第5章 SQL Server 2005 的安全性

- SQL Server 2005 安全性概述
- SQL Server 2005 登录用户介绍
- SQL Server 2005 权限管理
- SQL Server 2005 密码策略和证书

5-1 SQL Server 2005 安全性概述

- 了解数据库安全性的产生。
- 了解安全措施的 5 个级别。
- 可以区分 Windows 认证模式和 SQL Server 混合认证模式的区别。
- SQL Server 2005 安全性概述。
- 掌握用户身份认证，主体和安全对象的内涵。

SQL Server 作为一款不断发展壮大的数据库管理系统，正由中型数据库管理体系向大型数据库管理系统发展，而 SQL Server 2005 正是这一发展的结果。其中，数据库的安全性也随着版本的不断升级发生了革命性的变化。我们都知道，在过去十年里，数据库技术的侧重点已经从可伸缩性演变为可靠性，特别是在当前网络恐怖主义存在的时代，安全是数据库最重要的性能。虽然通常的应用程序部署都为恶意肇事者设置诸多障碍，如防火墙、密码、用户控制和监控等，但事实上数据库自身才是最后一道屏障。SQL Server 2005 的安全体系结构已经进行了重新设计，能够提供全方位的安全保障，来成功防范强力攻击、电子欺骗、SQL 注入和其他攻击。

目前所有的技术中，没有任何一种技术像数据库这样受到持续的审核和攻击，而对数据的威胁比黑客的威胁更可怕，因为数据库不仅仅是存储数据的载体，更是敏感信息的承接对象，一旦泄漏可能会对国家和企业造成无法挽回的恶果。所以大多数公司在具体的软件项目的研发过程或者实际应用中，对数据库的审核及信息发布都有着非常严格的安全措施，当然也包括设计信息的具体研发人员和使用者，如开发人员和数据库管理员等。

5-1-1 数据库安全性的产生

数据库的安全性是指在信息系统的不同层次保护数据库，防止未授权的数据访问，避免数据的泄漏、不合法的修改或对数据的破坏。安全性问题不是数据库系统所独有的，它来自各个方面，其中既有数据库本身的安全机制，如用户认证、存取权限、视图隔离、跟踪与审查、数据加密、数据完整性控制、数据访问的并发控制、数据库的备份和恢复等方面，也涉及到计算机硬件系统、计算机网络系统、操作系统、组件、Web 服务、客户端应用程序、网络浏览器等。只是在数据库系统中大量数据集中存放，而且为许多最终用户直接共享，从而使安全性问题更为突出，每一个方面产生的安全问题都可能导致数据库数据的泄露、意外修改、丢失等后果。

典型的数据库安全性问题是没有进行有效的用户权限控制引起的数据泄露。B/S 结构的网络环境下数据库或其他的两层或三层结构的数据库应用系统中，一些客户端应用程序总是使用数据库管理员权限与数据库服务器进行连接（如 Microsoft SQL Server 的管理员 SA），在客户端功能控制不合理的情况下，可能使操作人员访问到超出其访问权限的数据。

在安全问题上，DBMS 应与操作系统达到某种意向，理清关系，分工协作，以加强 DBMS 的安全性。数据库系统安全保护措施是否有效是数据库系统的主要指标之一。

为了保护数据库，防止恶意的滥用，可以在从低到高 5 个级别上设置各种安全措施。

（1）环境级：计算机系统的机房和设备应加以保护，防止有人进行物理破坏。

（2）职员级：工作人员应清正廉洁，正确授予用户访问数据库的权限。

（3）OS 级：应防止未经授权的用户从 OS 处着手访问数据库。

（4）网络级：大多数 DBS 都允许用户通过网络进行远程访问，因此网络软件内部的安全性至关重要。

（5）DBS 级：DBS 的职责是检查用户的身份是否合法及使用数据库的权限是否正确。

本章只讨论与数据库系统中的数据保护密切相关的内容。

5-1-2 SQL Server 2005 安全性概述

如果一个用户要访问 SQL Server 数据库中的数据，必须经过三个级别的认证过程，如图 5-1 所示。第一个认证过程是 Windows 级别，即 Windows 身份验证，需通过登录账户来标识用户。身份验证只验证用户是否具有连接到 SQL Server 数据库服务器的资格。第二个级别的认证是 SQL Server 级别的认证，该认证过程是当用户访问数据库时，必须具有对具体数据库的访问权，即验证用户是否是数据库的合法用户。第三个级别是数据库级，该级别是指当用户操作数据库中的数据对象时，必须具有相应的操作权，即验证用户是否具有操作权限。

这就好比一幢大楼，身份认证的用户是进入 SQL Server 数据库软件的第一道大门，要想进楼首先必须有这栋大楼的钥匙，这就是第一把钥匙——身份认证权限；进入大楼后，如果你想进某一家的门，就好比开始具体操纵某个数据库，你必须有第二把钥匙——数据库的访问权；进入某家后，如果你想具体操纵某张表、视图或者其他数据库里面的对象，就好比获取这家保险柜的文件必须有保险柜的钥匙一样，这就是第三把钥匙——操作权，如图 5-2 所示。

SQL Server 的安全性管理的概念包括用户身份认证、主体、安全对象等几个方面，下面重点介

绍用户的身份认证。

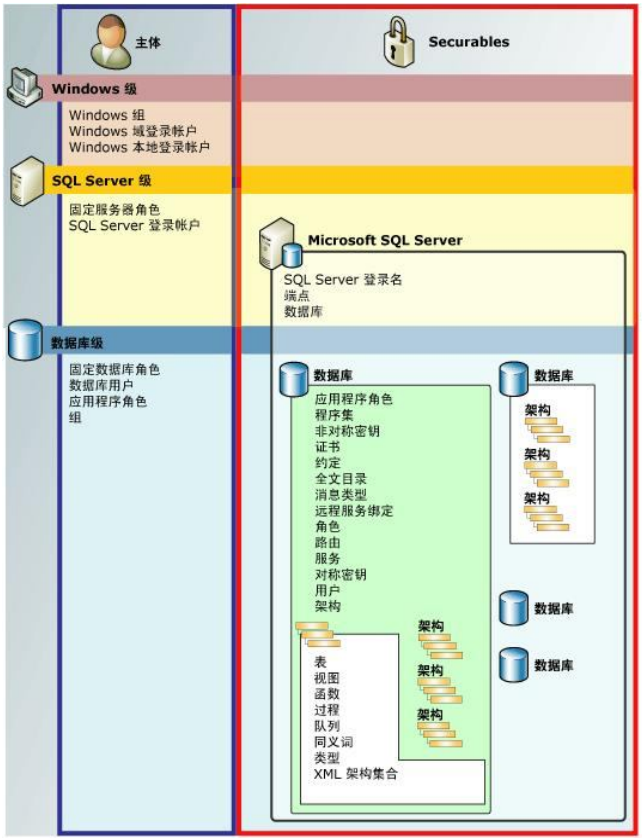

图 5-1　SQL Server 的安全认证级别示意图

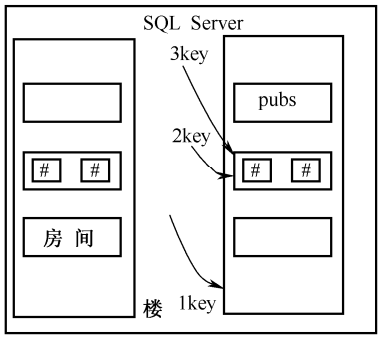

图 5-2　SQL Server 的安全体系结构

1. 用户身份认证

SQL Server 的用户有两种类型。

（1）Windows 授权用户：来自于 Windows 的用户或组。

（2）SQL 授权用户：来自于非 Windows 的用户，称其为 SQL 用户。

Microsoft SQL Server 为不同的用户类型提供了不同的安全认证模式。

（1）Windows 身份验证模式。Windows 身份验证模式使用户得以通过 Microsoft Windows NT 4.0 或 Windows 2000 用户账户连接 SQL Server。用户必须首先登录到 Windows 中，再登录到 SQL Server。用户登录到 SQL Server 时，只需选择 Windows 身份验证模式，无须再提供登录账户和密码，系统会从用户登录到 Windows 时提供的用户名和密码中查找用户的登录信息，以判断其是否为 SQL Server 的合法用户。

对于 SQL Server 来说，一般推荐使用 Windows 身份验证模式。因为这种安全模式能够与 Windows 操作系统的安全系统集成在一起，用户的网络安全特性在网络登录时建立，并通过 Windows 域控制器进行验证，从而提供更多的安全功能。但 Windows 验证模式只能用在 Windows NT 4.0 或 Windows 2000 Server 操作系统的服务器上，在 Windows 98 等个人操作系统上不能使用 Windows 身份验证模式，只能使用混合验证模式。

（2）混合验证模式。混合验证模式表示 SQL Server 接受 Windows 授权用户和 SQL 授权用户。如果不是 Windows 操作系统的用户或者是 Windows 客户端操作系统的用户使用 SQL Server，则应该选择混合验证模式。如果在混合模式下选择使用 SQL 授权用户登录 SQL Server，用户必须提供登录名和密码，SQL Server 使用这两部分内容来验证用户，SQL Server 通过检查是否已设置 SQL Server 登录账户，以及指定的密码是否与记录的密码匹配，进行身份验证。

SQL Server 在安装时，会自动创建一个 DB 服务器的登录用户 sa，即系统管理员（System Administrator），用于创建其他登录用户和授权。由于该用户的特殊性，一般在具体应用时并不启用，同时 SQL Server 2005 默认也是禁止使用该用户的，即便是启用该用户，也要注意设置较为安全的密码，以防止远程用户利用该用户登录后进行恶意破坏。

注意：Windows 认证模式和 SQL Server 混合认证模式的区别。

两个验证方式有明显的不同，主要集中在信任连接和非信任连接上。

Windows 身份验证相对于混合模式更加安全，使用本连接模式时，仅仅根据用户的 Windows 权限来进行身份验证，称为“信任连接”，但是在远程连接时会因 HTML 验证的缘故而无法登录。

- Windows 认证模式的优点是：数据库管理员的工作集中在管理数据库方面，而不是管理用户账户。对用户账户的管理可以交给 Windows 服务器去完成。Windows 服务器有着更强的用户账户管理工具。可以设置账户锁定、密码期限等。如果不是通过定制来扩展 SQL Server，SQL Server 是不具备这些功能的。Windows 服务器的组策略支持多个用户同时被授权访问 SQL Server。该模式是默认的身份验证模式，比混合模式更为安全。请尽可能使用 Windows 身份验证。
- 混合模式验证就是当本地用户访问 SQL Server 时采用 Windows 身份验证建立信任连接，当远程用户访问时由于未通过 Windows 认证，而进行 SQL Server 认证（使用 sa 的用户也可以登录 SQL），建立“非信任连接”，从而使得远程用户也可以登录。

混合认证模式的优点是：①创建 Windows 服务器之外的一个安全层次；②支持更大范围的用户，如 Novell 网用户等；③一个应用程序可以使用单个的 SQL Server 登录账号和口令。

二者在登录数据库的安全决策流程上不同，具体流程如图 5-3 所示。

2. 主体

主体是 SQL Server 2005 的术语，表示可以请求 SQL Server 资源的个体、组和过程。与 SQL Server 授权模型的其他组件一样，主体也可以按层次结构排列。每个主体都有一个唯一的安全标识符（SID），可为其授权访问数据库系统中的对象权限。主体存在三个级别：Windows 级别的主体、

SQL Server 级别的主体、数据库级别的主体，如表 5-1 所示。

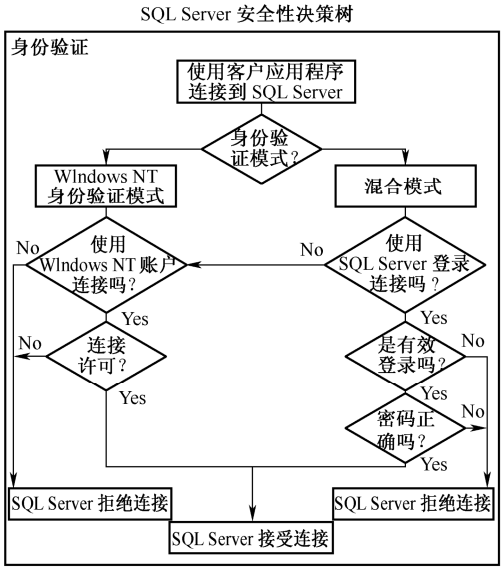

图 5-3　SQL Server 身份验证的不同流程

表 5-1　SQL Server 2005 级别对应的主体

级别	主体
Windows 级别的主体	● Windows 域用户账户 ● Windows 本地用户账户 ● Windows 组
SQL Server 级别的主体	● SQL Server 登录名 ● SQL Server 角色
数据库级别的主体	● 数据库用户 ● 数据库角色 ● 应用程序角色 ● public 数据库角色

