FoxTalk 2.0
Solutions for Microsoft® Visual FoxPro® Developers

- 体验刊 -

更多译文见 http://foxer.xicp.net

语言的战争 — Mike Gunderloy 著 Fbilo 译

Page.3

编程语言的孰优孰劣，自古以来都是程序员们津津乐道的话题，现在让我们来听听这位掌握 50 多种语言的 Mike Gunderloy 对他的孩子们（语言）的评价吧。

VFP 和 VB.Net 的数据管理 — Les Pinter 著 CY 译

Page.6

VFP 是一个极其优秀的开发工具，但这并不妨碍我们走进外面精彩的开发世界，好，现在就教你一手，看看 VFP 是怎么蹦到 VB.NET 的。

狐眼细看 MySQL — Dan Jurden 著 Yasur 译

Page.16

今天，MySQL 已经引起了极大的关注。那么，究竟有什么值得兴奋的呢？在这个系列的文章中，Dan Jurden 将带着我们探索 MySQL 从基础到高级的内容。他还将指出 MySQL 的一些优点和缺点，并做了大量与 SQL Server 的比较。最后，作为一个程序员，当你开始开发你的下一个客服/服务器应用程序的时候，你将会拥有足够的知识去作出明智的决定：选择哪一个数据库引擎才能最好的为你工作。(MySQL 的文章总共 4 篇连载，这是第一篇)

把你的爪子从我的资料上拿开 – Doug Hennig 著 CY 译

Page.20

你需要防止别人对你的 FoxPro 数据的未授权访问吗？ Doug Hennig 注意到 Xitech 的 Cryptor，它提供了一个很容易的办法来加密文件，并将它们加密保留在磁盘上，却可以让你在应用程序里以普通表文件一样来访问。

来自 VFP 开发组的 Tips – 微软开发团队成员著 LQL.NET 译

Page.32

雷德蒙得微软开发园区的 41 号楼这些日子里非常的繁忙，Visual FoxPro 开发组正在日以继夜的忙于 VFP 9 的新功能、修正 Bug、写文档和示例代码、以及穷举测试。百忙之中，他们也偷空在 Foxtalk 2.0 上开了一个专栏，每期介绍一些 VFP 新功能的使用方法，这一期讲的就是关于 VFP 9 中 SQL Select 语句的一些新功能...

语言的战争

原著：Mike Gunderloy

翻译：Fbilo

编程语言的孰优孰劣，自古以来都是程序员们津津乐道的话题，现在让我们来听听这位掌握 50 多种语言的 Mike Gunderloy 对他的孩子们（语言）的评价吧。

把几个程序员关在一间屋子里，要不了多久他们之间就会发生一场语言的战争。在 Internet 的论坛上也很容易找到这样的内容：“你在一些 Delphi 程序里面根本看不到什么 OO 的内容”，或者“C++ 是一个设计过度的灾难”、或者“C#只是 Microsoft 想用来杀死 Java 的东西”、或者“你不是一个真正的程序员，或者你最好使用一种真正的语言”。

这些讨论中通常都充斥着标语一样的口号“Linux 好，Microsoft 不行”，“Scripting 好，IDEs 不行”、“Java 好，.NET 不行”。当然，这样的东西叫得越多，来自语言战争中对立一方的反击就越多。争吵的最后，本来想要认真讨论问题的人会发现最后得到的只是一些意气用事的混战。

对我来说，语言战争开始的时候就开始退出了。这并不是说我认为为一个特定的项目选择哪种语言的问题不重要，而是我认为与试图找出“最好”的工具相比，还有一种更好的途径来作出选择。

选择正确的工具

在我的职业生涯中，我曾经使用过至少 50 种不同的计算机语言（牛人啊！），有的常见而有的默默无闻。不用太惊讶，我能够任何一种语言写程序。当然，从某种观点来看，这是一个琐碎的结果：所有我知道的计算机语言都是完善的，就是说，任何一种语言，都能够解决任何可用计算机解决的问题，不过有时可能会需要花上一千年就是了。

我认为，那些语言的斗士们犯的错误就是认为既然每一种语言都是不同的，那么其中必然有一种最好的。这显然是愚蠢的。问哪种语言最好就好像是问哪种扳手最好一样没有意义。最小的扳手和最大号的扳手都有最适合使用它们的场合。所以，关于计算机语言最有趣的事情不是哪一种最好，而是哪种最适合在某种特定的环境条件下的某个特定的任务。

从任务的角度来看，事情就简单多了：Visual Foxpro 长于快速建立 Windows 用户界面方面（当然，它并非是唯一长于这方面任务的工具），但在每个字节都需要斤斤计较的嵌入式系统开发上就不那么有用了。另一方面，即使是那些最

坚定的支持者也不会支持用他们自己的语言来一下子就开发出一个快速的 Windows 数据中心应用程序。C++、Perl、List、汇编...每一种都有它自己的问题领域，也有它最擅长的地方。

不要忘记了“环境”这个词。当你面向实现数据中心的应用程序的时候，你也许会发现 Visual Foxpro 和 Delphi 都能快速的开发。在这种情况下，除了简单的找出直接的需求以外，你还需要发掘出其它的事实以作出语言的选择。单向的部署是否重要？你的开发组是否已经完全掌握了 VFP？与只有一种支持的来源相比，你对不同的销售商之间的差异有什么想法？当缺乏需求方面足够的事实来指引的时候，类似这样的软件问题能够帮助你选择合适的语言。

没有哪种语言是一个孤岛

当你试图为你的下一个任务选择一种计算机语言的时候，你需要想想所有相关的事实，而不是只看哪种语言最适合这个任务。这里是应该能帮助你摸清情况的 10 点问题。

1、这种语言适合你手头的工作吗？

2、你的开发团队对于这门语言是否有足够的经验？如果不是，那么你要怎样处理培训和学习的问题？

3、这种语言是否能在尚余的项目开发生命周期内与你正在使用的工具很好的集成？想想需求的搜集、结构和建模、源代码控制、自动化编译过程、以及测试问题。你是否必须要在新工具上投入资金？

4、这种语言是否有多个厂商？如果是的话，你要怎么去选择哪一个？如果不是，你是否会对老是跟这么一个厂商交易感到舒服？

5、这种语言是否有一个用户团体？你是否能找到新闻组、用户组、和书并得到帮助？

6、将来你是否需要将你的代码从一个平台导入到另一个平台？如果那样的话会有多大的难度？

7、语言中是否有集成的钩子（hook）？你是否能够用它来与一个数据库、调用库、建立一个 Web Service 等相交互，或者你的项目中余下的内容相关联？

8、这种语言工具是否能用你已有的硬件来运行？

9、这种语言是否提供一种项目开发跟踪日志？它是否能处理需求？它是否能处理你应用程序的工程量问题？

10、也许最重要的是，为什么你要为这个项目考虑使用这种语言？是否是因为它是你最拿手的那种语言？还是你的老板刚刚跟语言的销售商吃了顿饭？使用这种语言是否有什么明确的技术问题？

记住，你的工作是开发能够满足一系列需求的计算机软件。你的计算机不会关心这个软件是用什么语言写成的（除了那种语言本身也属于需求的情况，比如，某些客户可能会要求软件必须以 ADA 写成）。换几种语言不会使你得到额外的奖励，老是使用同一种语言也一样。

