回表的默认视图。上述代码访问了名为 newtable 表的 DataTable 对象。开发人员能够自定义 DataView 对象，该对象能够筛选表达式来设置 DataView 对象的 RowFilter 属性，筛选表达式的值必须为布尔值。同时，该对象能够设置 Sort 属性进行排序，排序表达式可以包括列名或一个算式，示例代码如下所示。

```
DataView dv = new DataView(); //创建数据视图对象
DataSet ds = new DataSet(); //创建数据集
ds.Tables.Add(Table); //增加数据表
dv = ds.Tables["newtable"].DefaultView; //设置默认视图
dv.RowFilter = "id" = "1"; //设置筛选表达式
dv.Sort = "id"; //设置排序表达式
```

技巧：要显示 DataSet 中某项的值，可以使用语法 ds.Tables[“表名称”].Rows[0][“列名称”].ToString() 来显示，这种语法通常需要知道行的数目以免在访问数据时越界。

7.10 DataReader 数据访问对象

DataSet 的最大好处在于，能够提供无连接的数据库副本，DataSet 对象在表的生命周期内会为这些表进行内存的分配和维护。如果有多个用户同时对一台计算机进行操作，内存的使用就会变得非常的紧张。当对数据所需要进行一些简单的操作时，就无需保持 DataSet 对象的生命周期，可以使用 DataReader 对象。

7.10.1 DataReader 对象概述

当使用 DataReader 对象时，不会像 DataSet 那样提供无连接的数据库副本。DataReader 类被设计为产生只读，只进的数据流。这些数据流都是从数据库返回的。所以，每次的访问或操作只有一个记录保存在服务器的内存中。相比与 DataSet 而言，DataReader 具有较快的访问能力，并且能够使用较少的服务器资源，DataReader 具有以下快速的数据库访问、只进和只读、减少服务器资源等特色。

1. 快速的数据库访问

DataReader 类是轻量级的，相比之下 DataReader 对象的速度要比 DataSet 要快。因为 DataSet 在创建和初始化时，可能是一个或多个表的集合，并且 DataSet 具有向前，向后读写和浏览的能力，所以当创建一个 DataSet 对象时，会造成额外的开销。

2. 只进和只读

当对数据库的操作没有太大的要求时，可以使用 DataReader 显示数据。这些数据可以与单个 list-bound 控件绑定，也可以填充 List 接口。当不需要复杂的数据库处理时，DataReader 能够较快的完成数据显示。

3. 减少服务器资源

因为 DataReader 并不是数据的内存的表示形式，所以使用 DataReader 对服务器占用的资源很少。

4. 自定义数据库管理

DataReader 对象可以使用 Read 方法来进行数据库遍历，当使用 Read 方法时，可以以编程的方式自定义数据库中数据的显示方式，当开发自定义控件时，可以将这些数据整合到 HTML 中，并显示数据。

5. 手动连接管理

DataAdapter 对象能够自动的打开和关闭连接，而 DataReader 对象需要用户手动的管理连接。DataReader 对象和 DataAdapter 对象很相似，都可以从 SQL 语句和一个连接中初始化。

7.10.2 DataReader 读取数据库

创建 DataReader 对象，需要创建一个 SqlCommand 对象来代替 SqlDataAdapter 对象。与 SqlDataAdapter 对象类似的是，DataReader 可以从 SQL 语句和连接中创建 Command 对象。创建对象后，必须显式的打开 Connection 对象。示例代码如下所示。

```
string str = "server=(local);database=mytable;uid='sa';pwd='sa'";
SqlConnection con = new SqlConnection(str);
con.Open();                                     //打开连接
SqlCommand cmd = new SqlCommand("select * from mynews", con); //创建 Command 对象
SqlDataReader dr = cmd.ExecuteReader();          //创建 DataReader 对象
con.Close();
```