3. 安全对象

SQL Server 2005 存在 3 种安全对象的作用域范围：服务器、数据库和架构。如表 5-2 所示为 SQL Server 2005 安全对象包含的作用域范围。

表 5-2　SQL Server 2005 安全对象包含的作用域范围

安全对象作用域	包含的安全对象
服务器作用域	● 端点、登录账户、数据库
数据库作用域	● 用户、角色、应用程序角色、程序集、消息类型、路由、服务 ● 远程服务绑定、全文目录、证书、非对称密钥、对称密钥 ● 约定、架构
模式作用域	● 聚合、约束、函数、过程、队列、统计信息 ● 同义词、表、视图

5-2 SQL Server 2005 登录用户

- 掌握建立 Windows 认证模式下的用户登录。
- 掌握建立 sa 用户登录。
- 掌握建立 SQL Server 用户登录。
- 掌握通过命令方式授权 Windows 用户登录 SQL Server。
- 掌握通过命令方式创建 SQL Server 登录账户。
- 学习查看 SQL Server 登录账户。
- 学习修改和删除 SQL Server 登录账户信息。

在 5-1 节中我们知道了 SQL Server 2005 的用户身份认证模式可以分为两种：Windows 认证模式和 SQL Server 混合认证模式，因此 SQL Server 2005 的用户根据身份认证模式的不同，也可以分为两种：Windows 用户和 SQL Server 用户。本节将通过实验的形式，分别实现这两种不同用户的注册和登录。

5-2-1 建立 SQL Server 2005 安全用户

本实验将完成在上一节中提到的 Windows 用户登录、SQL Server 用户登录（包括 sa 用户登录）的 3 个实验，具体实验操作如下。

实验 1：建立 Windows 认证模式下的用户登录

（1）在操作系统的计算机管理界面下，展开本地用户和组，在用户下建立 3 个用户 u1、u2、u3，密码与用户名相同，如图 5-4 所示。然后新建一个组，命名为 qq，并将 u2 和 u1 放到该组下。放置一个用户在一个组里面的具体步骤是：右击 u1 用户，在弹出的快捷菜单中选择“隶属于”命令，在弹出对话框的“隶属于”选项卡中将 qq 组添加进来，如图 5-5 所示。

图 5-4 新建操作系统用户

（2）展开数据库管理界面，右击“安全性”的“登录名”，在弹出的快捷菜单中选择“新建登

录名”命令，如图 5-6 所示。

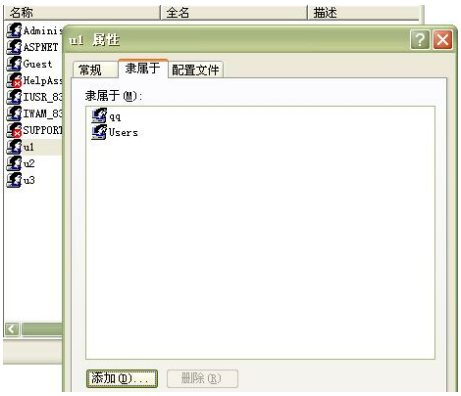

图 5-5　配置 u1 用户隶属于 qq 组界面

图 5-6　新建登录名

（3）在弹出的“新建登录名”对话框中单击“搜索”按钮，会弹出“选择用户和组”对话框。在该对话框内单击“对象类型”按钮，然后在弹出的“对象类型”对话框中选择全部类型，也包括组。确定“对象类型”后，单击“选择用户和组”界面中的“高级”按钮，选择组用户 qq 后，返回“选择用户和组”对话框，单击“确定”按钮后建立登录名工作完成，如图 5-7 所示。

图 5-7　建立 Windows 组用户的登录权限

该步骤的目的是为下一步的“组用户”授权做准备，即当为 qq 组授权后，该组下面的 u1、u2 用户就可以正常登录了。当然，这一步也可以为某个具体的用户授权，如 u3 用户，操作步骤相同，这里就不再多述，请读者自行完成。

（4）测试。切换 Windows 用户，以 u1 用户的身份进入操作系统后，启动 SQL Server 2005 数据库，查看是否可以凭 Windows 用户的身份验证登录。此时登录成功后会发现什么也做不了，因为没有授权操作具体的数据库和表，如同仅有楼钥匙而没有家门和保险柜的钥匙一样。

实验 2：建立 sa 用户登录

sa 用户是 SQL Server 的超级管理员用户，由于该用户的特殊性，往往容易成为被攻击的漏洞对象，因此建议不要轻易启动该用户。下面讲述如何以 sa 用户的身份登录当前的 SQL Server 系统。

（1）右击当前的实例，在弹出的快捷菜单中选择“属性”命令。在弹出的“服务器属性”对

话框中选择“选择页”中的“安全性”，将服务器安全认证改为“SQL Server 和 Windows 身份验证模式”，如图 5-8 所示。

（2）展开安全性的登录名文件夹，右击下面的 sa 用户，然后在“登录属性-sa”对话框中选择“状态”页并将“登录”设为“启用”，最后单击“应用”、“确定”按钮保存设置，如图 5-9 所示。这里建议回到常规界面，设置 sa 用户的密码并强制实施密码策略，起到对该用户的最后一道密码保护作用。

图 5-8　改为 SQL Server 和 Windows 身份认证模式

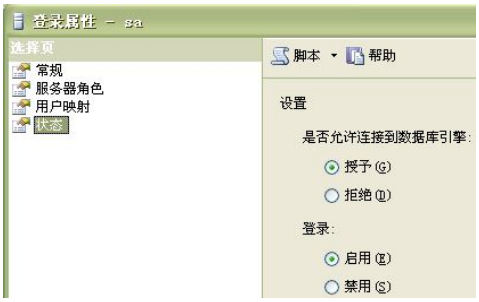

图 5-9　将 sa 用户的登录状态改为启用

（3）测试。选择新建一个实例连接，在弹出的对话框中选择身份认证为“SQL Server 身份验证”，键入用户名 sa 和密码后测试连接，如图 5-10 所示。

图 5-10　服务器安全认证改为 SQL Server 和 Windows 身份验证

问题：按照上述步骤配置后依然无法成功使得 sa 用户登录数据库系统，应如何处理呢？

（1）打开 SQL Server 外围应用配置器，选择“服务器和连接的外围应用配置器”，然后在“服务器和连接的外围应用配置器”对话框中选择“远程连接”页，并将本地连接和远程连接设为同时使用 TCP/IP 和 named pipes，最后单击“应用”、“确定”按钮保存设置，如图 5-11 所示。

（2）打开 SQL Server Configuration Manager，单击 SQL Server 2005 服务，右击 SQL Server（MSSQLSERVER），在弹出的快捷菜单中选择“重新启动”命令，如图 5-12 所示。

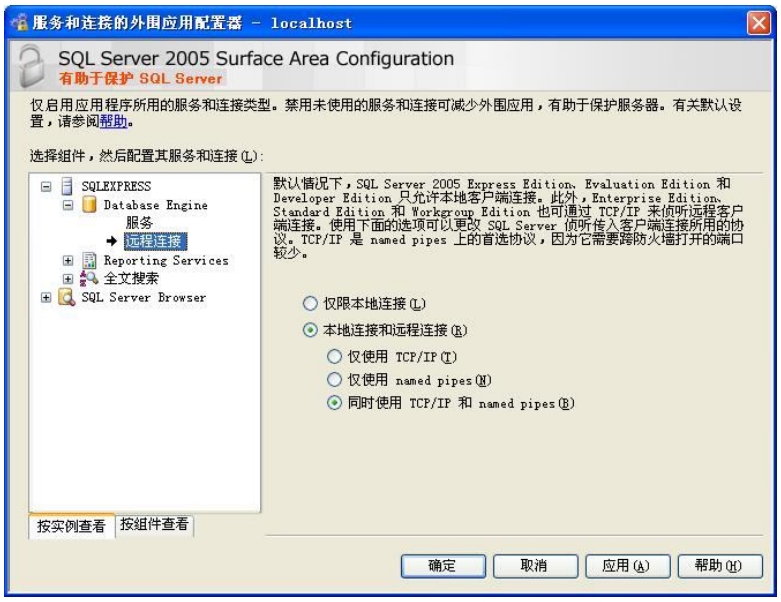

图 5-11　重新设置远程连接属性

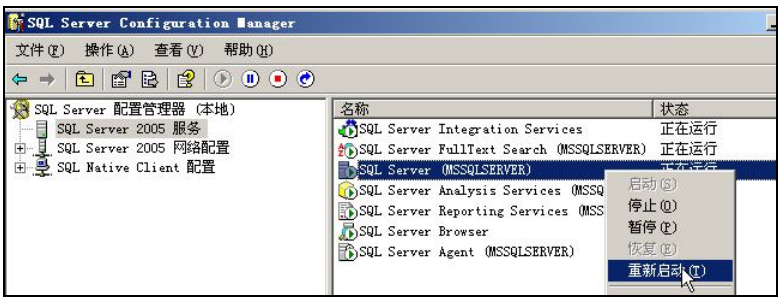

图 5-12　重新启动 SQL Server 服务

实验 3：建立 SQL Server 用户登录

SQL Server 用户的建立和 sa 用户设置非常类似，唯一不同的是该用户的名称可以由用户自行定义。

（1）右击“安全性”中的“登录名”，在弹出的快捷菜单中选择“新建登录名”命令，如图 5-13 所示。

图 5-13　新建登录名

（2）在展开的“新建登录名”对话框的“常规”选择页里面，键入登录名为 qq，选择“SQL Server 身份验证”，并输入密码和确认密码，取消选中“强制实施密码策略”复选项。最后在“状

态”选项中确定授权都已经授权和启用后，单击确定建立 qq 用户，如图 5-14 所示。

图 5-14　新建 SQL Server 用户

（3）测试。新建一个用户连接，以 SQL Server 身份认证登录，用户名为 qq，键入密码后登录 SQL Server 系统。但是将发现，无论是查询还是删除等操作，都将被禁止完成，同样是因为没有被授权。

5-2-2　通过命令方式建立 SQL Server 2005 安全用户

上一节在可视化环境下分别完成了 Windows 用户登录、sa 用户登录，以及 SQL Server 用户登录的 3 个实验，该系列实验的目的是完成 SQL Server 登录账户的管理工作。在进入下面的实验之前，先了解一下登录账户和数据库用户之间的区别和联系。

（1）登录账户是访问 SQL Server 的通行证，但并不能访问服务器中的数据库；即没有权限的数据库用户才是一个登录账户，而登录账户是成为数据库用户的前提。

（2）要访问特定的数据库必须要有用户名，用户名的信息存放在该数据库的 SysUsers 表中，该用户名是没有密码的，我们将这个可以访问数据库的用户名称为数据库用户。数据库用户是一个登录账户在某个数据库中的映射；一个登录账户可以同时与多个数据库发生关联；只有登录账户创建完成后，才能为其创建数据库用户名。

本节的重点依然是建立 SQL Server 登录账户，仅将实现的方式变为通过命令行的方式建立。具体实验步骤如下。

实验 4：授权 Windows 用户登录 SQL Server

可以利用系统存储过程 sp_grantlogin 实现 windows 用户登录授权，语法如下：

```
sp_grantlogin [@loginame =] 'login'
```

系统存储过程 sp_revokelogin 实现收回 windows 用户登录权限，语法如下：

```
sp_revokelogin [@loginame =] 'login'
```