新的思考方法：

不管你站在语言战争中的哪一边，我都建议你不要太专注于空谈理论。从 Lisp 到 Visual Foxpro 到 C++ 再到

Python....计算机语言有着各种各样的风格（参见 http://www.catseye.mb.ca/esoteric/index.html）。为了保持你的技术的不断提高，你应该努力来增加在你尚未使用过的语言方面的知识。务实程序员们（http://www.pragmaticprogrammer.com/loty/index.html）甚至建议每年学习一门新的语言，我认为这倒是个好主意。他们指出，不同的语言以不同的途径解决同一个问题，所以，通过学习多种语言，你可以避免陷入困境。当下一次为一个新项目选择语言的时候，你能够为自己作出更明智的选择，并避免陷入到语言战争的争论中去。另一个好处是成为一个更好的程序员！

你有自己的语言战争的故事吗？或者你已经做到了每年学习一门新语言，而不管你用不用的上？写信给我分享吧！

Mike Gunderloy，MCSE,MCSD,MCDBA，是 MCP 杂志的一位投稿编辑。他还是一个众多数据库和开发方面书籍和文章的作者。你可以用这个地址跟 Mike 联系：mikeg1@larkfarm.com

VFP 和 VB.Net 的数据管理

原著: Les Pinter

翻译: CY

注：这里是摘抄自 Les Pinter 的白皮书《从 VFP 迁移到 VB.Net》，这将会在本月稍晚的时候在 Montreal 的 DevTeach 上演讲。

在VB.NET 里没有哪一个方面会比它处理数据更令人失望的。在某些小数据量下，数据处理机制的效率还是可以接受的。带宽通常是所关心的事，并且在数据处理机制里的大多数变量是根据需要与小数据一样在线路上移动。数据可以是来自 MS SQLServer 或是其他任何地方。SQLServer（或者，MSDE，支持以开发和测试为目的的少量连接的开发工具）在 VS 文档里被大大地宣传，这是可以理解的。难以容忍的是不再能够方便的使用非 SQL 数据源。其实，这和使用 SQL 是一样的困难。

因为这样，我们可以推荐给客户的成本优势的事实-- FoxPro 的数据访问对程序而言是低廉的，已经不在了。你使用 OLEDB 数据源所获得的仅有优势是你不再需要支付许可费给 MS。但是 OLEDB 的性能与纯 FoxPro 的数据访问相比是糟糕的。于是这个优势也不在了。其实，我很难见到在.NET 应用里使用 FoxPro 表的理由。这里表达了你最好还是去做的事；它是拙劣的,所以我在这里不会使用它。

FoxPro

这里是你在 FoxPro 里怎么处理数据的：

```
USE
```

有问题吗？

事实上，有些小问题。为了在多用户环境下使用数据，你必须设置 SET EXCLUSIVE OFF 和 SET MULTILOCKS ON，然后在修改前发出 CursorSetProp('Buffering',5)，并使用 TableUpdate(.T.) 或 TableRevert(.T.)来分别保存或放弃更改，然后再设置缓冲为 1。但是，这是对应于 DBF 访问，除非你想去检查是否有其他用户已经修改了你所编辑的用户记录，在大多数情况下这是百万分之一的机会。

SQLServer(或 MSDE)引起另一个问题。从一个 SQL 表获取数据需要建立一个连接，执行一个 SELECT 语句，然后关闭连接。这里是代码：

```
Handle = SQLStringconnect( ;
   "Driver={SQL Server};Server=(local);Database=NorthWind;UID=sa;PWD=;")

   SQLExec(Handle,"SELECT * FROM CUSTOMERS", "MyCursor" )
```

```
SQLDisconnect(0)
```

这把返回的记录保存在一个名为“MyCursor”的游标里。使用 VFP8 新的 CursorAdapter 类，你可以不需要代码来完成同样的事。

回传更改会更复杂些。SQL 所寻找的是一个 INSERT 或 UPDATE 命令。下面是写给你的两个过程：

列表 1：创建你自己的 SQL INSERT 和 UPDATE 命令

```
PROCEDURE SaveRecord
    PARAMETERS pTable, pKeyField, pAdding
    IF pAdding
        THIS.InsertRecord ( pTable, pKeyField )
    ELSE
        THIS.UpdateRecord ( pTable, pKeyField )
    ENDIF
ENDPROC
PROCEDURE InsertRecord
    LPARAMETERS pTable, pKeyField
    cExpr = THIS.BuildInsertCommand ( pTable, pKeyField )
    DO CASE
    CASE THIS.AccessMethod = [SQL]
        lr = SQLExec ( THIS.Handle, cExpr )
        IF lr < 0
            msg = [Unable to insert record; command follows:] + CHR(13) + cExpr
            MESSAGEBOX( Msg, 16, [SQL error] )
        ENDIF
    ENDCASE
ENDPROC
PROCEDURE UpdateRecord
    LPARAMETERS pTable, pKeyField
    cExpr = THIS.BuildUpdateCommand ( pTable, pKeyField )
    _ClipText = cExpr
    DO CASE
    CASE THIS.AccessMethod = [SQL]
        lr = SQLExec ( THIS.Handle, cExpr )
        IF lr < 0
            msg = [Unable to update record; command follows:] + CHR(13) + cExpr
            MESSAGEBOX( Msg, 16, [SQL error] )
```

```
            ENDIF
        ENDCASE
ENDPROC
FUNCTION BuildInsertCommand
    PARAMETERS pTable, pKeyField
    Cmd = [INSERT ] + pTable + [ ( ]
    FOR I = 1 TO FCOUNT()
        Fld = UPPER(FIELD(I))
        * Can't deal with GENERAL fields, so ignore them
        IF TYPE ( Fld ) = [G]
            LOOP
        ENDIF
        Cmd = Cmd + Fld + [, ]
    ENDFOR
    Cmd = LEFT(Cmd,LEN(Cmd)-2) + [ } VALUES ( ]
    FOR I = 1 TO FCOUNT()
        Fld = FIELD(I)
        IF TYPE ( Fld ) = [G]
            LOOP
        ENDIF
        Dta = ALLTRIM(TRANSFORM ( &Fld ))
        Dta = CHRTRAN ( Dta, CHR(39), CHR(146) ) && get rid of single quotes       in the data
        Dta = IIF ( Dta = [/ /], [], Dta )
        Dta = IIF ( Dta = [.F.], [0], Dta )
        Dta = IIF ( Dta = [.T.], [1], Dta )
        Dlm = IIF ( TYPE ( Fld ) $ [CM],['],;
        IIF ( TYPE ( Fld ) $ [DT],['],;
        IIF ( TYPE ( Fld ) $ [IN],[], []))
        Cmd = Cmd + Dlm + Dta + Dlm + [, ]
    ENDFOR
    Cmd = LEFT ( Cmd, LEN(Cmd) -2) + [ )] && Remove ", " add "       )"
    RETURN Cmd
ENDFUNC
FUNCTION BuildUpdateCommand
    PARAMETERS pTable, pKeyField
    Cmd = [UPDATE ] + pTable + [ SET ]
```

```
    FOR I = 1 TO FCOUNT()
        Fld = UPPER(FIELD(I))
        IF Fld = UPPER(pKeyField)
            LOOP
        ENDIF
        IF TYPE ( Fld ) = [G]
            LOOP
        ENDIF
        Dta = ALLTRIM(TRANSFORM ( &Fld ))
        IF Dta = [.NULL.]
            DO CASE
            CASE TYPE ( Fld ) $ [CMDT]
                Dta = []
            CASE TYPE ( Fld ) $ [INL]
                Dta = [0]
            ENDCASE
        ENDIF
        Dta = CHRTRAN ( Dta, CHR(39), CHR(146) ) && get rid of single quotes       in the data
        Dta = IIF ( Dta = [/ /], [], Dta )
        Dta = IIF ( Dta = [.F.], [0], Dta )
        Dta = IIF ( Dta = [.T.], [1], Dta )
        Dlm = IIF ( TYPE ( Fld ) $ [CM],['],;
        IIF ( TYPE ( Fld ) $ [DT],['],;
        IIF ( TYPE ( Fld ) $ [IN],[], []))
        Cmd = Cmd + Fld + [=] + Dlm + Dta + Dlm + [, ]
    ENDFOR
    Dlm = IIF ( TYPE ( pKeyField ) = [C], ['], [] )
    Cmd = LEFT ( Cmd, LEN(Cmd) -2 ) ;
    + [ WHERE ] + pKeyField + [=] ;
    + + Dlm + TRANSFORM(EVALUATE(pKeyField)) + Dlm
    RETURN Cmd
ENDFUNC
PROCEDURE DeleteRecord
    LPARAMETERS pTable, pKeyField
    KeyValue = EVALUATE ( pTable + [.] + pKeyField )
    Dlm = IIF ( TYPE ( pKeyField ) = [C], ['], [] )
```

```
    cExpr = [DELETE ] + pTable ;
    + [ WHERE ] + pKeyField + [=] ;
    + Dlm + TRANSFORM ( m.KeyValue ) + Dlm
    lr = SQLExec ( THIS.Handle, cExpr )
    IF lr < 0
        Msg = [Unable to delete record] + CHR(13) + cExpr
        MESSAGEBOX( Msg, 16, [SQL error] )
    ENDIF
ENDPROC
```