上述代码创建了一个 DataReader 对象，从上述代码中可以看出，创建 DataReader 对象必须经过如下几个步骤：

- ☐ 创建和打开数据库连接。
- ☐ 创建一个 Command 对象。
- ☐ 从 Command 对象中创建 DataReader 对象。
- ☐ 调用 ExecuteReader 对象。

`DataReader` 对象的 `Read` 方法可以判断 `DataReader` 对象中的数据是否还有下一行，并将光标下移到下一行。通过 `Read` 方法可以判断 `DataReader` 对象中的数据是否读完。示例代码如下所示。

```
while (dr.Read())
```

通过 `Read` 方法可以遍历读取数据库中行的信息，当读取到一行时，需要获取某列的值只需要使用 “[” 和 “]” 运算符来确定某一列的值即可，示例代码如下所示。

```
while (dr.Read())
{
    Response.Write(dr["title"].ToString()+"<hr/>");
}
```

上述代码通过 `dr["title"]` 来获取数据库中 `title` 这一列的值，同样也可以通过索引来获取某一列的值，示例代码如下所示。

```
while (dr.Read())
{
    Response.Write(dr[1].ToString()+"<hr/>");
}
```

7.10.3 异常处理

在使用 `DataReader` 对象进行连接时，需要使用 `Try...Catch...Finally` 语句进行异常处理，以保证如果代码出现异常时连接能够关闭，否则连接将保持打开状态，影响应用程序性能。示例代码如下所示。

```
protected void Page_Load(object sender, EventArgs e)
{
    string str = "server=(local);database='mytable';uid='sa';pwd='Bbg0123456#'";
    SqlConnection con = new SqlConnection(str);
    con.Open();
    SqlCommand cmd = new SqlCommand("select * from mynews", con);
    SqlDataReader dr;
    try
    {
        dr = cmd.ExecuteReader();
        while (dr.Read())
        {
            Response.Write(dr[1].ToString() + "<hr/>");
        }
    }
    catch (Exception ee) //出现异常
    {
        Response.Write(ee.ToString()); //出现异常则抛出错错误语句
    }
    finally
    {
        dr.Close(); //强制关闭连接
        con.Close(); //强制关闭连接
    }
}
```

上述代码当出现异常时，则会抛出异常，并强制关闭连接。这样做就能够在程序发生异常时，依旧关闭连接应用程序与数据库的连接，否则大量的异常连接状态的出现会影响应用程序性能。

7.11 连接池概述

在应用程序与数据库交互中，建立和关闭数据库连接都是非常消耗资源的过程。如果一个应用程序需要大量的与数据库进行交互，则很有可能造成假死，以及崩溃的情况。使用连接池能够提高应用程序的性能。

连接池是 SQL Server 或 OLEDB 数据源的功能，它可以使特定的用户重复使用连接，数据库连接池技术的思想非常简单，将数据库连接作为对象存储在一个 Vector 对象中，一旦数据库连接建立后，不同的数据库访问请求就可以共享这些连接，这样，通过复用这些已经建立的数据库连接，可以极大地节省系统资源和时间。连接池的主要操作如下所示：

- ❑ 建立数据库连接池对象。
- ❑ 对于一个数据库访问请求，直接从连接池中得到一个连接。如果数据库连接池对象中没有空闲的连接，且连接数没有达到最大，创建一个新的数据库连接。
- ❑ 存取数据库。
- ❑ 关闭数据库，释放所有数据库连接。
- ❑ 释放数据库连接池对象。

注意：在关闭数据库这一步中，并非真正的关闭，而是将其放入空闲队列中。如实际空闲连接数大于初始空闲连接数则释放连接。

当一个网站用户需要同数据库之间进行交互时，服务器会为网站用户建立一个业务对象，每个业务对象维护自身的连接，这些业务对象自身会创建连接。当用户无需该业务对象时，业务对象会释放连接，如图 7-25 所示。