系统存储过程 sp_denylogin 实现拒绝 windows 用户登录，语法如下：

```
sp_denylogin [@loginame =] 'login'
--管理 windows 用户和组需要使用 sp_grantlogin
EXEC sp_grantlogin 'TEACHER9\u1'
--删除 windows 登录
EXEC  sp_revokelogin 'TEACHER9\u1'
--拒绝 windows 登录
```

```
EXEC  sp_denylogin  'TEACHER9\u1'
--更改 windows 登录
EXEC sp_change_users_login 'TEACHER9\u1'
```

实验 5：创建 SQL Server 登录账户实验

可以利用系统存储过程 sp_addlogin 创建 SQL Server 登录账户，语法如下：

```
sp_addlogin [ @loginame = ] 'login'
  [ , [ @passwd = ] 'password' ]
  [ , [ @defdb = ] 'database' ]
  [ , [ @deflanguage = ] 'language' ]
  [ , [ @sid = ] sid ]
  [ , [ @encryptopt = ] 'encryption_option' ]
```

一般的语法书写格式如下所示：

```
sp_addlogin '用户名','密码' [,'登录用户使用的默认数据库']
```

例如：创建一个登录账户：名为 u3，密码为 u3，使用的默认数据库为 school。

```
EXEC sp_addlogin 'u3','u3','school'
```

注意：执行后将出现下列错误：密码有效性验证失败。该密码不够复杂，不符合 Windows 策略要求，要求密码的复杂度必须提高。更改后的代码如下所示：

```
EXEC sp_addlogin 'u3','123456','school'
```

实验 6：查看 SQL Server 登录账户

可以利用系统存储过程 sp_helplogins 查看 SQL Server 登录账户，该存储过程将提供有关每个数据库中的登录及相关用户的信息。语法如下：

```
sp_helplogins [ [ @LoginNamePattern = ] 'login' ]
```

需要注意的是，如果指定该参数，则'login'用户必须存在。如果未指定'login'，则返回有关所有登录的信息。

例如：查看 sa 用户的详细登录信息。

```
EXEC sp_helplogins 'sa'
```

执行结果如图 5-15 所示。

	LoginName	SID	DefDBName	DefLangName	AUser	ARemote
1	sa	0x01	Family	简体中文	yes	no

	LoginName	DBName	UserName	UserOrAlias
8	sa	Family	dbo	User
9	sa	Grade_Sys	db_owner	MemberOf
10	sa	Grade_Sys	dbo	User

图 5-15　sa 用户的详细登录信息

实验 7：修改 SQL Server 登录账户信息

系统存储过程 sp_defaultdb 可以更改 SQL Server 登录名的默认数据库，语法格式如下：

```
sp_defaultdb [ @loginame= ] 'login', [ @defdb= ] 'database'
```

系统存储过程 sp_defaultlanguage 可以更改 SQL Server 登录后的默认语言，但需要注意的是 SQL Server 2005 的后续版本将删除该功能，语法格式如下：

```
sp_defaultlanguage [ @loginame= ] 'login'  [ , [ @language= ] 'language' ]
```

系统存储过程 sp_password 可以更改 SQL Server 登录用户的登录密码，同样 SQL Server 2005 的后续版本将删除该功能，语法格式如下：

```
sp_password [ [ @old = ] 'old_password' , ] { [ @new =] 'new_password' }
[ , [ @loginame = ] 'login' ]
```

例 1．将 u3 用户的默认数据库改为 school 数据库。

```
EXEC sp_defaultdb 'u3','school'
```

例 2．将 u3 用户的默认语言改为法语。

```
EXEC sp_defaultlanguage 'u3', 'french'
```

例 3．将 u3 用户的原始登录密码 u3 改为新的密码 1234。

```
EXEC sp_password 'u3', '1234','u3'
```

实验 8：删除 SQL Server 登录账户

系统存储过程 sp_droplogin 将删除某个已经注册的登录用户，其语法格式如下：

```
sp_droplogin [ @loginame= ] 'login'
```

例 4．删除注册登录用户 u3。

```
EXEC sp_droplogin 'u3'
```

5-3　SQL Server 2005 权限管理

- 掌握用户和模式的分离，以及执行上下文概念的内涵。
- 掌握通过管理控制平台对用户进行授权。
- 掌握通过命令行对用户进行授权与收权。
- 掌握用户的角色概念，掌握对用户进行服务器角色授权技术，掌握通过命令形式对用户进行数据库角色授权。
- 学习应用程序角色的创建和使用。

用户登录 SQL Server 服务器之后，并不代表该用户可以访问所有数据库资源。一个 SQL Server 服务器上可以有很多个数据库，每个数据库里可以有很多个数据表。因此，不可能让每一个能登录 SQL Server 服务器的账户都能控制所有的数据库；即使在同一个数据库中，也不一定每个账户都能访问所有数据表，例如不同部门的人只能访问属于该部门的业务表；即使是同一个数据表，不同账户的访问权限可能也不尽相同，例如对某个数据表来说管理人员可以完全操作其中的数据，而普通人员只能查看这些数据。

5-3-1　用户权限概述

SQL Server 2005 中，权限可以分为两个方面，一个是对数据库服务器自身的控制权限，如创

建、修改、删除数据库，管理磁盘文件，添加、删除连接服务器等；另一个是对数据库数据方面的控制权限，如可以访问数据库中的哪些数据表、视图、存储过程等，或是对数据表执行哪些操作，是 insert，还是 update，或是 select 等。在 SQL Server 2005 中，可以把访问权限设置给用户或角色。

SQL Server 2005 又增加了两个新特性：其一是用户与数据库的模式被分离开来，其二是可以执行上下文。

1. 用户和模式的分离

模式是一个逻辑容器，可以在其中对数据库对象（表、存储过程、视图等）进行逻辑分组，这与.NET 框架基类库中的命名空间是一样的。因为 SQL Server 中的一个模式也可能有一个所有者，因此可以指定在一个给定模式中的每个对象都拥有一个相同的所有者。SQL Server 2000 的实现模式并不很好：模式的名字与用户的名字是相同的。因此，在模式和数据库对象的所有者之间存在一个直接的关系，如图 5-16 所示。

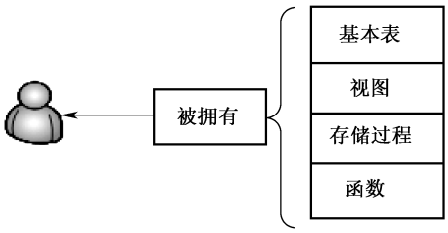

图 5-16　SQL Server 2000 中的用户和模式关系

而在 SQL Server 2005 中，模式成为数据库中一个单独的拥有名字和所有者的本机对象。这样的模式与用户不再存在关系，如图 5-17 所示。

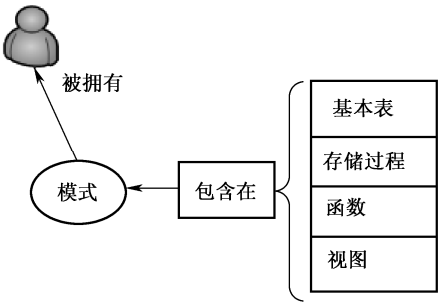

图 5-17　在 SQL Server 2005 中用户和模式分离开

2. 执行上下文

SQL Server 2005 的另外一个新特征是允许改变一个存储过程的执行上下文。可以通过用 EXECUTE AS 语句来改变执行上下文进而控制这些数据库对象是以哪些用户身份执行的。EXECUTE AS 语句支持下列选项：

```
EXECUTE AS CALLER
EXECUTE AS USER='user name'
EXECUTE AS SELF
EXECUTE AS OWNER
```

EXECUTE AS CALLER 是默认选项，这时一个存储过程是以调用用户的身份执行的。使用 EXECUTE AS USER 选项时，存储过程则用指定的用户身份执行。当使用 EXECUTE AS SELF 时，存储过程用创建者的身份执行。最后，可以使用 EXECUTE AS OWNER 来执行存储过程，其身份是该对象的所有者。

5-3-2 用户权限配置的实现

1. 用户权限有哪些

用户也就是使用 SQL Server 的人，每个用来登录数据库的账户都是一个用户。通过用户这个对象，可以设置数据库的使用权限。同一个数据库可以拥有多个用户，同一个用户也可以同时访问多个数据库。用户权限配置的根本是授予用户管理的权限，而管理权限包括授予或废除执行以下活动的用户权限：

（1）处理数据和执行过程（对象权限）。

- 处理数据或执行过程时需要的权限称为对象权限的权限类别。
- SELECT、INSERT、UPDATE 和 DELETE 语句权限，它们可以应用到整个表或视图上。
- SELECT 和 UPDATE 语句权限，它们可以选择性地应用到表或视图中的单个列上。

（2）创建数据库或数据库中的项目（语句权限）。

- 创建数据库或数据库中的项（如表或存储过程）所涉及的活动要求的一类权限称为语句权限。

（3）利用授予预定义角色的权限（暗示性权限）。

- 数据库对象所有者还有暗示性权限，可以对所拥有的对象执行一切活动。例如，拥有表的用户可以查看、添加或删除数据，更改表定义，或控制允许其他用户对表进行操作的权限。
- 暗示性权限可以控制那些只能由预定义系统角色的成员或数据库对象所有者执行的活动。例如，sysadmin 固定服务器角色成员自动继承在 SQL Server 安装中进行操作或查看的全部权限。

2. 用户权限的类别

用户权限的类别包括授权、拒绝、收权。授权是将操作具体数据库对象的具体操作类型授予特定的系统或自定义的用户；收权是将用户对具体数据库对象的操作权限收回；拒绝是使得系统或自定义的用户无法对数据库对象进行某种特定的操作。同时，权限的类别可以彼此转换，如图 5-18 所示。

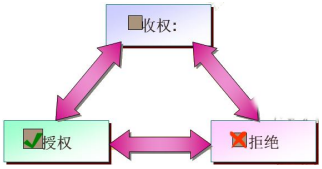

图 5-18 用户权限的类别图

实验 9：通过管理控制平台对用户进行授权

本节继承 5-2-1 节的实验内容，继续对建立的自定义用户 qq 进行授权工作。

（1）依次单击“对象资源管理器”窗口中树型节点前的“＋”号，直到展开目标数据库的“用户”节点为止，如图 5-19 所示。在“用户”节点下面的目标用户上右击，在弹出的快捷菜单中选择“属性（R）”命令。

图 5-19　用对象资源管理器为用户添加对象权限

（2）弹出“数据库用户”对话框，选择“选择页”列表中的“安全对象”项，进入权限设置页面，单击“添加”按钮，如图 5-20 所示。

图 5-20　“数据库用户”对话框

（3）弹出“添加对象”对话框，如图 5-21 所示，单击要添加的对象类别前的单选按钮，添加权限的对象类别，然后单击“确定”按钮。

图 5-21　“添加对象”对话框

（4）弹出“选择对象”对话框，如图 5-22 所示，单击“对象类型”按钮。

图 5-22　“选择对象”对话框

（5）弹出“选择对象类型”对话框，依次选择需要添加权限的对象类型前的复选框，选中其对象，如图 5-23 所示。最后单击“确定”按钮。

（6）回到“选择对象”对话框，此时在该对话框中出现刚才选择的对象类型，单击该对话框中的“浏览”按钮。在弹出的“查找对象”对话框中依次选择要添加权限的对象前的复选框，选中其对象，如图 5-24 所示。最后单击“确定”按钮。

图 5-23　“选择对象类型”对话框

图 5-24　“查找对象”对话框

（7）回到“选择对象”对话框，并且已包含了选择的对象。确定无误后，单击该对话框中的“确定”按钮，完成对象选择操作。然后回到“数据库用户”对话框，此对话框中已包含用户添加的对象，依次选择每一个对象，并在下面该对象的“显示权限”窗口中根据需要选择“授予/拒绝”列的复选框，添加或禁止对该（表）对象的相应访问权限，如图 5-25 所示。

图 5-25　“数据库用户”对话框

在图 5-25 中，授予 qq 用户对 course 表具有修改、控制、插入和查询的权限，同时具有授予其他用户修改的权限，另外也拒绝 qq 用户删除 course 表信息的权限。对于 qq 用户的查询权限还可以进行更细致的设置工作，即当授予 select 权限后就可以单击“列权限”进行配置了。

（8）如图 5-26 所示，授予 qq 用户查询 CNO、CPNO、TNO 属性的权利，拒绝 qq 用户查询 CNAME 的权限。设置完每一个对象的访问权限后，单击“确定”按钮，完成给用户添加数据库对象权限的所有操作。