这些过程与 VB.NET DataAdapter 所做的完全一样。VB 使用 CommandBuilder 来从 SELECT 命令里推断出 Update、Insert 和 Delete 命令和表的主键。这就是那些代码所做的。

通过互联网访问数据，给 FoxPro 开发者引入最头痛的问题。MS 推荐创建 XML WebService DLL。这很容易创建，并且“消费”(使用)代码是由 VFP8 的 Web Service 的 Toolbox 为你编写的。还有更容易的吗？

WebConnection 是很容易的。大约是一个早上的报酬，你就可以获得一个完整、全面的解决方案工具包，它是由 Rick Strahl 编写的，他是一个比我们任何一个都聪明得多的人。在 Www.west-wind.com 上，只是购买是糟透的事。

VB.NET

第一次我见到.NET 处理数据时，我震惊了。我完全懵了，要不是我找到了一个运行的示例，我还无从下手。这个改进超过了 FoxPro？我感到惊讶。除非你是 MS，它提高了他们的收入。

你当在 VB.NET 里使用与 FoxPro 开发者不太一样的方法来处理数据，这实际上是很简单的事情。首先，是基础。在.NET 里的四个数据来源是：Oracle、ODBC、OLE DB 和 SQL。SQL 是 MS 的 SQLServer，或 MSDE，它与单线程的 SQLServer 一样工作。OLEDB 可以用于几乎其他全部。你也可以使用 ODBC 访问（ODBC 和 OLEDB 的连接串是不同的，详见 www.ConnectionStrings.com），虽然 OLEDB 会更好些。Oracle 是有钱人的当然选择。

MS 已经使得访问 SQLServer 和访问本地表一样容易了。程序员没有动机来使用除 SQL 外的任何其他东西。 我们可以在开发环境使用 MSDE，当然，MS 不在乎我们的钱。他们在乎我们客户的钱。

你所决定的可以用来存取数据的驱动程序，性能有好有坏。如果你想找与 FoxPro 一样快的东东，你在.NET 里是找不到的。现在有一个叫做 VistaDB 的数据库产品没落了，它是免费的。这意味着 MS 将会买下它并杀死它，让 Anthony Carrabino（开发者）成为富翁。但是，它也许会在一段时间内用在一两个项目上。参见我的网页上的发展远景。

从 DataAdapter 开始

在.NET 里的访问是从连接开始的。连接串是特定的驱动程序。www.connectionstrings.com 是一个寻找主意的好地方。在下面示例中，我将建立一个连接并返回一个数据集里的部分数据。

```
Imports System.Data.SQLClient
    ...
    Dim ConnStr as string = "Server=localhost;Database=NorthWind;UID=sa;PWD=;"
```

```
Dim da as New SQLDataAdapter("SELECT * FROM CUSTOMERS",ConnStr)
Dim ds as New Dataset
Da.Fill(ds)
DataGrid1.DataSource = ds.Tables(0)
```

这是你可以用来获取数据的几行代码。你所获得是数据集，它是在内存中的包含有一个或多个表的文本字符，它可以用来做为表格的数据源，正如我在上面代码的末行里所做。（基于 VB 的古怪的计数方案，Table(0)-表号 0-在数据集是第一个也只有一个。）在.NET 里创建对象的一个理由是，重载 New 构造函数数方法 （如，Init）将极易导致混乱。DataAdapter 的 New 方法有多个重载。其中之一要求你创建一个 SQLConnection 对象，传递给它 ConnectionString，并且调用它的 Open 方法，然后再把 SQLConnection 作为第二个参数来传递，如下 ：

```
Imports System.Data.SQLClient
...
Dim ConnStr as string = "Server=localhost;Database=NorthWind;UID=sa;PWD=;"
Dim Conn as New SQLConnection ( ConnStr )
Conn.Open()
Dim da as New SQLDataAdapter( "SELECT * FROM CUSTOMERS", Conn )
Dim ds as New Dataset
Da.Fill(ds)
Conn.Close()
DataGrid1.DataSource = ds.Tables(0)
```

因为 SQL DataAdapter 构造函数有多个重载，你也可以使用连接串作为第二个参数并清除三行的代码。

Datasets

DataAdapter 的 Fill 方法从 SQL 里的 Customers 表提取数据，并放入到 XML 数据集。DataSet 实际上保存有多个表；它更象当作 FoxPro 表单的数据环境来考虑。这就是为什么表格的数据源是 ds.Tables(0)。Tables(0)是数据集里的第一个表（在这里也只有一个）。你也可以通过它的名来引用，比如，ds.Tables("Customers")，因为 Tables()是在重载时使用表号或是表名。

我想说 XML 是 MS 用来替代 DBF 的。Dataset 是用来代替多表 DBF 的。你甚至可以编写一个应用程序来从磁盘文件里直接使用 XML。它没有索引，因此你将会在你的表达到相当大小后，不得不要把它升迁到 SQL。

你可能会认为类似于 SKIP 的指针移动命令也适用于数据集，但并不是如此。对于数据集没有“当前行”的概念。表单有一个对象叫作 BindingContext 可以处理这些事。它简单的以数据集表作为参数来重载表单。

在表单的数据集表里的 SKIP 命令如下：

```
BindingContext(ds.Tables(0)).Position += 1
```

SKIP －1 如下：

```
BindingContext(ds.Tables(0)).Position -= 1
```

GOTP 如下：

BindingContext(ds.Tables(0)).Position = 0

GO BOTTOM 如下：

BindingContext(ds.Tables(0)).Position = ds.Tables(0).Rows.count-1

然而，TableUpdate() 和 TableRevert()是数据集的方法，只不过它们叫作 AcceptChanges() 和 RejectChanges()。但是它们只接受或拒绝对数据集的更改。它并没有返回你的数据到它的来源处。