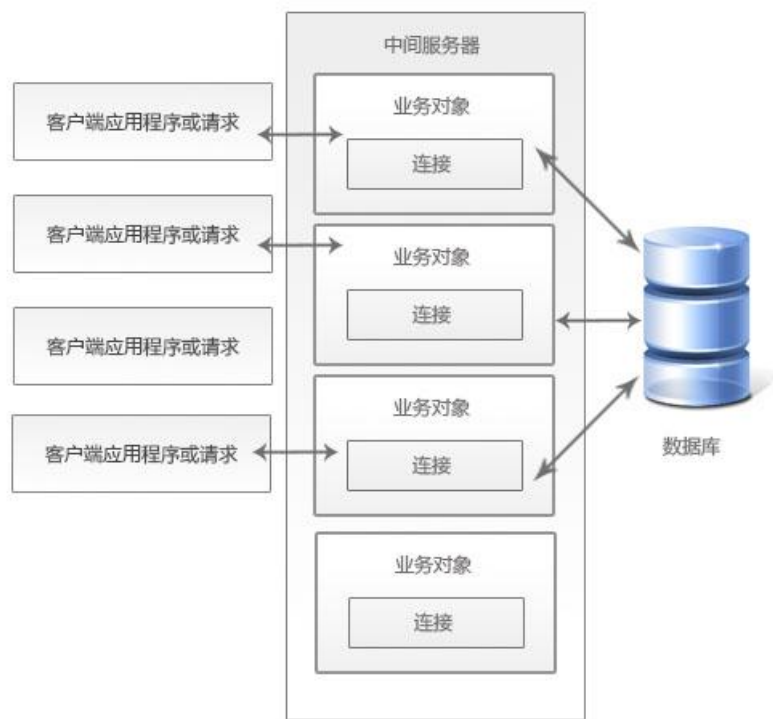


图 7-25 多层构架应用程序

当业务对数据库进行复杂的操作，并不停的打开和断开数据库连接，这样的操作会造成应用程序性能降低，因为重复的打开和断开数据库连接是非常消耗资源的，而使用连接池则可以避免这样的问题。连接池并不会真正的完全的关闭数据库与应用程序的连接，而将这些连接存放在应用程序连接池中。当一个新的业务对象产生时，会在连接池中检查是否已有连接，若无连接，则会创建一个新连接，否则会使用现有的匹配的连接，这样就提高了性能，如图 7-26 所示。

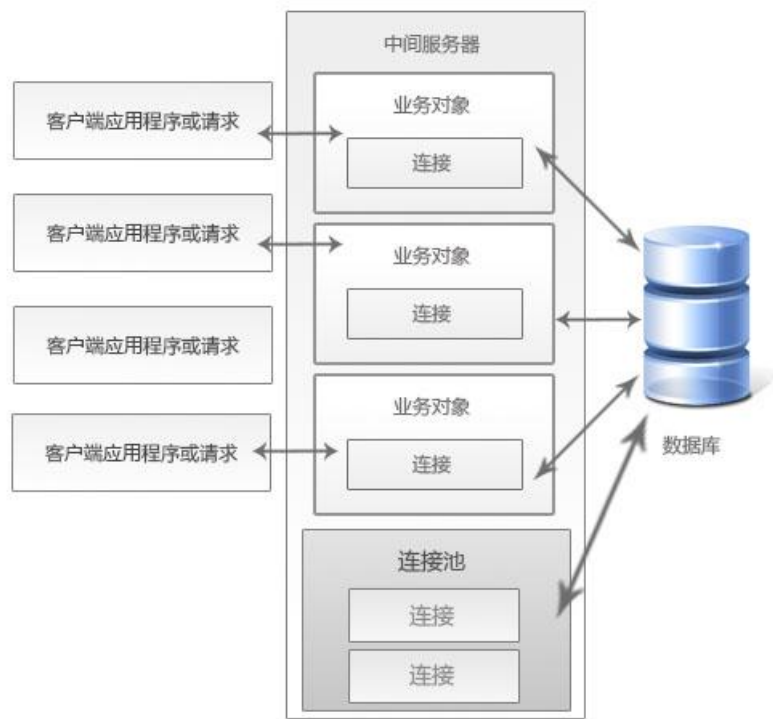


图 7-26 使用连接池

使用连接池能够提升应用程序的性能，特别是开发 Web 应用程序时，Web 应用程序通常需要频繁的与数据库之间进行交互，应用程序池能够解决 Web 引用中假死等情况，也能够节约服务器资源。但是，在创建连接时，良好的关闭习惯也是非常必要的。