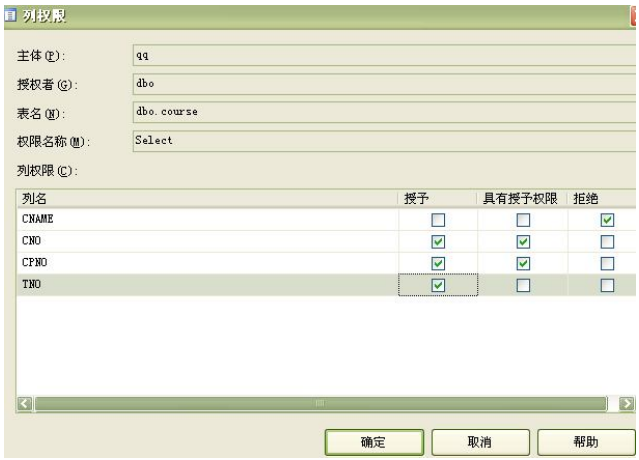

图 5-26　“列权限”对话框

实验 10：通过命令行对用户进行授权与收权

本节继承 5-2-1 节的实验内容，继续对建立的自定义用户 qq 进行授权工作。

1. 建立用户 qq，并将 qq 用户添加到 school 数据库的用户列表中

此步骤中学习系统存储过程 sp_grantdbaccess，该存储过程的语法格式如下：

```
sp_grantdbaccess [ @loginame = ] 'login'
    [ , [ @name_in_db = ] 'name_in_db' [ OUTPUT ] ]
```

其中，参数[@loginame =] 'login '是映射到新数据库用户的 Windows 组、Windows 登录名或 SQL Server 登录名的名称；参数[@name_in_db=] 'name_in_db' [OUTPUT]是新数据库用户的名称。

```
EXEC sp_addlogin 'qq','123456','school'
--建立 qq 用户，密码为 123456，默认登录数据库为 school
use school
EXEC sp_grantdbaccess 'qq','qq'
--将 SQL Server 用户 qq 作为 school 数据库的用户出现，名称也叫 qq
```

2. 使得 QQ 用户在 school 数据库的用户列表中撤销

学习使用系统存储过程 sp_grantdbaccess，该存储过程将从当前数据库中删除数据库用户，但删除数据库用户时，依赖于该用户的权限和别名也会删除。语法格式如下：

```
sp_revokedbaccess [ @name_in_db = ] 'name'
```

其中，参数[@name_in_db =] 'name'是要删除的数据库用户名称。

```
EXEC sp_revokedbaccess 'qq'  --删除 school 数据库的用户'qq'
```

3. 通过关键词 GRANT 对 qq 用户进行授权

GRANT 语句的完整语法非常复杂，但其基本意义是将安全对象的权限授予特定用户主体。语法格式如下：

```
GRANT { ALL [ PRIVILEGES ] }
| permission [ ( column [ ,...n ] ) ] [ ,...n ]
[ ON [ class :: ] securable ] TO principal [ ,...n ]
[ WITH GRANT OPTION ] [ AS principal ]
```

（1）授予 ALL 参数相当于授予以下权限：

- 如果安全对象为数据库，则 ALL 表示 BACKUP DATABASE、BACKUP LOG、CREATE DATABASE、CREATE DEFAULT、CREATE FUNCTION、CREATE PROCEDURE、CREATE RULE、CREATE TABLE 和 CREATE VIEW。
- 如果安全对象为标量函数，则 ALL 表示 EXECUTE 和 REFERENCES。
- 如果安全对象为表值函数，则 ALL 表示 DELETE、INSERT、REFERENCES、SELECT 和 UPDATE。
- 如果安全对象为存储过程，则 ALL 表示 DELETE、EXECUTE、INSERT、SELECT 和 UPDATE。
- 如果安全对象为表，则 ALL 表示 DELETE、INSERT、REFERENCES、SELECT 和 UPDATE。
- 如果安全对象为视图，则 ALL 表示 DELETE、INSERT、REFERENCES、SELECT 和 UPDATE。

（2）关键词 on 后面可以跟具体的授权操作对象，如表、视图等。

（3）关键词 to 后面可以跟具体的授权用户。

（4）WITH GRANT OPTION 参数：如果指定了 WITH GRANT OPTION 子句，则获得某种权限的用户还可以把这种权限再授予其他的用户。如果没有指定 WITH GRANT OPTION 子句，则获得某种权限的用户只能使用该权限，但不能传播该权限。因此使用该参数也被称为“传播性授权”。

例 1．为用户 qq 授予 STUDENT 表的查询权。

```
GRANT SELECT ON STUDENT TO qq
```

例 2．为用户 qq 授予 SCore 表的查询和插入记录权。

```
GRANT SELECT,INSERT ON SCore TO qq
```

例 3．授予 qq 创建数据库表的权限。

```
GRANT CREATE TABLE TO qq
```

例 4．授予 qq 和 guest 创建数据库表和视图的权限。

```
GRANT CREATE TABLE,CREATE VIEW TO qq,guest
```

例 5．授予 qq 对 school 数据库中的 student 表进行 INSERT、UPDATE 和 DELETE 的权限。WITH GRANT OPTION 表示 qq 用户也可以用这些语句来向其他用户授权。

```
GRANT INSERT, UPDATE, DELETE ON student TO qq WITH GRANT OPTION
```

例 6．将对 Student 表的所有权限都授予 qq 用户。

```
GRANT ALL PRIVILEGES ON  Student TO qq
```

例 7．将对 SCor 表的查询权限授予 PUBLIC 角色。

```
GRANT  SELECT  ON    SCore  TO  PUBLIC
```

例 8．将对 Student 表的部分修改和查询权限授予 qq。

```
GRANT  UPDATE(Sno), SELECT(sno,sname)  ON   Student TO  qq;
```

4. 删除授权（REVOKE）和阻止授权（DENY）

REVOKE 语句可用于删除已授予的权限，DENY 语句可用于防止主体通过 GRANT 获得特定权限。授予权限将删除对所指定安全对象的相应权限的 DENY 或 REVOKE 权限。如果在包含该安全对象的更高级别拒绝了相同的权限，则 DENY 优先。但是，在更高级别撤消已授予权限的操作并不优先。语法格式与 GRANT 的基本语法结构和参数一致，此处不再复述。

例 9．收回用户 qq 对表 STUDENT 的查询权。

```
REVOKE SELECT ON STUDENT FROM qq
```

例 10．拒绝 qq 用户对 SCore 表进行更改。

```
DENY UPDATE ON SCore TO qq
```

例 11．收回 qq 创建数据库表的权限。

```
REVOKE CREATE TABLE FROM qq
```

例 12．拒绝 qq 创建视图的权限。

```
DENY CREATE VIEW TO qq
```

5-3-3　用户的角色

用上面的方式设置用户权限，看上去很直观很方便，然而一旦数据库的用户数很多的时候，设置权限的工作将会变得烦琐复杂。SQL Server 2005 里通过为角色设置权限解决这个问题。

角色是用来指定权限的一种数据库对象，每个数据库都有自己的角色对象，可以为每个角色设置不同的权限。利用角色，SQL Server 管理者可以将某些用户设置为某一角色，这样只要对角色进行权限设置便可以实现对所有用户权限的设置，大大减少了管理员的工作量。

SQL Server 2005 中，角色分为 3 种：服务器角色、数据库角色和应用程序角色。

1. 服务器角色

服务器角色是指根据 SQL Server 的管理任务，以及这些任务相对的重要性等级来把具有 SQL Server 管理职能的用户划分为不同的用户组，每一组所具有的管理 SQL Server 的权限都是 SQL

Server 内置的。服务器角色存在于各个数据库之中。

SQL Server 2005 提供了 8 种常用的固定服务器角色，其具体含义如下：

- 系统管理员（sysadmin）：拥有 SQL Server 所有的权限许可。
- 服务器管理员（Serveradmin）：管理 SQL Server 服务器端的设置。
- 磁盘管理员（diskadmin）：管理磁盘文件。
- 进程管理员（processadmin）：管理 SQL Server 系统进程。
- 安全管理员（securityadmin）：管理和审核 SQL Server 系统登录。
- 安装管理员（setupadmin）：增加、删除连接服务器，建立数据库复制以及管理扩展存储过程。
- 数据库创建者（dbcreator）：创建数据库，并对数据库进行修改。
- 批量数据输入管理员（bulkadmin）：管理同时输入大量数据的操作。

实验 11：对用户进行服务器角色授权

本次实验继续对建立的自定义用户 qq 进行角色授权工作。

（1）展开“安全性”目录中的“登录名”窗口，在“登录名”节点下面右击，在弹出的快捷菜单中选择“新建登录名”命令。在弹出的登录名窗口中，除了设置用户名称和密码为 qq 外，在其“服务器角色”项目中设置 qq 用户的服务器角色为系统管理员（sysadmin），如图 5-27 所示。单击“确定”按钮后建立 qq 用户，并且该用户具有系统管理员的权限。

图 5-27　授予 qq 用户系统管理员的权限

（2）测试。使用 qq 用户登录 SQL Server，测试其操作权限。

2. *数据库角色*

数据库角色是为某一用户或某一组用户授予不同级别的管理或访问数据库以及数据库对象的权限，这些权限是数据库专有的，并且可以使一个用户具有属于同一数据库的多个角色。

SQL Server 提供了两种类型的数据库角色：其一是固定的数据库角色，其二是用户自定义的数据库角色。

固定的数据库角色是指 SQL Server 已经定义了这些角色所具有的管理、访问数据库的权限，而且 SQL Server 管理者不能对其所具有的权限进行任何修改。SQL Server 中的每一个数据库中都有一组固定的数据库角色，在数据库中使用固定的数据库角色可以将不同级别的数据库管理工作分给不同的角色，从而有效地实现工作权限的传递。

SQL Server 提供了 10 种常用的固定数据库角色来授予数据库用户权限，具体内容如下：

- public：每个数据库用户都属于 public 数据库角色，当尚未对某个用户授予或拒绝对安全对象的特定权限时，则该用户将继承授予该安全对象的 public 角色的权限。
- db_owner：可以执行数据库的所有配置和维护活动。
- db_accessadmin：可以增加或者删除数据库用户、工作组和角色。
- db_ ddladmin：可以在数据库中运行任何数据定义语言（DDL）命令。
- db_securityadmin：可以修改角色成员身份和管理权限。
- db_backupoperator：可以备份和恢复数据库。
- db_datareader：能且仅能对数据库中的任何表执行 select 操作，从而读取所有表的信息。
- db_datawriter：能够增加、修改和删除表中的数据，但不能进行 SELECT 操作。
- db_denydatareader：不能读取数据库中任何表的数据。
- db_denydatawriter：不能对数据库中的任何表执行增加、修改和删除数据操作。

实验 12：在管理平台下对用户进行数据库角色授权

依次按照 school 数据库→安全性→用户的顺序展开文件夹，右击 qq 用户，在弹出的快捷菜单中选择“属性”命令。在展开的数据库用户 qq 的管理平台中，授予 qq 用户 db_owner 的数据库操作权限，如图 5-28 所示。最后，以 qq 用户的身份登录测试权限的效用。

图 5-28　授予 qq 用户系统管理员的权限

实验 13：通过命令形式对用户进行数据库角色授权

（1）为数据库角色添加成员。基本语法为：

```
sp_addrolemember '角色名','用户名'
```

例 1．授予 GUEST 用户具有操作 db_datareader 的数据库角色。

```
Exec sp_addrolemember 'db_datareader','GUEST'
```

（2）为数据库角色删除成员。基本语法为：

```
sp_droprolemember '角色名','用户名'
```

例 2．取消 GUEST 用户具有操作 db_datareader 的数据库角色。

```
Exec sp_droprolemember 'db_datareader','GUEST'
```

（3）查看固定数据库角色。基本语法为：

```
sp_helpdbfixedrole
```

例 3．查看当前数据库的角色。

```
exec sp_helpdbfixedrole
```

3. 应用程序角色

应用程序角色是特殊的数据库角色，用于允许用户通过特定应用程序获取特定数据。应用程序角色不包含任何成员，而且在使用它们之前要在当前连接中将它们激活。激活一个应用程序角色后，当前连接将丧失它所具备的特定用户权限，只获得应用程序角色所拥有的权限。