在.NET 里的数据管理是基于非联机的数据集。你可以打开连接，获取数据，然后再关闭连接。数据的存储，就象是你的住在疗养院里的伯祖父 Bob，不记得你是从哪儿来的。为了应用你的更改，你需要构造 INSERT、 UPDATE 和 DELETE 语句，并发送它们到通常使用 DataAdapter 来存储数据的地方。

DataAdapters

你可以使用 Command 对象或 DataAdapter 对象来返回数据集。我现在将略过 OLEDBCommand 和 SQLCommand， OLEDBDataAdapter 和 SQLDataAdapter 的差别；只是知道是两回事。

如果你使用 DataAdapter，并且表有主键，你可以使用 CommandBuilder 和 SELECT 语句来建立 UPDATE、INSERT 和 DELETE 命令。在你的表单里的每个表你都需要一个 DataAdapter。如果你设计一个包含所有记录的列表，或者是符合查询条件的所有记录，而且当前又有一个记录可供查看和编辑，那么为同一个表你就需要两个 DataAdapter。你可以使用向导来创建每一个 DataAdapter。从表单设计器里的工具栏里拖动一个 SQL DataAdapter。一个 SQLConnection 控件和 SQLDataAdapters 控件将会加入到设计器底部的托盘里。连接需要一个 ConnectionString，还有 Open 和 Close 的方法。你需要创建连接然后再打开它。

向导的第一页是询问连接。你可能多半是在项目的开始时就创建数据库和连接的。第二页是询问是否使用 SQL 语句或存储过程。如果你不需要更新表，选择“高级”并取消“生成 Insert， Update 和 Delete 语句”选项。由向导创建的 SQLDataAdapter 有 INSERT， UPDATE 和 DELETE 方法；向导的最后一页是确认并创建。

如果你不使用向导，你仍然可以利用这个性能。仅需要创建一个 SQLCommandBuilder 对象，传递给它你的 SQLDataAdapter 的名。确信在你提供 SELECT 语句给 DataAdapter 后再创建它，要么或者你可以浪费一个小时来试图找出它为什么不能工作，就如我在写这篇时所发生的那样。

下面是我写的最短代码，是关于创建你自己的 DataAdapter，并填充它和准备以 INSERT， UPDATE 和 DELETE 命令来用于以你的用户所做的更改来更新后台数据。（注意，如果连接串，SELECT 命令和表名是属性，你可以把这个代码做得更容易些。去试试吧。）

在表单类的 Declarations(申明)里：

```
Public ConnStr As String = "Server=localhost;Database=Northwind;UID=sa;"
    Public Conn As SqlConnection
    Public da As SqlDataAdapter
    Public cb As SqlCommandBuilder
    Public ds As New DataSet
```

在表单的 Load(装载) 事件里：

```
Conn = New SqlConnection(ConnStr)
    da = New SqlDataAdapter("SELECT * FROM CUSTOMERS", Conn)
    Conn.Open()
    da = New SqlDataAdapter("SELECT * FROM Customers", Conn)
    cb = New SqlCommandBuilder(da) ' creates the Insert, Update and Delete commands
    da.Fill(ds, "Customers")
    Conn.Close()
    DataGrid1.DataSource = ds.Tables(0)
```

在表单的 Update(更新)控钮的 Click(单击)事件里：

```
Conn.Open()
    da.Update(ds, "Customers")
    Conn.Close()
```

你可以使用任何的 SELECT 语句来检索数据。可是，如果你想用向导来自动生成 INSERT， UPDATE 和 DELETE 语句，你所选择的表必须要有主键。

如果你当需要的是从表里读取部分数据，你可以在你的代码里实例化一个 DataAdapter，指定一个 SELECT 语句，并使用它来返回一个数据集，所有的代码如下：

```
Dim da as New SQLDataAdapter ( "SELECT * FROM CUSTOMERS", "Server=localhost;...")
Dim ds as New Dataset Da.Fill(ds)
```

模板数据集

数据集可以作为任何控件的数据源（从技术上说，任何控件都支持 IBindable 接口，关于接口的更多内容后谈）。表格可以显示数据集的内容，仅需要把数据集指定给连接的相应指针。当你打开在属性表单里的+复合属性（DataBuildings），然后在 Text 属性旁边输入表名.列名，你并没有提及数据集。当你从表单项设计器的底部托盘的 DataAdapter 里生成一个数据集时，IDE（集成开发环境）会为你编写一个 VB 程序。

点击 Solution Explorer 里的 “Show All Files（显示所有文件）”图标，你将会见到我所说的。Customers.CompanyName 并不是 表名.列名，在 FoxPro 下它是的；它是 对象名.属性名。这个对象名对象是基于 IDE 为你编写的那个类。属性名是在同一个程序里取得的属性。你可以自行打开它查看。我可以等着。

很吃惊？我是如此的。在 FoxPro 里非常简单明了的事在.NET 却是这么繁琐？相信这是有原因的，但是我并不关心这个。

没有什么是免费的，并且先前的关于免费的告诫还仍然存在。多个公司，包括 RapTier， Visible Developer 和 DeKlarit，销售的创建模板数据集的生成器都有附加功能。这些天已经在互联网上快要免费了的。就如你所知，你自己可以编写一个更好的模板数据集生成器。我真的,真的希望 MS 能够在不远的将来排除这样的需要。我们在 Xbase 世界已经 20 年都没有它，所以我知道它并不是必要的。

使用 DataView 来排序和过滤数据

FoxPro 有两项令人惊奇的性能，SET FILTER 和 SET ORDER，可以让我们限制用户可以见到的记录和他们所见到的记录顺序。数据集也有同样的性能—排序。告诉我是不是它让你觉得有点棘手？

数据集可以包括有多个表。它们类似于表的集合，比如，dsCusts1.Tables(0) 或 dsCusts1.Tables("Customers")

对于表是没有排序和过滤性能。然而，每个表都有一个 DefaultView（默认视图），它却是有的。于是，为了使用这些，你将不得不创建一个本地 DataView（数据视图）对象。

加入两个标题分别为“Filter（过滤）”和“Sort（排序）”的命令按钮到你刚刚创建的表单，并使用下面的代码来作快速演示。

```
Private Sub Form1_Load( _
    ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles MyBase.Load
    Me.OdbcDataAdapter1.Fill(DsCusts1)
    End Sub
Private Sub cmdFilter_Click( _
    ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles cmdfilter.Click
    Dim dv As DataView
    dv = DsCusts1.Tables(0).DefaultView
    dv.RowFilter = "CompanyName Like 'A%'"
    DataGrid1.DataSource = dv
    MessageBox.Show("Only A's")
    dv.RowFilter = ""
    End Sub
Private Sub cmdSort_Click( _
    ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles cmdSort.Click
    Dim dv As DataView
    dv = DsCusts1.Tables(0).DefaultView
    dv.Sort = "CompanyName DESC"
    MessageBox.Show("Descending...")
    dv.Sort = "CompanyName"
    MessageBox.Show("Asscending...")
    End Sub
```