7.12 参数化查询

在 Web 应用程序的开发过程中，Web 安全是非常重要的，现存的很多网站也都存在一些非常严重的安全漏洞，其中 SQL 注入是非常常见的漏洞，如果将查询语句进行参数化查询，可以减少 SQL 注入漏洞的概率，参数化查询示例代码如下所示。

```
string strSql = "select * from mynews where id= @id";
```

上述代码使用了参数化查询，在存储过程中，参数化是非常常见的，存储过程通过 Command 对象进行参数的添加和赋值。同样，参数化查询也可以通过 Command 对象进行添加和赋值，参数化查询过程如下所示。

- ❑ 创建一个 Command 对象。

- ❑ Command 对象增加一个参数。
- ❑ 通过索引对 Command 参数进行赋值。
- ❑ 执行 ExecuteReader 方法返回个 DataReader 对象。

通过 Command 对象可以为存储过程，以及参数化查询语句进行参数的添加，示例代码如下所示。

```
protected void Page_Load(object sender, EventArgs e)
{
    string str = "server=(local);database=mytable;uid='sa';pwd='sa'";
    SqlConnection con = new SqlConnection(str);
    con.Open();
    string strSql = "select * from mynews where id = @bh";
    SqlCommand cmd = new SqlCommand(strSql, con);           //创建 Command 对象
    cmd.Parameters.Add("@bh", SqlDbType.Int);               //增加参数 @bh
    cmd.Parameters[0].Value = 4;                             //通过索引为参数赋值
    SqlDataReader dr = cmd.ExecuteReader();                  //执行后返回 DataReader 对象
    while (dr.Read())                                       //遍历 DataReader 对象
    {
        Response.Write(dr["title"].ToString()+"<br>");
    }
}
```

参数化查询能够有效的解决一些安全问题，提高 Web 应用的安全性。同时，参数化查询能够极大的简化程序设计。只需要通过数值的更改而不需要修改 SQL 语句，极大的方便了应用程序的维护。

注意：如果未初始化 Parameter 数据类型的属性，但设置了 Value 属性，那么 Parameter 会自动选择合适的数据类型。

7.13 小结

本章接单的介绍了数据的基础知识，包括什么是数据库，数据库的作用。然后讲述了 SQL Server 2005 的数据库基本使用，并介绍了 SQL Server Management 管理工具的使用。通过介绍 SQL Server Management 管理工具，介绍了如何使用 SQL Server Management 管理工具和 SQL 语句创建表，删除表等过程。本章还包括：

- ❑ ADO.NET 连接 SQL 数据库：使用 ADO.NET 连接 SQL 数据库示例。
- ❑ ADO.NET 与 ADO：ADO 与 ADO.NET 发展史和利弊。
- ❑ Connection 对象：Connection 对象概述。
- ❑ 连接 SQL 数据库：使用 Connection 对象连接 SQL 数据库。
- ❑ 连接 Access 数据库：使用 Connection 对象连接 Access 数据库。
- ❑ DataAdapter 对象：讲解了 DataAdapter 对象的使用。
- ❑ Command 对象：讲解了 Command 对象的使用。
- ❑ DataSet 对象：讲解了 DataSet 对象中常用的方法，并高效使用 DataSet 开发。
- ❑ DataReader 对象：讲解了 DataReader 对象。
- ❑ 连接池概述：讲解了连接池。
- ❑ 参数化查询：讲解了使用参数化查询提供安全性保证和简化开发。

在了解了基本的 ADO.NET 对象，以及 ADO.NET 对象的作用后，下一章将讲解如何使用数据源控件来显式和操作数据库。