实验 14：应用程序角色的创建和使用

（1）建立应用程序角色。基本语法为：

```
CREATE APPLICATION ROLE '角色名' [WITH  PASSWORD = '自定义密码'];
```

例 1．建立应用程序角色 FinancialRole，设置登录密码为 123456。

```
CREATE APPLICATION ROLE FinancialRole WITH PASSWORD = '123456';
```

（2）使用应用程序角色。应用程序角色在使用之前必须激活。可以通过执行 sp_setapprole 系统存储过程来激活应用程序角色。在连接关闭或执行系统存储过程 sp_unsetapprole 之前，被激活的应用程序角色都将保持激活状态。虽然应用程序角色旨在由客户的应用程序使用，但同样可以在 T-SQL 批处理中使用它们。系统存储过程 sp_setapprole 的基本语法格式如下：

```
sp_setapprole [ @rolename = ] 'role', [ @password = ] { encrypt N'password' }|
       'password' [ , [ @encrypt = ] { 'none' | 'odbc' } ] [ , [ @fCreateCookie
= ] true | false ]
       [ , [ @cookie = ] @cookie OUTPUT ]
```

其中，[@rolename =] 'role'表示当前数据库中定义的应用程序角色的名称；[@password =] { encrypt N'password' }表示激活应用程序角色所需的密码；[@fCreateCookie =] true | false 指定是否要创建 cookie；[@cookie =] @cookie OUTPUT 指定包含 cookie 的输出参数，只有当 @fCreateCookie 的值为 true 时，才生成 cookie。

在使用 sp_setapprole 系统存储过程参数特性时，为了保证安全性，必须在 sp_setapprole 使用一个选项来返回一个 cookie。这个 cookie 是 sp_unsetapprole 需要的，这样可以禁止用户任意调用 sp_unsetapprole。

以下存储过程演示了如何激活应用程序角色 FinancialRole 并解除这个操作。

例 2．取消 GUEST 用户具有操作 db_datareader 的数据库角色。

```
DECLARE @theCookie varbinary(256)
EXEC sp_setapprole 'FinancialRole', '123456',
     @fCreateCookie = true, @cookie = @theCookie OUTPUT
-- 检查当前用户,该用户应该为 FinancialRole
SELECT USER_NAME()
-- 下面释放被激活的应用程序角色 FinancialRole
EXEC sp_unsetapprole @theCookie
-- 再次检查当前用户,该用户应该为系统默认用户 dbo
SELECT USER_NAME()
GO
```

（3）删除应用程序角色。如果需要删除应用程序角色，可以使用 DROP APPLICATION ROLE 语句。

例 3．

```
DROP APPLICATION ROLE FinancialRole
```

5-4　SQL Server 2005 密码策略和证书

- 了解加密技术的历史，了解对称加密技术，非对称加密技术和数字证书的概念。
- 学习 SQL Server 2005 数据加密技术和加密各级别密钥的层次架构。
- 掌握备份服务主密钥和恢复服务主密钥基本语法。
- 掌握创建、备份、恢复、删除数据库主密钥。
- 掌握创建、修改、删除、SQL Server 2005 数字证书，并可以使用 SQL Server 2005 数字证书加密/解密数据。
- 掌握使用对称密钥加解密 SQL Server 2005 的数据的方法。
- 掌握使用非对称密钥加解密 SQL Server 2005 的数据的方法。

SQL Server 2005 的升级，特别是在安全性方面的升级主要体现在其密码策略和证书的考虑上面，即强化了密码的加密技术设置，这点在以前的 SQL Server 版本中是没有的。

5-4-1　加密技术概述

加密技术源于人类战争中对军事信息的保密需要，古时被称为隐写术（steganography），即通过隐藏消息的存在来保护消息。所谓加密，就是用基于数学方法的程序和保密的密钥对信息进行编码，把计算机数据变成一堆杂乱无章难以理解的字符串，也就是把明文变成密文，如图 5-29 所示。

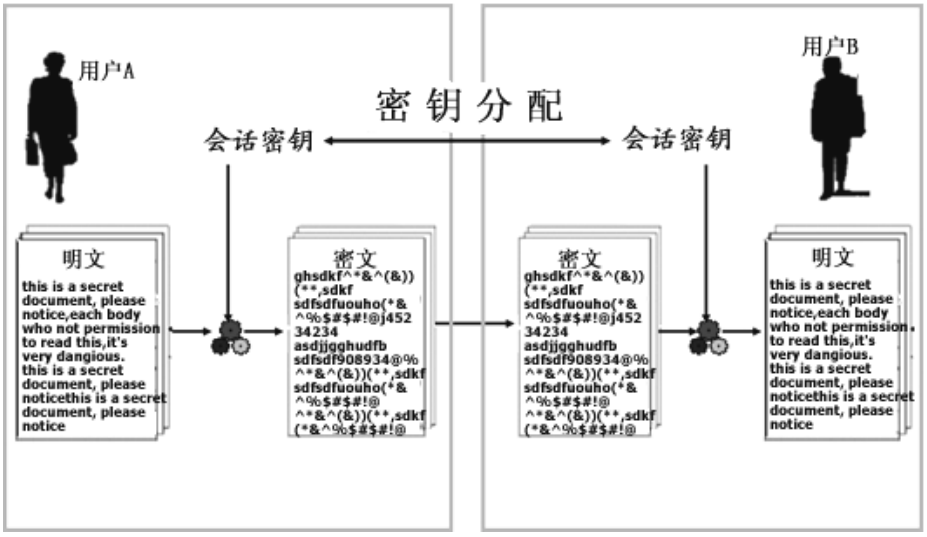

图 5-29　加解密过程示意图

加密技术的发展分为三个阶段：第一阶段在 1949 年之前，此时密码学是一门艺术；第二阶段是在 1949—1975 年，密码学成为科学；第三阶段是在 1976 年以后，密码学开始向公钥密码学方向发展，目前主要的加密技术分为对称加密技术和非对称加密技术两种。

1. 对称加密系统

对称加密系统的基本加密过程是：首先，发送方用自己的私有密钥对要发送的信息进行加密；其次，发送方将加密后的信息通过网络传送给接收方；最后，接收方用发送方进行加密的那把私有密钥对接收到的加密信息进行解密，得到信息明文。对称加密的基本原理如图 5-30 所示。

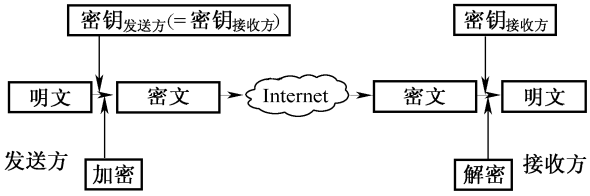

图 5-30　对称加密系统过程示意图

2. 非对称加密系统

非对称加密系统属于公开密钥加密系统，其加密模式过程是：首先，发送方用接收方公开密钥对要发送的信息进行加密；其次，发送方将加密后的信息通过网络传送给接收方；最后，接收方用自己的私有密钥对接收到的加密信息进行解密，得到信息明文。非对称加密的基本原理如图 5-31 所示。

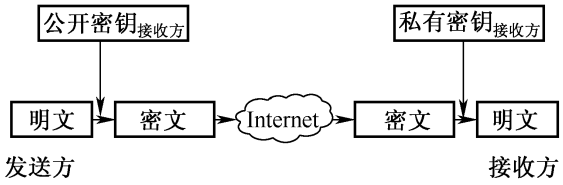

图 5-31　非对称加密系统过程示意图

3. 数字证书

数字证书是标志网络用户身份信息的一系列数据，用来在网络通信中识别通信各方的身份，即在 Internet 上解决“我是谁”的问题，就如同现实中每个人都要拥有一张证明个人身份的身份证或驾驶执照一样，以表明身份或某种资格。

数字证书采用公－私钥密码体制，每个用户拥有一把仅为本人所掌握的私钥，用它进行信息解密和数字签名；同时拥有一把公钥，并可以对外公开，用于信息加密和签名验证。数字证书可用于发送安全电子邮件、访问安全站点、网上证券交易、网上采购招标、网上办公、网上保险、网上税务、网上签约和网上银行等安全电子事务处理和安全电子交易活动。

目前国际流行的数字证书标准是 X.509 数字证书，该证书包含以下内容：

- 证书拥有者的姓名。
- 证书拥有者的公钥。
- 公钥的有限期。
- 颁发数字证书的单位。

- 颁发数字证书单位的数字签名。
- 数字证书的序列号等。

如图 5-32 所示为通过 Internet 查看具体数字证书的过程。

图 5-32　查看 Windows 下的数字证书

5-4-2　SQL Server 2005 数据加密技术

数据是一切软件应用技术研发的基础，是信息技术的根基。但是在 SQL Server 2000 认证保护时代，有太多的实践方法证明可以绕过 SQL Server 2000 直接获取数据，最简单的是通过使用没有口令的 sa 账号。尽管 SQL Server 2005 远比它以前的版本安全，但攻击者还是有可能获得存储的数据。因此，数据加密成为更彻底的数据保护战略，即使攻击者得以存取数据，还不得不解密，因而对数据增加了一层保护。

SQL Server 2000 以前的版本没有内置数据加密功能，若要在 SQL Server 2000 中进行数据加密，不得不买第三家产品，然后在服务器外部作 COM 调用或者是在数据送服务器之前在客户端的应用中执行加密。这意味着加密的密钥或证书不得不由加密者自己负责保护，而保护密钥是数据加密中最难的事，所以即使很多应用中数据已被很强地加密过，数据保护仍然很弱。

SQL Server 2005 通过将数据加密作为数据库的内在特性解决了这个问题。它除了提供多层次的密钥和丰富的加密算法外，最大好处是用户可以选择数据服务器管理密钥。SQL Server 2005 服务器支持的加密算法如下：

1. 对称式加密（Symmetric Key Encryption）

对称式加密方式对加密和解密使用相同的密钥。通常这种加密方式在应用中难以实施，因为用同一种安全方式共享密钥很难。当数据储存在 SQL Server 中时，这种方式很理想，可以让服务器管理它。SQL Server 2005 提供 RC4、RC2、DES 和 AES 系列加密算法。

2. 非对称密钥加密（Asymmetric Key Encryption）

非对称密钥加密使用一组公共/私人密钥系统，加密时使用一种密钥，解密时使用另一种密钥。公共密钥可以广泛地共享和透露。当需要用加密方式向服务器外部传送数据时，这种加密方式更方便。SQL Server 2005 支持 RSA 加密算法以及 512 位、1024 位和 2048 位的密钥强度。

3. 数字证书（Certificate）

数字证书是一种非对称密钥加密，但是，一个组织可以使用证书并通过数字签名将一组公钥和私钥与其拥有者相关联。SQL Server 2005 支持“因特网工程工作组”（IETF）X.509 版本 3（X.509v3）规范。一个组织可以对 SQL Server 2005 使用外部生成的证书，或者可以使用 SQL Server 2005 生成证书。

SQL Server 2005 采用多级密钥来保护它内部的密钥和数据，如图 5-33 所示。

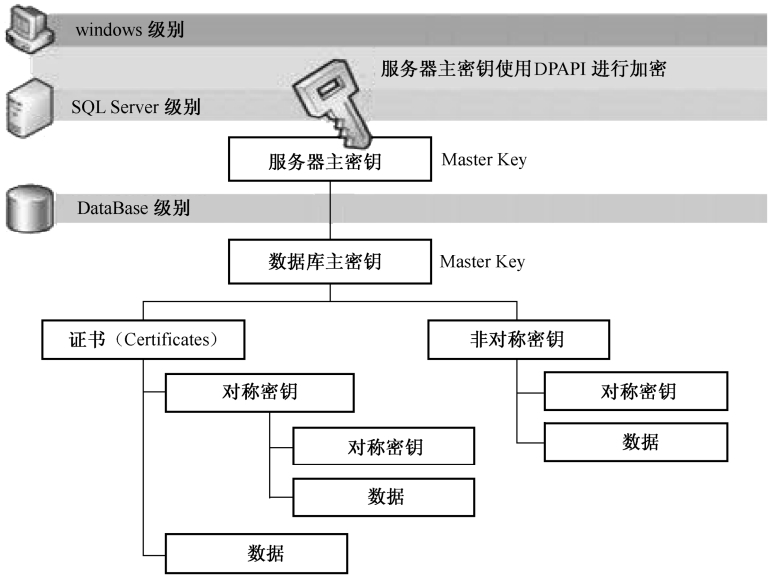

图 5-33　用来加密各级别密钥的层次架构图

图 5-33 中服务主密钥（Service Master Key）保护数据库主密钥（Database Master Keys），而数据库主密钥又保护证书（Certificates）和非对称密钥（Asymmetric Keys），最底层的对称性密钥（Symmetric Keys）被证书、非对称密钥或其他的对称性密钥保护，用户只需通过提供密码来保护这一系列的密钥。