正如你所见，在以.NET 处理数据时是个大大的问题。我有说到它慢吗？然而，现在开始试验.NET 却是个好主意，

如果你还没有开始，因为我非常确信它在不久的将来会成为我们职业的一部分。

再见。

Les Pinter MS VFP MVP

Member INETA Speakers' Bureau

狐眼细看 MySQL（一、简介篇）

原著：Dan Jurden

翻译：Yasur

今天，MySQL 已经引起了极大的关注。那么，究竟有什么值得兴奋的呢？在这个系列的文章中，Dan Jurden 将带着我们探索 MySQL 从基础到高级的内容。他还将指出 MySQL 的一些优点和缺点，并做了大量与 SQL Server 的比较。最后，作为一个程序员，当你开始开发你的下一个客服/服务器应用程序的时候，你将会拥有足够的知识去作出明智的决定：选择哪一个数据库引擎才能最好的为你工作。

要知道 MySQL 数据库服务器是最有名的开放源代码数据库，它有着超过 400 万的用户。它是一个真正多平台的应用程序。它可以运行在 Linux、Microsoft Windows、FreeBSD、Sun Solaris、IBM AIX、Mac OS X、HP-UX、AIX、Novell Netware、SCO OpenUnix、SGI Irix、以及 Dec OSF 等多种操作系统平台上。你可以使用大多数编程语言、从所有的主流操作系统平台上连接到 MySQL 数据库服务器。它飞速运行，并且它是免费的！

一些世界最大的公司正在使用它——德克萨斯仪器、美国人口统计局、Yahoo 法国、美国航空航天局、爱立信电子、硅谷科技有限公司、Slashdot、奥马哈牛排、罗德岛州、美联社、Cox 通讯，以及其它许多企业。

你现在相信了吗？好，我们继续。正因为它深受喜爱、能够运行在多种操作系统平台上、能够通过几乎所有的编程语言连接到它上面、并且被许多大型企业所使用，所以不要错过它了。让我告诉你更多的信息。

MySQL 中有什么东西？

我发现 MySQL 的功能相当的多。这里是一个在 MySQL 最新的版本(4.0.16)中主要功能的列表：

许可授权

MySQL 是双重许可的。用户可以选择作为一个遵守 GNU 协议（参见 www.gnu.org/licenses）的开放源代码/免费软件来使用它，也可以选择从 MySQL AB(www.mysql.com)购买一个商业授权版。

连接到服务器

连接到一个服务器是通过一个 IP 地址来完成的。可以通过局域网或者 Internet 来连接。MySQL 默认使用 306 端

口，但这是可以设置的。此外，还有一个 ODBC 驱动和一个自定义 .NET 数据源适配器(Data Adaptor)可以让你的应用程序连接到 MySQL，都是免费的。

ANSI SQL 语法支持

MySQL 支持 ANSI SQL 99 语法标准的一个子集。当然，它还有一些自己的扩展，例如 REPLACE 语句，以及用于 SELECT 语句和 DELETE 语句的一个 LIMIT 子句。MySQL 还支持来自于其它数据库系统的一些语法，以便于本来使用其它数据库的应用程序能更轻松的与 MySQL 对接。在这个系列文章中的后续部分中将会讲述更多关于扩展的内容。

独立的存储引擎

这是 MySQL 一个独有的功能。它支持多种存储引擎（或者表类型）。用户可以选择最适合某个特定的应用程序的引擎。例如，如果你需要行锁定、外部键约束、和事务支持，你应该选择 InnoDB 存储引擎。当应用程序不需要这些支持，而需要更好的性能时，你可以选择 MyISAM 存储系统。

你甚至可以让一个数据库中的表使用不同的存储引擎。例如，对于那些没有多少数据、用于查询和校验的参照表，用 MyISAM 引擎最恰当。它可以提供非常快速的搜索、并且只需要很少的存储空间。

相反，对于那些拥有大量的数据、并且还有外键约束和参照完整性约束（级连删除）的表来说，InnoDB 引擎会是一个更好的选择。此外，通过 ALTER TABLE 命令还可以将表从一种存储引擎转换到另一种。后续的文章中将会谈到更多关于这些引擎的内容。

事务和外键约束

如前所述，InnoDB 和 Berkeley DB 存储引擎支持事务。这样就让数据改动可以被封装在一个事务中，一起被提交、或者在出错时一起被回滚。在 SQL Server 中，有多种事务隔离层次来控制怎样处理锁定。InnoDB 存储引擎也支持外键约束。

灵活的安全性

MySQL 有一个先进而灵活的安全系统。包括对 SSL 传送级加密的支持。用户名包括两个部分：用户 ID 和一个主机名。这种机制提供了这样的能力：JOE 可以从他的办公室的机器上连接到数据库服务器，而他家里的机器则不行。这种安全机制还允许限制每个用户能够使用的资源。

全文本索引和搜索

全文本索引允许对包含任意文本的文本字段查询特定的关键字和短语。这种全文本搜索能力包括相关性等级（Relevance rankings）、准确短语匹配、和二进制搜索操作。它让你可以对搜索的结果进行很大的控制。

数据库容量

MyISAM 引擎的表的最大限度是 800 万特（TB）。所以，事实上，表文件的大小只受使用的操作系统的限制。曾有报道说，某些用户的数据库中竟然用了超过 60000 个表、而且还有超过 50 亿条记录。

每个表最多能有 32 个索引。每个索引可以由 16 个字段组成。索引的最大长度是 500 字节。

管理

有一个免费的管理工具可供使用。MySQL 控制中心是一个与平台无关的图形用户界面管理客户端。它有一个能高亮显示语法的编辑器以供执行交互式查询，还有数据库和表管理、服务器管理、以及安全设置。

更多内容

其它功能包括查询缓存、复制、一个更宽范围的数据类型（包括日期、时间和二进制大对象字段，等等）。这个系列文章的后续部分将讲述更多这方面的内容。

下一个版本有什么？

还有一些功能的缺乏使得某些开发人员不愿意选择 MySQL 作为他们的数据库引擎。这里是对 MySQL 未来版本的一个 Wish list。注意这里列出的版本只是对何时这些功能被发布的看法：

支持全部 Unicode 的字符集处理

当前的 MySQL 可以被设置为支持某个特定的字符集。这是一个服务器层次的设置。4.1 版将支持在服务器、数据库、表或者字段层次上的多字符集设置。将会增加支持 UCS2 和 UTF8 解码的新字符集。

扩展对子查询的支持

目前，子查询是以某种象 INSERT INTO ...SELECT...这样的形式来支持的。在 4.1 版中将支持 SELECT 查询。

存储过程和触发器

5.0 版计划支持存储过程和触发器。此外，这个版本还将提供对用多种语言实现的存储过程的 hook。

增强的外键支持

5.0 版将为 MyISAM 数据引擎增加外键支持。

视图

视图被计划在 5.1 版中增加到 MySQL 数据库服务器上。

更快的速度

每个版本都保证提供更好的性能！

总结

如你所见，MySQL 有一些上佳的功能，并且将来的版本中还会有更多。这些功能让它成为目前和将来的客户/服务器应用程序开发中的一个实用的选择。

我想要强调的是，我并非想让 SQL Server 的用户们转投到 MySQL 阵营里来。这个系列文章的目标是使开发人员们了解这个工具。随着 Linux 的深入推广、以及 MySQL 的大量应用，程序员们将会有大量的机会被要求使用 MySQL。有足够的知识对客户说“行”、而不是将他们推给其它的程序员肯定是最好不过的了。