顶层的服务主密钥是在安装 SQL Server 2005 新实例时自动产生和安装的，用户不能删除此密钥，但数据库管理员能对它进行基本的维护，如备份该密钥到一个加密文件，当其危及到安全时更新它，恢复它。

服务主密钥由 DPAPI（Data Protection API）管理。DPAPI 在 Windows 2000 中引入，建立于 Windows 的 Crypt32 API 之上。SQL Server 2005 管理与 DPAPI 的接口，服务主密钥本身是对称式加密，用来加密服务器中的数据库主密钥。

数据库主密钥与服务主密钥的不同之处在于加密数据库中的数据之前，必须由数据库管理员创建数据库主密钥。通常管理员在产生数据库主密钥时会提供一个口令，该口令和服务主密钥一起加密数据库主密钥。如果有足够的权限，用户可以在需要时显示或自动地打开该密钥。在进行 SQL Server 安全性密钥的加密选择时，基本的区别如表 5-3 所示。

表 5-3　SQL Server 安全性密钥的加密选择

保密类型	可以使用的加密方法	多个可能的加密
（数据库）主密钥	密码（1 个或多个），服务主密钥（默认的，但它是可选的，并且可以被删除）	是
证书（私有密钥）	数据库主密钥，密码	否
非对称密钥（私有密钥）	数据库主密钥，密码	否
对称密钥	密码，证书，非对称密钥，对称密钥	是

5-4-3　SQL Server 2005 的服务主密钥

当第一次需要使用数据库主密钥进行加密时，便会自动生成服务主密钥。服务主密钥为 SQL Server 加密层次结构的根。服务主密钥直接或间接地保护树中的所有其他密钥和机密内容。使用本地计算机密钥和 Windows 数据保护 API 对服务主密钥进行加密。该 API 使用从 SQL Server 服务账户的 Windows 凭据中派生出来的密钥。

1. 备份服务主密钥

由于服务主密钥是自动生成且由系统管理的，它只需要很少的管理，并且服务主密钥是 SQL Server 自动生成的，因此它没有对应的 CREATE 和 DROP 语句。服务主密钥可以通过 BACKUP SERVICE MASTER KEY 语句来备份，格式为：

```
BACKUP SERVICE MASTER KEY TO FILE = 'path_to_file'
ENCRYPTION BY PASSWORD = 'password'
```

其中，参数 FILE = 'path_to_file'指定要将服务主密钥导出到的文件的完整路径（包括文件名），此路径可以是本地路径，也可以是网络位置的 UNC 路径；参数 PASSWORD = 'password'是用于对备份文件中的服务主密钥进行加密的密码，此密码应通过复杂性检查。

应当对服务主密钥进行备份，并将其存储在另外一个单独的安全位置。创建该备份应该是首先在服务器中执行的管理操作之一。

例 1.

```
BACKUP SERVICE MASTER KEY TO FILE = 'c:\service_master_key.txt'
ENCRYPTION BY PASSWORD = '123456';
```

2. 恢复服务主密钥

如果需要从备份文件中恢复服务主密钥，使用 RESTORE SERVICE MASTER KEY 语句，基本语法格式为：

```
RESTORE SERVICE MASTER KEY FROM FILE = 'path_to_file'
DECRYPTION BY PASSWORD = 'password' [FORCE]
```

其中，参数 FORCE 指即使存在数据丢失的风险，也要强制替换服务主密钥。但需要注意的是，如果在使用 RESTORE SERVICE MASTER KEY 时不得不使用 FORCE 选项，可能会遇到部分或全部加密数据丢失的情况。

例 2.

```
RESTORE SERVICE MASTER KEY FROM FILE = 'c:\service_master_key.txt'
DECRYPTION BY PASSWORD = '123456'
```

5-4-4　SQL Server 2005 的数据库主密钥

正如每个 SQL Server 有一个服务主密钥一样，每个数据库都有自己的数据库主密钥。但与服务主密钥不同的是，数据库主密钥需要由管理员进行创建，系统不提供主动创建的功能。

1. 创建数据库主密钥

数据库主密钥通过 CREATE MASTER KEY 语句生成，其基本语法格式如下：

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'password'

例 1.

```
USE SCHOOL
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '123456'
```

这个语句创建数据库 school 的主密钥，使用指定的密码加密它，并保存在数据库中。同时，数据库主密钥也在用于服务主密钥加密之后保存在 master 数据库中，这就是所谓的“自动密钥管理”。

2. 备份数据库主密钥

备份数据库主密钥可以通过使用 BACKUP MASTER KEY 语句进行，基本语法格式如下：

```
BACKUP MASTER KEY TO FILE = 'path_to_file'
ENCRYPTION BY PASSWORD = 'password'
```

例 2.

```
BACKUP MASTER KEY TO FILE = 'c:\database_master_key.txt'
ENCRYPTION BY PASSWORD = '123456'
```

3. 恢复数据库主密钥

恢复数据库主密钥使用 RESTORE MASTER KEY 语句，它需要使用 DECRYPTION BY PASSWORD 子句提供备份时指定的加密密码，还要使用 ENCRYPTION BY PASSWORD 子句来确认创建该数据库主密码时的密码，SQL Server 使用它提供的密码来加密数据库主密钥之后保存在数据库中。基本语法格式如下：

```
RESTORE MASTER KEY FROM FILE = 'path_to_file'
DECRYPTION BY PASSWORD = 'password'
ENCRYPTION BY PASSWORD = 'password'
[ FORCE ]
```

建议在创建数据库主密钥之后立即备份数据库主密钥，并把它保存到一个安全的地方。同样，使用 FORCE 语句可能导致已加密数据的丢失。

例 3.

```
RESTORE MASTER KEY FROM FILE = 'c:\database_master_key.txt'
DECRYPTION BY PASSWORD = '123456'
ENCRYPTION BY PASSWORD = '123456'
```

4. 删除数据库主密钥

要删除数据库主密钥，可以使用 DROP MASTER KEY 语句，它删除当前数据库的主密钥。在执行之前，确定在正确的数据库上下文中。

例 4.

```
DROP MASTER KEY
```

5-4-5 SQL Server 2005 的数字证书

当配置好服务主密钥和数据库主密钥后，就可以着手创建 SQL Server 2005 的数字证书。SQL Server 可以创建自签名的 X.509 证书，该证书的实现是通过 CREATE CERTIFICATE 语句来实现的。

1. 创建 SQL Server 2005 数字证书

创建 SQL Server 2005 数字证书的具体语法格式如下：

```
CREATE CERTIFICATE certificate_name [ AUTHORIZATION user_name ]
   { FROM <existing_keys> | <generate_new_keys> }
```

```
    [ ACTIVE FOR BEGIN_DIALOG =  { ON | OFF } ]
<existing_keys> ::=
    ASSEMBLY assembly_name
    | { [ EXECUTABLE ] FILE = 'path_to_file'
        [ WITH PRIVATE KEY ( <private_key_options> ) ]}
<generate_new_keys> ::=
    [ ENCRYPTION BY PASSWORD = 'password']
    WITH SUBJECT = 'certificate_subject_name'
    [ , <date_options> [ ,...n ] ]
<private_key_options> ::=
    FILE = 'path_to_private_key'
    [ , DECRYPTION BY PASSWORD = 'password' ]
    [ , ENCRYPTION BY PASSWORD = 'password' ]
<date_options> ::=
    START_DATE = 'mm/dd/yyyy' | EXPIRY_DATE = 'mm/dd/yyyy'
```

CREATE CERTIFICATE 语句虽然有这么多的选项，但幸运的是大多数时候只用到很少的选项。下面的案例将创建一个用密码保护的数字证书。

例 1.

```
CREATE CERTIFICATE TestCertificate
ENCRYPTION BY PASSWORD = '123456'
WITH SUBJECT = 'This is a test certificate',
START_DATE = '12/28/2009',
EXPIRY_DATE = '1/1/2010';
```

如果不使用 ENCRYPTION BY PASSWORD 子句，证书将使用数据库主密钥来保护。如果不指定 START_DATE 子句，将使用执行此命令的日期来填写证书的 Start Date 字段。

2. 修改 SQL Server 2005 数字证书

修改 SQL Server 2005 数字证书的具体语法格式如下：

```
ALTER CERTIFICATE certificate_name
    REMOVE PRIVATE KEY
    | WITH PRIVATE KEY ( <private_key_spec> [ ,... ] ) |
    WITH ACTIVE FOR BEGIN DIALOG = [ ON | OFF ]
<private_key_spec> ::=
    FILE = 'path_to_private_key'
    | DECRYPTION BY PASSWORD = 'key_password'
    | ENCRYPTION BY PASSWORD = 'password'
```

下面的案例将修改上面案例中所创建的数字证书信息。

例 2. 更改数字证书的密码。

```
    ALTER CERTIFICATE TestCertificate
    WITH PRIVATE KEY
   (DECRYPTION BY PASSWORD = '123456',
    --指定解密私钥所需的密码是 123456
    ENCRYPTION BY PASSWORD = '654321'
    --指定用于对数据库中的证书私钥进行加密的密码。此密码受密码复杂性策略约束
    );
```

3. 删除 SQL Server 2005 数字证书

删除 SQL Server 2005 数字证书的语法较为简单，格式如下：

```
DROP CERTIFICATE certificate_name
```

例 3.

```
DROP CERTIFICATE TestCertificate
```

4. 使用 SQL Server 2005 数字证书加密/解密数据

通过内置的函数 EncryptByCert、DecryptByCert 和 Cert_ID，可以使用证书来加密和解密数据。

（1）Cert_ID 函数。Cert_ID 函数得到指定名字的证书的 ID。格式为：

```
Cert_ID ( 'cert_name' )    --cert_name 为证书的名字
```

例 4.

```
select Cert_ID('TestCertificate')
```

（2）EncryptByCert 函数。EncryptByCert 函数是指使用数字证书的公钥加密数据。只能使用相应的私钥对加密文本进行解密。此类非对称转换比使用对称密钥进行加密和解密的方法开销更大。建议在处理大型数据集（如多个表中的用户数据）时不使用非对称加密。格式为：

```
EncryptByCert ( certificate_ID , { 'cleartext' | @cleartext } )
```

该函数基本参数的含义是：certificate_ID 为通过 Cert_ID 函数得到的证书 ID；cleartext 为要加密的明文，类型为 nvarchar、char、varchar、binary、varbinary 或 nchar。EncryptByCert 函数的返回值是最大大小为 8,000 个字节的 varbinary。

下面的案例使用名为 TestCertificate 的数字证书对某明文进行加密，加密的数据插入表 student 中。经过数字证书加密后的信息如图 5-34 所示。

134	王萍	女	NULL
145	df	男	NULL
156	4545	男	NULL
998	jz?搭嗾&u`o'祯 ?↖^┬菜藺┬...	男	NULL

图 5-34 数字证书加密后的明文信息

例 5.

```
INSERT INTO student(sno,sname)
values( 998, EncryptByCert(Cert_ID('TestCertificate'), '张飞绣花爱美丽') );
```

（3）DecryptByCert 函数。DecryptByCert 函数指用证书的私钥解密数据，基本语法格式为：

```
DecryptByCert (certificate_ID , { 'ciphertext' | @ciphertext }
            [ , { 'cert_password' | @cert_password } ] )
```

其中，参数 ciphertext 是已用证书的公钥加密的数据的字符串，而 cert_password 是用来加密证书私钥的密码。特别需要说明的是，无论是已经加密的字符串信息，还是加密证书的私钥密码，都必须为 Unicode 字符类型，否则解密将出现查询为空值的现象。