很多时间和场合下，使用 SQL Server 是最好的选择。同样也有很多时间和场合下使用 VFP 表是最好的选择。当 MySQL 是最佳选择时，你最好拥有足够的知识和背景，并能提供这样的应用程序。这就是这个系列文章的目的。

把你的爪子从我的资料上拿开

原著: Doug Hennig

翻译: CY

你需要防止别人对你的 FoxPro 数据的未授权访问吗？ Doug Hennig 注意到 Xitech 的 Cryptor，它提供了一个很容易的办法来加密文件，并将它们加密保留在磁盘上，却可以让你在应用程序里以普通表文件一样来访问。

在 VFP 在线论坛里的常见问题是如何对非授权用户保护你的数据。这里有几个选择：

- 升迁为更安全的数据引擎，比如 SQLServer 或 Oracle。这或许是你的长期策略（或许不是，取决于你的观点），这意味着对你的应用程序大部分的重新设计。
- 加密表内数据。虽然，你可以使用你自己的加密和解密过程，更好的选择是利用 Windows 内置的加密函数；虽然它们使用复杂，但是 VFP 在 FoxPro 基础类里带来了一个包装的类（在_Crypt.VCX 里的_CryptAPI），可使得这更容易使用。然而，这个方法却是难于加入到应用里，因为对于访问的表内的所有数据 ，在读取时要解密数据，在写入时要加密数据。
- 加密磁盘上的表，当你的应用程序打开它时解密，并在处理时解密它。类似于前面的选择，这也是复杂的，事实是在访问表内的任何数据都需要进行这样的操作。当你在应用程序时打开它时，它从此时就保抚持为未加密的状态，因而对于非授权访问也是保持为打开状态，加上它在多用户并发下会有问题。

幸运的，这时有更好的解决办法：来自 Xitech 的 Cryptor。这个库提供了加密和解密字符与文件等的函数，甚至，可以保留文件为加密状态在磁盘上，即使你的应用程序在访问它的时候。

安装 Cryptor

你可以从 www.xitech-europe.co.uk 里下载评估版本。其主文件是 XICRCORE.DLL；它提供了 Cryptor 服务的函数。还有几个给 VFP 开发者的 FLL 文件，它提供了一个 XICRCORE 里的包装函数。然而，因为它是针对特定版本的（比如，C40FOX70.FLL 是针对 VFP7），这意味着你至少需要多个文件来随着你的应用程序来发放，并且因为使用 XICRCORE 函数本身并不困难，我的主张是并不需要把 FLL 文件分开。Cryptor 带有一个 Help 帮助文件，并支持使用于 Delphi，C++和 VB。

使用 Cryptor

因为 Cryptor 函数是包含在 DLL 里，你需要在使用前对它们进行申明。这里有些代码（修改自伴随于下载文件的 TESTAPI.PRG），它申明了我将使用的 XICRCORE.DLL 的函数：

```
lcDLL = 'XICrCore.DLL'
declare integer CRYIni_Initialize in XICrCore.DLL ;
integer LoadMode
declare integer CRYIni_UnInitialize in XICrCore.DLL
declare integer CRYUtl_EncodeString in XICrCore.DLL ;
string strSrc, string @ strDest, integer dwLength, ;
string strPassword, integer dwMethod
declare integer CRYUtl_DecodeString in XICrCore.DLL ;
string strSrc, string @ strDest, integer dwLength, ;
string strPassword, integer dwMethod
declare string CRYUtl_GetErrorMessage in XICrCore.DLL ;
integer errorCode
declare integer CRYUtl_Encode in XICrCore.DLL ;
string strFilename, string strPassword, ;
string strBackupExt, integer bKeepBackup, ;
integer dwMethod
declare integer CRYUtl_Decode in XICrCore.DLL ;
string strFilename, string strPassword, ;
string strBackupExt, integer bKeepBackup, ;
integer dwMethod
declare integer CRYMan_Register in XICrCore.DLL ;
string strFilename, string strPassword, ;
integer dwFlags, integer dwMethod
declare integer CRYMan_Unregister in XICrCore.DLL ;
string strFilename
```

在你可以利用 Cryptor 做任何事以前，你还必须利用 CRYIni_Initialize 来初始化它。这个函数必须传递一个申明“装载模式”的参数（Cryptor 帮助文件里解释了可使用的具体值和原由）；在这个文章的随后和其他代码里，CRYPTOR_LOADMODE_NORMAL 和其他大写的内容是定义在 SFCRYPTOR.H 里常数。就象其他 Cryptor 函数，CRYIni_Initialize 返回一个数值，以表明是否成功（0，或者常数 CRYPTOR_ERR_SUCCESS），或者如果不成功，则是错误代码。在这个代码里，我们接受 CRYPTOR_ERR_SUCCESS 或 CRYPTOR_ERR_ALREADY_INITIALIZED （表明 Cryptor 已经被初始化）意味着成功。

```
lnStatus = CRYIni_Initialize(CRYPTOR_LOADMODE_NORMAL)
if not inlist(lnStatus, CRYPTOR_ERR_SUCCESS, ;
CRYPTOR_ERR_ALREADY_INITIALIZED)
```

```
messagebox('Could not initialize Cryptor: status ' + ;
'code ' + transform(lnStatus))
return
endif not inlist(lnStatus ...
```

一旦你结束使用 Cryptor，你还应该复原它：

```
lnStatus = CRYIni_UnInitialize()
```

加密字符是很容易的：只需要调用 CRYUtl_EncodeString 函数，传递给它待加密的字符，一个将存储加密后结果的字符（按参传递），字符的长度，用来加密的密码，和使用的加密方法。Cryptor 提供了几个加密级别；在帮助文件里仅简要的提及这些级别的差异，很可能是因为商业秘密的原因。如果加密处理失败，你可以调用 CRYUtl_GetErrorMessage 来获取错误信息的文本（虽然帮助文件里讲述 CRYUtl_EncodeString 都一直可以返回 CRYPTOR_ERR_SUCCESS）。这里是使用 2 级加密：

```
lcString = 'MyString'
lnLen = len(lcString)
lcEncrypted = space(lnLen)
lcPassword = 'FoxRocks!'
lnMethod = ENCODER_LEVEL2
lnStatus = CRYUtl_EncodeString(lcString, ;
@lcEncrypted, lnLen, lcPassword, lnMethod)
if lnStatus = CRYPTOR_ERR_SUCCESS
messagebox(lcString + ' encrypts using method ' + ;
transform(lnMethod) + ' and password ' + ;
lcPassword + ' to ' + lcEncrypted)
else
messagebox('Could not encrypt string: ' + ;
CRYUtl_GetErrorMessage(lnStatus))
endif lnStatus = CRYPTOR_ERR_SUCCESS
```

解密也是很容易的，并使用同样的代码。这里，我们将简单的解密先前加密过的字符，以确信我们可以把它复原回来。

```
lcString = space(lnLen)
lnStatus = CRYUtl_DecodeString(lcEncrypted, ;
@lcString, lnLen, lcPassword, lnMethod)
if lnStatus = CRYPTOR_ERR_SUCCESS
```

```
messagebox('The encrypted string decrypts back to ' + ;
lcString)
else
messagebox('Could not decrypt string: ' + ;
CRYUtl_GetErrorMessage(lnStatus))
endif lnStatus = CRYPTOR_ERR_SUCCESS
```