实验 15：利用数字证书加密和解密数据

本实验将利用数字证书技术，对临时明文数据进行加密，然后再对加密后的密文信息进行解密，以期达到对数字证书加解密过程的深入理解。

```
--案例：使用证书加密数据。
--首先建立测试数据表
CREATE TABLE tb(ID int IDENTITY(1,1),data varbinary(8000));
```

```
GO
--建立当前数据库的主密钥
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '123456'
--建立证书一，该证书使用数据库主密钥来加密
CREATE CERTIFICATE Cert_Demo1
WITH
  SUBJECT=N'cert1 encryption by database master key',
  START_DATE='2009-12-28',
  EXPIRY_DATE='2010-12-31'
GO
--建立证书二，该证书使用密码来加密
  CREATE CERTIFICATE Cert_Demo2
  ENCRYPTION BY PASSWORD='qianshao123'
  WITH
  SUBJECT=N'cert2 encrption by password',
  START_DATE='2009-12-28',
  EXPIRY_DATE='2010-12-31'
GO
--此时，两个证书已经建立完，现在可以用这两个证书对数据加密
--在对表 tb 做 INSERT 时，使用 ENCRYPTBYCERT 加密
INSERT tb(data)
  SELECT ENCRYPTBYCERT(CERT_ID(N'Cert_Demo1'),
  N'这是证书 1 加密的内容-IamQianShao');
 --使用证书 1 加密
INSERT tb(data)
  SELECT ENCRYPTBYCERT(CERT_ID(N'Cert_Demo2'),
  N'这是证书 2 加密的内容-IamQianShao');
--使用证书 2 加密
--提示：N'是 Unicode 编码的意思，一般来说，英文字符由一个字节组成，但是国际上的字太多了，
因此就用两个字节来表示字符，满足国际化的需要
--已经对数据加密保证了，现在 SELECT 看看
SELECT * FROM tb ;
--对内容进行解密显示
--解密时，使用 DECRYPTBYCERT
SELECT
证书 1 解密=CONVERT(NVARCHAR(50),DECRYPTBYCERT(CERT_ID(N'Cert_Demo1'),data)),
      --使用证书 2 解密时，要指定 DECRYPTBYCERT 的第三个参数，
      --因为在创建时，指定了 ENCRYPTION BY PASSWORD。
      --所以这里要通过这个密码来解密,否则解密失败
      证书 2 解密=CONVERT(NVARCHAR(50),DECRYPTBYCERT(CERT_ID(N'Cert_Demo2'),
                  data,N'qianshao123'))
FROM tb ;
--可以看到，因为第 2 条记录是证书 2 加密的，所以使用证书 1 将无法解密，所以返回 NULL
/*
证书 1 解密                                          证书 2 解密
-------------------------------------------------------------------------
这是证书 1 加密的内容-liangCK                              NULL
NULL                                              这是证书 2 加密的内容-liangCK
```

```
(2 行受影响)
*/GO
--删除测试证书与数据表
DROP CERTIFICATE Cert_Demo1;
DROP CERTIFICATE Cert_Demo2;
DROP TABLE tb;
GO
```

5-4-6 使用对称密钥加解密 SQL Server 2005 的数据

可以通过使用 CREATE SYMMETRIC KEY 语句实现对称密钥的加解密工作，当然也可以使用证书来创建用来在数据库中进行加密和解密的对称密钥。具体语法格式如下：

```
CREATE SYMMETRIC KEY key_name [ AUTHORIZATION owner_name ]
    WITH <key_options> [ , ... n ]
    ENCRYPTION BY <encrypting_mechanism> [ , ... n ]
<encrypting_mechanism> ::=
    CERTIFICATE certificate_name |
    PASSWORD = 'password' |
    SYMMETRIC KEY symmetric_key_name |
    ASYMMETRIC KEY asym_key_name
<key_options> ::=
    KEY_SOURCE = 'pass_phrase' |
    ALGORITHM = <algorithm> |
    IDENTITY_VALUE = 'identity_phrase'
<algorithm> ::=
DES | TRIPLE_DES | RC2 | RC4 | DESX | AES_128 | AES_192 | AES_256
```

1. 加密算法

加密算法定义了如何使用密钥对数据进行加密。可以提供的加密算法有 9 种，分别是 DES、TRIPLE_DES、RC2、RC4、RC4_128、DESX、AES_128、AES_192 和 AES_256，它们的速度和强度各自不同。如 DES 算法为美国政府 1976 年使用的加密算法，目前计算机已经可以在 24 小时将其破解。高级加密标准（AES，也称为 Rijneqel 算法）于 2001 年 11 月获得美国国家标准和技术协会（NIST）的批准。这些算法名称中的 128、192、256 指的是密钥的长度（单位为位）。在目前的 SQL Server 中，最强的加密算法是 AES_256。

注意：加密算法的操作系统使用范围。SQL Server 是利用 Windows 提供的加密算法，因此 SQL Server 无法支持使用在 Windows 中没有安装的加密算法。Windows XP 和 Windows2000 不支持 AES。

在 SQL Server 加密过程中，通过 ALGORITHM 参数来具体决定使用什么样的加密算法。

同 CREATE CERTIFICATE 语句一样，CREATE SYMMETRIC KEY 语句相当灵活。多数情况下，只需使用少量的选项。下例的案例将使用前节中创建的证书来加密并创建一个对称密钥。

例 1.

```
CREATE SYMMETRIC KEY TestSymmetricKey
WITH ALGORITHM = TRIPLE_DES
ENCRYPTION BY CERTIFICATE TestCertificate;
```

2. 使用对称密钥加密数据

SQL Server 使用下面的函数进行对称密钥加密：EncryptByKey、DecryptByKey 和 Key_GUID。

（1）Key_GUID 返回特定对称密钥的 GUID。语法为：

```
Key_GUID( 'Key_Name' )
```

（2）EncryptByKey 为对称密钥加密函数，但要使用对称密钥，首先要通过“OPEN SYMMETRIC KEY 对称密钥名”打开对称加密密钥。在进行解密后，通过“CLOSE SYMMETRIC KEY 对称密钥名”关闭密钥，及时关闭密钥是一种良好的编程习惯。语法为：

```
EncryptByKey( key_GUID, {'cleartext' | @cleartext }[,{add_authenticator | @add_authenticator} , { authenticator | @authenticator } ] )
```

其中，Key_GUID 是对称密钥的 GUID，cleartext 为明文，Add_authenticator 和 authenticator 指示是否使用验证器来禁止对加密字段进行整个值替换。

（3）DecryptByKey 为对称密钥解密函数，语法为：

```
DecryptByKey({'ciphertext'|@ciphertext}[,add_authenticator,{authenticator|@authenticator}])
```

DecryptByKey 做 EncryptByKey 相反的事情，它解密先前使用 EncryptByKey 加密的数据。

下面的代码演示使用对称密钥来加密和解密。

例 2．本案例将使用对称密钥加密数据，对称密钥又使用证书来加密。

```
create database mydb
--创建测试数据表 tb，注意需要加密的数据项最好设置为 varbinary 类型，因为任何字符串加密后都将以 varbinary 形式保存在计算机的磁盘中。这也是实验成败的关键点
use mydb
CREATE TABLE tb(ID int IDENTITY(1,1),data varbinary(8000));
GO
--建立证书，该证书用于加密对称密钥
CREATE CERTIFICATE Cert_Demo
ENCRYPTION BY PASSWORD=N'qianshao123'
WITH
  SUBJECT=N'cert encryption by password',
  START_DATE='2010-01-11',
  EXPIRY_DATE='2010-01-20'
GO
--建立对称密钥
CREATE SYMMETRIC KEY Sym_Demo
WITH
   ALGORITHM=DES  --使用 DES 加密算法
ENCRYPTION BY CERTIFICATE Cert_Demo --使用 Cert_Demo 证书加密
GO
--要使用 Sym_Demo 对称密钥。必须使用 OPEN SYMMETRIC KEY 来打开它
OPEN SYMMETRIC KEY Sym_Demo
   DECRYPTION BY CERTIFICATE Cert_Demo
      WITH PASSWORD=N'qianshao123'
--插入加密数据
INSERT tb(data)
SELECT ENCRYPTBYKEY(KEY_GUID(N'Sym_Demo'),N'这是加密的数据,能显示出来吗?')
```

```
--关闭密钥
CLOSE SYMMETRIC KEY Sym_Demo
--插入完加密数据，现在使用 SELECT 来查询一下数据
SELECT * FROM tb
GO
--现在来解密此数据。同样，还是要先打开对称密钥
OPEN SYMMETRIC KEY Sym_Demo
   DECRYPTION BY CERTIFICATE Cert_Demo
     WITH PASSWORD=N'qianshao123'

SELECT CONVERT(NVARCHAR(50),DECRYPTBYKEY(data)) --这里可见，数据已经解密出来了
FROM tb
--完成事务后及时关闭密钥是一种良好的编程习惯
CLOSE SYMMETRIC KEY Sym_Demo
GO

--删除测试
DROP SYMMETRIC KEY Sym_Demo
DROP CERTIFICATE Cert_Demo
DROP TABLE tb
```

5-4-7 使用非对称密钥加解密 SQL Server 2005 的数据

非对称密钥加密使用一组公共/私人密钥系统，加密时使用一种密钥，解密时使用另一种密钥。公共密钥可以被广泛地共享和透露。当需要用加密方式向服务器外部传送数据时，这种加密方式更加方便。SQL Server 2005 支持的非对称加密算法有 RSA_512、RSA_1024 和 RSA_2048，这些算法的差异在于私钥的长度。与对称密钥不同的是，非对称密钥加密时并不需要数字证书的辅助加密，就可以实施对数据进行加密，主要差异如图 5-35 所示。

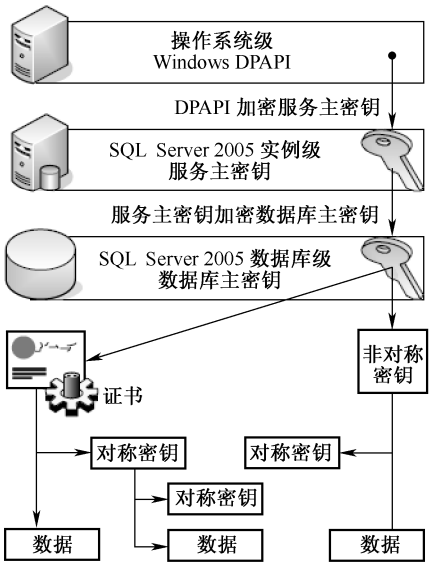

图 5-35 非对称密钥和对称密钥之间的主要差异对照图

具体语法格式如下：

```
CREATE ASYMMETRIC KEY Asym_Key_Name
[ AUTHORIZATION database_principal_name ]
{FROM <Asym_Key_Source> |WITH ALGORITHM ={RSA_512 | RSA_1024 | RSA_2048 }}
ENCRYPTION BY PASSWORD = 'password'
<Asym_Key_Source>::=FILE = 'path_to_strong-name_file'
|EXECUTABLE FILE = 'path_to_executable_file'  |ASSEMBLY Assembly_Name
```

下面的代码演示使用非对称密钥加密和解密的全过程。

例 1．本案例将使用非对称密钥加密数据，同时介绍利用非对称密钥进行解密的全过程。

```
--建立 testdb 数据库
create database testdb
--1．创建非对称密钥
CREATE ASYMMETRIC KEY asy_TestKey WITH ALGORITHM = RSA_1024
ENCRYPTION BY PASSWORD = '123456'
GO
--查询系统中目前非对称密钥的情况
SELECT * FROM sys.asymmetric_keys

--2．创建示例表
USE testdb
CREATE TABLE test( EmpID int, Title nvarchar(50), Salary varbinary(500))
GO

--3．向表中插入数据，并对 Salary 列的数据进行加密
INSERT INTO test VALUES
(1, 'CEO',EncryptByAsymKey(AsymKey_ID('asy_TestKey'),'20000'))
INSERT INTO test VALUES
(2, 'Manager',EncryptByAsymKey(AsymKey_ID('asy_TestKey'),'10000'))
INSERT INTO test VALUES
(3, 'DB Admin',EncryptByAsymKey(AsymKey_ID( 'asy_TestKey'),'5000'))
GO

--4．查看表中存放的数据
SELECT * FROM test

--5．解密被加密了的数据列
SELECT EmpID, Title,
CONVERT(varchar(60),DecryptByAsymKey(AsymKey_Id('asy_TestKey'),
Salary,N'123456'))
FROM test
```