除字符之外，你也可以利用 CRYUtl_Encode 和 CRYUtl_Decode 函数来加密和解密整个文件。传递给函数指定文件的全名，使用的密码，Cryptor 在处理中创建的用以备份的扩展名（.NULL.，意味着使用默认的 CBK 扩展名），0 表示完成后删除备份文件，1 表示保留备份文件，还有使用的加密方法。这里的示例是加密一个表，并试图打开它（这肯定会失败，因为它不再是可识别的 VFP 表），然后解密并试图再次打开它（这肯定会成功）。

```
create table TEST (FIELD1 C(10), MEMO1 M)
index on FIELD1 tag FIELD1
insert into TEST values ('A', 'My memo')
use
lcFile = fullpath('TEST.DBF')
lnStatus = CRYUtl_Encode(lcFile, lcPassword, .NULL., 0, ;
lnMethod)
if lnStatus = CRYPTOR_ERR_SUCCESS
messagebox('We should get an error when trying to ' + ;
"USE the table because it's encrypted. Choose " + ;
'Ignore so we can continue.')
use TEST
lnStatus = CRYUtl_Decode(fullpath('TEST.DBF'), ;
lcPassword, .NULL., 0, lnMethod)
if lnStatus = CRYPTOR_ERR_SUCCESS
use TEST
if used('TEST')
messagebox('The table was successfully decrypted.')
browse
use
endif used('TEST')
else
messagebox('Could not decrypt table: ' + ;
CRYUtl_GetErrorMessage(lnStatus))
```

```
endif lnStatus = CRYPTOR_ERR_SUCCESS
else
messagebox('Could not encrypt table: ' + ;
CRYUtl_GetErrorMessage(lnStatus))
endif lnStatus = CRYPTOR_ERR_SUCCESS
```

所有这些都是优秀的，但是 Cryptor 已经显示出可在内存中解密文件的能力。以其他机制加密文件，你将不得不在使用前把它解密。那意味着它在使用时，它是对窥探者是敏感的。利用 Cryptor 的 CRYMan_Register 函数，你可以指定某个特定文件以加密方式保留在磁盘上，但在应用程序使用时可以自动的在内存中解密。于是，你可以得到保护防止非授权讯问文件，因为它们在磁盘上永远不会被解密。

为利用 Cryptor 注册一个或多个文件，传递给 CRYMan_Register 函数文件名（你可以使用通配符，比如 MYTABLE.*，以在一次调用时注册多个文件），密码，数值 0（这个参数没有用；它只是为了与先前版本的 Cryptor 保持兼容），以及加密方法。这个示例假定 TEST.DBF，CDX 和 FPT 是加密的，于是以 Cryptor 来注册它们以使得 VFP 可以打开表。然后使用 CRYMan_Unregister 函数来得取消注册它们；VFP 将不再能打开表。

```
lcFile = fullpath('TEST.*')
lnStatus = CRYMan_Register(lcFile, lcPassword, 0, ;
lnMethod)
if inlist(lnStatus, CRYPTOR_ERR_SUCCESS, ;
CRYPTOR_ERR_ALREADY_REGISTERED)
use TEST
if used('TEST')
messagebox('The table was decrypted in memory ' + ;
'so we can use it.')
browse
use
endif used('TEST')
lnStatus = CRYMan_Unregister(lcFile)
if lnStatus = CRYPTOR_ERR_SUCCESS
messagebox('We should get an error when trying ' + ;
"to USE the table because it's still " + ;
'encrypted. Choose Ignore so we can continue.')
use TEST
else
messagebox('Could not unregister table: ' + ;
CRYUtl_GetErrorMessage(lnStatus))
```

```
endif lnStatus = CRYPTOR_ERR_SUCCESS
else
messagebox('Could not register table: ' + ;
CRYUtl_GetErrorMessage(lnStatus))
endif inlist(lnStatus ...
```

SFCryptor

虽然使用 Cryptor 是很容易的，这里还是需要申明函数，初始化和复原，同样的参数传递给同样的方法，等等。我创建了一个名为 SFCryptor 的包装类，是基于 SFCustom（在 SFCTRLS.VCX 的自定义基类），用来处理所有杂乱的细节。比如，这里有些代码（修改自 TESTSFCRYPTOR.PRG），可以加密和解密字符。注意，你不需要在每次调用时传递密码和加密方法，也不需要做任何初始化或其他工作。

```
loCryptor = newobject('SFCryptor', 'SFCryptor.vcx')
loCryptor.nEncryptionMethod = ENCODER_LEVEL2
loCryptor.cPassword = 'FoxRocks!'

* Encrypt a string.
*  加密字符

lcString = 'MyString'
lcEncrypted = loCryptor.EncryptString(lcString)
if not isnull(lcEncrypted)
messagebox(lcString + ' encrypts using method ' + ;
transform(loCryptor.nEncryptionMethod) + ;
' and password ' + loCryptor.cPassword + ' to ' + ;
lcEncrypted)

* Decrypt the same string.
*  解密字符

lcString = loCryptor.DecryptString(lcEncrypted)
if not isnull(lcString)
messagebox('The encrypted string decrypts back ' + ;
'to ' + lcString)
else
messagebox(loCryptor.cErrorMessage)
```

```
endif lnStatus = CRYPTOR_ERR_SUCCESS
else
messagebox(loCryptor.cErrorMessage)
endif not isnull(lcEncryptedString)
```

EncryptString 和 DecryptString 方法是非常简单的：它们只调用保护的 ProcessString 方法。这里给出 DecryptString 方法（EncryptString 除了是传递给 ProcessString “加密”以外都是一样的）。注意，它可以接受密码和加密方法参数的；如果你没有传递，cPassword 和 nEncryptionMethod 属性的值将会被使用。

```
lparameters tcString, ;
tcPassword, ;
tnMethod
return This.ProcessString(tcString, tcPassword, ;
tnMethod, 'decrypt')
```

ProcessString 在开始时会确认是否传递了字符；它使用了在 VFP8 里新增的 EVL()函数来检查 tcString 参数的数据类型（如果你想使更早版本的 VFP 里使用这个类，使用 VARTYPE()来代替这个语句，以确保传递的是字符）。然后它将设置必要的变量，检查以确认 Cryptor 已经初始化，并根据它所假定要做的来调用 CRYUtl_DecodeString 或 CRYUtl_EncodeString 函数。如果成功，它将返回加密后或解密后的字符；否则，它将设置 cErrorMessage 属性为相应的值并返回.NULL.。

```
lparameters tcString, ;
tcPassword, ;
tnMethod, ;
tcProcess
local lcPassword, ;
lnMethod, ;
lnLen, ;
lcString, ;
lnStatus
with This

* Ensure a string was passed.
*  确认传递的字符

assert not empty(evl(tcString, '')) ;
message 'Must pass string'
```

```
* Use the cPassword and nEncryptionMethod properties if
* the parameters weren't passed. Create other variables
* we need.
*  如果没有参数未传递，就利用 cPassword  和  nEncryptionMethod 属性，并创建其他必要的变量。

lcPassword = iif(empty(evl(tcPassword, '')), ;
.cPassword, tcPassword)
lnMethod = iif(vartype(tnMethod) <> 'N', ;
.nEncryptionMethod, tnMethod)
lnLen = len(tcString)
lcString = space(lnLen)

* If Cryptor is initialized, try to process the string.
*  如果 Cryptor 已经初始化，尝试开始处理字符。

if .InitializeCryptor()
if tcProcess = 'decrypt'
lnStatus = CRYUtl_DecodeString(tcString, ;
@lcString, lnLen, lcPassword, lnMethod)
else
lnStatus = CRYUtl_EncodeString(tcString, ;
@lcString, lnLen, lcPassword, lnMethod)
endif tcProcess = 'decrypt'
if lnStatus <> CRYPTOR_ERR_SUCCESS
.cErrorMessage = 'Could not ' + tcProcess + ;
' string: ' + CRYUtl_GetErrorMessage(lnStatus)
lcString = .NULL.
endif lnStatus = CRYPTOR_ERR_SUCCESS
else
lcString = .NULL.
endif .InitializeCryptor()
endwith
return lcString
```