5-5 SQL Server 2005 的安全性实训

- 通过 SQL Server 登录账户管理实验，掌握和深入理解登录账户的概念。
- 通过 SQL Server 数据库用户管理，了解数据库用户的概念。
- 通过用户安全登录及授/收权实验，了解权限的内涵以及授权和授权的过程。
- 通过对服务主密钥、数据库主密钥、数字证书的加密体系的综合实训，旨在学习者提高在 T-SQL 下进行数据安全加密的综合应用能力。

5-5-1 SQL Server 登录账户管理

本次实验的主要目的是通过 T-SQL 脚本完成建立 Windows 用户和 SQL Server 用户的实验。

1. 使用 SQL 语言创建 SQL Server 登录账户

（1）语法。

```
sp_addlogin '用户名', '密码'[,'登录用户使用的默认数据库']
```

（2）创建一个登录账户：名为 u3，密码为 u3，使用的默认数据库为 school。

```
EXEC sp_addlogin 'u3','u3','school'
```

2. 查看登录账户

```
exec sp_helplogins
```

3. 修改登录账户

（1）修改默认的数据库，系统存储过程：sp_defaultdb。

```
例：EXEC sp_defaultdb 'u3','school'
```

（2）修改默认的语言，系统存储过程：sp_defaultlauguage。

```
例：EXEC sp_defaultlanguage 'u3', 'french'
```

（3）修改登录密码，系统存储过程：sp_password。

```
例：EXEC sp_password 'u3', '1234','u3'
```

4. 删除登录账户

语法：

```
系统存储过程：sp_droplogin '账户名'
例：EXEC sp_droplogin 'u3'
```

5-5-2 SQL Server 数据库用户管理

本实验的主要目的是通过 TSQL 脚本完成使用 SQL 语句创建、查看、删除数据库用户。

1. 使用 SQL 语句创建数据库用户

（1）语法：sp_grantdbaccess '登录名' [,'用户名']

例：在数据库 school 中创建用户 qq，对应的登录账号是 qq。

```
use school
EXEC sp_addlogin 'qq','qq','school'
exec sp_grantdbaccess 'qq', 'qq'
```

（2）说明。

1）在执行本存储过程前，登录名必须已经存在。

2）一般情况下，登录名和用户名相同，所以第 2 个参数通常省略。

3）在执行本存储过程前，首先确认当前使用的数据库是要增加用户的数据库。

2. 使用 SQL 语句查看数据库用户

```
sp_helpuser
```

3. 使用 SQL 语句删除数据库用户

```
语法：sp_revokedbaccess '用户名'
例：EXEC sp_revokedbaccess 'qq'
```

5-5-3　用户安全登录及授/收权实验

1. 实训任务 1：授权与收权

按照下列代码编写的要求，在 Windows 操作系统或 school 数据库中建立若干用户，通过对这些用户的授权和授权，了解通过命令行的方式是如何实现用户安全登录及授/收权工作的。

```
------------------------实验名称：用户安全登录及授/收权实验-----------------
第一步，建立若干 SQL 登录用户（U1—U7）。
第二步，在 school 数据库用户下面允许 u1,u2 为合法用户，同时执行下列的代码：
       GRANT  INSERT  ON  SCORE  TO U1    WITH  GRANT  OPTION
请测试：
insert into score values('122','234',45)
--能否成功?
GRANT  SELECT  ON  Student  TO  U1
GRANT ALL PRIVILEGES ON  Student TO U2, U3
GRANT  SELECT  ON    SC  TO  PUBLIC
GRANT  UPDATE(Sno), SELECT  ON   Student TO  U4
--请分别叙述上述四句话的含义是什么。
第三步，建立角色 QQ，并且将 u2,u1 分配到该 QQ 名下。
 GRANT  INSERT  ON  SCORE  TO QQ WITH  GRANT  OPTION
 再查看 u1 和 u2 是否已经具有了插入数据的能力?
 insert into score values('122','234',45)
第四步，执行下面的代码。
    GRANT  CREATE TABle  TO  U6     -- U6 用户的权限是什么?
第五步，收回权限实验。
（1）REVOKE  select  ON  SCore  FROM  PUBLIC
（2）REVOKE  select(birthday)  ON  teacher  FROM  U2
（3）REVOKE ALL PRIVILEGES ON  Student TO U2, U3,u4,u5
--上述三句话的意思是什么?
第六步，权限拒绝。
（1）DENY SELECT, INSERT, UPDATE, DELETE ON score TO u1
（注意：如果拒绝，必须收回所有授权）
```

```
--上述意思是什么?
（2）exec sp_grantdbaccess 'u7','we'
为 Microsoft? SQL Server? 登录或 Microsoft Windows NT? 用户或组在当前数据库中添加一个安全账户，并使其能够被授予在数据库中执行活动的权限。
```

2. 实训任务 2：课堂测试

本实训的要求是：假设在 school 数据库下建立下面的用户 A1、A2、A3、A4、A5、A6，并需要完成如下操作：

（1）建立上面的用户，并且指定默认数据库为 school。同时要求将所有的用户添加到 school 数据库下。

（2）A1、A2 访问 student 表，A3、A4 访问 score 表，A5、A6 访问 course 表。

（3）建立角色 qq，并将权限设定为查询 student；传播授权给 A5 和 A6 用户。

（4）A1 允许对 student 表的 sno 属性进行插入操作，并对 sname 属性进行修改操作，A2 允许对 score 表进行所有的权限操作。

（5）拒绝用户操作实验：拒绝 A3 对 student 表的 sno 属性进行插入操作；拒绝 A3 对 student 表的 sname 属性进行修改操作；拒绝 A4 对 score 表进行所有的权限操作。

（6）收回 A1 的所有权限。

5-5-4 了解数据库加密体系结构

通过本实验，提高在 T-SQL 下进行服务主密钥、数据库主密钥、数字证书的加密体系的综合理解能力。

```
--************************一、建立并实现服务主密钥************************
--准备工作阶段，创建测试数据库 TestDB
--1)备份服务主密钥
backup service master key to file='c:\smk.bak'
encryption by password='p@ssw0rd'
 --2)生成新的主密钥
Alter service master key regenerate
 --3)从备份文件还原服务主密钥
Restore service master key from file='c:\smk.bak'
decryption by password='p@ssw0rd'
 --************************二、数据库主密钥************************
--1)为数据库创建数据库主密钥
create master key encryption by password='p@ssw0rd'
go
--2)查看数据库加密状态
select [name],is_master_key_encrypted_by_server
from sys.databases where name='TestDB'
--3)查看数据库主密钥的信息
select * from sys.symmetric_keys
--4)备份数据库主密钥
backup master key
```

```
    to file='c:\testdbkey.bak' encryption by password='p@ssw0rd'
    --5)删除服务主密钥对数据库主密钥的保护，创建非对称密钥成功，自动使用服务主密钥解密并使用该数据库主密钥
    create asymmetric key asy_Testkey1 with algorithm=RSA_1024
    go
    --删除服务主密钥对数据库主密钥的保护
    alter master key
    drop encryption by service master key
    go
    --查看数据库加密状态
    select [name],is_master_key_encrypted_by_server
    from sys.databases where name='TestDB'
    --创建非对称密钥失败，因为数据库主密钥未打开
    create asymmetric key asy_Testkey2 with algorithm=RSA_1024
    go
    --打开数据库主密钥
    open master key decryption by password='p@ssw0rd'
    select * from sys.openkeys
    go
    --创建非对称密钥成功
    create asymmetric key asy_Testkey2 with algorithm=RSA_1024
    go
    --恢复服务主密钥对数据库主密钥的保护
    alter master key
    add encryption by service master key
    close master key
    go
    --*******************************三、数字证书***************************
    --1)创建自签名证书
    create certificate cert_Testcert
    encryption by password='p@ssw0rd'
    with subject='TestCert1',
    start_date='1/31/2006',
    expiry_date='1/31/2008'
    go
    select * from sys.certificates
    --2)从文件导入证书
    Create certificate cert_TestCert2 From file='c:\MSCert.cer'
    Go
    --3)备份导出证书和密钥
    backup certificate cert_Testcert to file='c:\Testcert.cer'
    with private key
    (decryption by password='p@ssw0rd',
    file='g:\TestCert_pvt',--私钥
    encryption by password='p@ssw0rd')
    go
```

```
    --4)使用证书加解密数据。加密:使用证书的公钥
    declare @cleartext varbinary(200)
    declare @cipher varbinary(200)
    set @cleartext=convert( varbinary(200),'Test text string')
    set @cipher=EncryptByCert(Cert_ID('cert_TestCert'),@cleartext)
    select @cipher
    --解密:使用证书的私钥
    select convert(varchar(200),DecryptByCert(Cert_ID('cert_TestCert'),@cipher,
N'p@ssw0rd')) as [cleartext]
    --5) 删除证书私钥
    alter certificate cert_TestCert
    remove private key
    go
    --加密成功
    declare @cleartext varbinary(200)
    declare @cipher varbinary(200)
    set @cleartext=convert( varbinary(200),'Test text string')
    set @cipher=EncryptByCert(Cert_ID('cert_TestCert'),@cleartext)
    select @cipher
    --解密失败:因为私钥被删除
    select convert(varchar(200),DecryptByCert(Cert_ID('cert_TestCert'),@cipher,
N'p@ssw0rd')) as [cleartext]
```

本章考纲

- 了解数据库安全性的产生过程和安全措施的 5 个级别，可以区分 Windows 认证模式和 SQL Server 混合认证模式的区别。
- 掌握用户身份认证，主体和安全对象的内涵，如何建立 Windows 认证模式下的用户登录，如何建立 sa 用户登录和 SQL Server 用户登录；掌握通过命令方式授权 Windows 用户及 SQL Server 用户登录账户，学习查看、修改和删除 SQL Server 登录账户信息。
- 掌握用户和模式的分离，以及执行上下文概念内涵；掌握通过管理控制平台及命令行对用户进行授权与收权。
- 掌握用户的角色概念，掌握对用户进行服务器角色授权技术，掌握通过命令形式对用户进行数据库角色授权。
- 学习应用程序角色的创建和使用。
- 了解加密技术的历史，了解对称加密技术、非对称加密技术和数字证书的概念；学习 SQL Server 2005 数据加密技术和加密各级别密钥的层次架构；掌握备份服务主密钥和恢复服务主密钥的基本语法；掌握创建、备份、恢复、删除数据库主密钥；掌握创建、修改、删除、SQL Server 2005 数字证书，并可以使用 SQL Server 2005 数字证书加密/解密数据；掌握使用对称密钥和非对称密钥加解密 SQL Server 2005 的数据的方法。

一、填空题

1．为了保护数据库，防止恶意的滥用，可以在从低到高的 5 个级别上设置各种安全措施，分别是_____、_____、_____、_____和_____。

2．如果一个用户要访问 SQL Server 数据库中的数据，必须经过三个级别的认证过程，分别是_____、_____和_____。

3．SQL Server 的用户有两种类型，分别是_____和_____。

4．服务器的登录用户 sa 是_____用户，用于创建其他登录用户和授权。

5．SQL Server 2005 级别对应的主体分别包括_____、_____和_____。

6．我们可以利用系统存储过程_____实现 Windows 用户登录授权；系统存储过程_____查看 SQL Server 登录账户；系统存储过程_____可以更改 SQL Server 登录用户的登录密码。

7．用户权限的类别包括三种类别：_____、_____和_____。

8．SQL Server 2005 中，角色分为 3 种：_____、_____和_____。

9．SQL Server 2005 服务器支持的加密算法如下：_____、_____和_____。

二、简答题

1．简述数据库的安全性的概念。

2．SQL Server 为什么一般推荐使用 Windows 身份验证模式？

3．混合认证模式的优点是什么？

4．配置 sa 用户登录数据库系统时候无法登录应如何处理呢？

5．在授权的过程中，如果使用了 WITH GRANT OPTION 参数有什么效果？

6．简述数字证书的概念。