在使用 Cryptor 函数的 SFCryptor 里的所有方法都调用 InitializeCryptor 方法。这个方法提供了函数的申明和初始化。这样比在类的 Init 方法里做有两个好处：这个任务仅在实际需要时才调用，并且某些属性可以告诉 SFCryptor 如何以 Cryptor 工作（cCryptorPath，它指出了 XICRCORE.DLL 的位置，nLoadMode，它指出了使用的装载模式），它可以在类具体实例化后自动被设置。

```
local lcDLL, ;
llReturn
with This

* If we're not already initialized, see if we can find
* XICrCore.dll.
*  如果我们没有初始化，看看是否可以找到 XICrCore.dll。

if not .lInitialized
lcDLL = fullpath('XICrCore.dll', ;
addbs(.cCryptorPath))
llReturn = file(lcDLL)
if llReturn

* Declare the functions that we'll use in XICrCore.
*  申明我们将要使用的 XICrCore 里的函数。

declare integer CRYIni_Initialize in (lcDLL) ;
integer LoadMode
*** other function declarations omitted for brevity
***  其他函数申明省略

* Initialize Cryptor.
*  初始化 Cryptor

lnStatus = CRYIni_Initialize(.nLoadMode)
if inlist(lnStatus, CRYPTOR_ERR_SUCCESS, ;
CRYPTOR_ERR_ALREADY_INITIALIZED)
.lInitialized = .T.
else
.cErrorMessage = 'Could not initialize ' + ;
```

```
'Cryptor: status code ' + transform(lnStatus)
llReturn = .F.
endif inlist(lnStatus ...
else
.cErrorMessage = 'XICrCore.dll was not found.'
endif llReturn
else
llReturn = .T.
endif not .lInitialized
endwith
return llReturn
```

这里也有加密和解密文件的方法（Encrypt 和 Decrypt），以及注册和取消注册文件的方法（Register t 和 Unregister）；所有这些方法只是调用保护的 ProcessFile 方法来进行实际的操作。你可以自行查看这些方法的代码。

ReleaseMembers 方法是在 SFCryptor 对象释放时来调用的，通过取消注册所有已注册的文件来清除，复原 Cryptor，并清除 Cryptor 函数。

```
local lcFile, ;
lnFlags, ;
lnMethod, ;
lnCount, ;
lnFiles, ;
lnStatus, ;
laFiles[1], ;
lnI
if This.lInitialized

* Unregister all files. We have to gather a list of
* files first, then unregister them, because you can't
* unregister a file while a list operation is in
* progress.
*  对所有文件取消注册，首先我们将要收集文件列表，然后取消注册，因为你无法在列表操作正在进行时来取消注册文件。

lcFile = space(260)
lnFlags = 0
```

```
lnMethod = 0
lnCount = 0
lnFiles = 0
lnStatus = CRYMan_List(CRYPTOR_REGLIST_FIRST, ;
@lcFile, @lnFlags, @lnMethod, @lnCount)
do while inlist(lnStatus, CRYPTOR_ERR_SUCCESS, ;
CRYPTOR_ERR_END_OF_LIST)
lnFiles = lnFiles + 1
dimension laFiles[lnFiles]
laFiles[lnFiles] = lcFile
lnStatus = CRYMan_List(CRYPTOR_REGLIST_NEXT, ;
@lcFile, @lnFlags, @lnMethod, @lnCount)
enddo while inlist(lnStatus ...
for lnI = 1 to lnFiles
lnStatus = CRYMan_Unregister(laFiles[lnI])
next lnI

* Uninitialize Cryptor.
*  复原 Cryptor

CRYIni_UnInitialize()

* Clear all functions.
*  清除所有函数

clear dlls CRYIni_Initialize, CRYIni_UnInitialize, ;
CRYUtl_EncodeString, CRYUtl_DecodeString, ;
CRYUtl_GetErrorMessage, CRYUtl_Encode, ;
CRYUtl_Decode, CRYMan_Register, CRYMan_Unregister
endif This.lInitialized

* Carry on with the usual behavior.
*  继续回到可视界面

dodefault()
```

价格与授权

Cryptor 4.0 是$299，提供 50 个运行时授权。另外的 500 个运行时授权将会是$199。无限制版本是$499。正如这里所写的，你无法从 Xitech 网站下单订购 Cryptor。你可以发邮件要求订购 Cryptor，或者是从他们的经销商订购，比如 Hallogram (www.hallogram.com)。

从我开始使用 Cryptor，5.0 版本已经发布。我还没有见到，但是 Xitech 网站上有新特性的细节。最大吸引力的就是 COM 接口，消除了对 DLL 内函数申明的需要。在 Xitech 同站上找不到价格和授权信息；对此他们需要做得更好些。Cryptor 4.0 仍然可用，我想 5.0 版本可能会更贵。

总结

Cryptor 是一个对于防止非授权访问 VFP 数据问题的良好解决方案，并且也可以加密字符（比如存储在 INI 文件或是 Wndows 注册表里的密码）。我使用它将近一年了都很成功。

来自 VFP 开发团队的 Tips

原著:微软开发团队成员

翻译: LQL.NET

雷德蒙得微软开发园区的 41 号楼这些日子里非常的繁忙，Visual FoxPro 开发组正在日以继夜的忙于 VFP 9 的新功能、修正 Bug、写文档和示例代码、以及穷举测试。百忙之中，他们也偷空在 Foxtalk 2.0 上开了一个专栏，每期介绍一些 VFP 新功能的使用方法，这一期讲的就是关于 VFP 9 中 SQL Select 语句的一些新功能...

以下是来自 Microsoft Visual FoxPro 开发组月刊的第一部分。如需 Visual FoxPro 更多信息和资料可以访问 http://msdn.com/vfoxpro 。如要有关 Visual FoxPro 9.0 的更多信息，可以参考各种月刊在 http://msdn.com/vfoxpro/letters.

在 Visual FoxPro 9.0 里有许多新增的内容，包括 SQL 语言的重大改变。不久，你就可以下载 Visual FoxPro 9.0 公开测试版去体验下新的功能。如果要更详细地了解 VFP9 Beta，就请关注 Foxtalk 6 月刊吧。

看，在 6 月到来之前，为吊起你们的胃口，我们提供了一些 VFP9 SQL 语言方面样例，以及 VFP9 对前期版本的一些改动。

Joins

VFP9：无限制

VFP8：限 9 个

子查询

VFP9：无限制

VFP8：限 9 个

UNIONs

VFP9：无限制

VFP8：限 9 个

连接表数量

VFP9：无限制

VFP8：限 30 个

IN()

VFP9：无限制

VFP8：有 24 位限制。实际上仍然受限于 SYS(3055)的设置，SYS(3055)设多少就限多少。

VFP9：允许 UNION 子句中包含 OrderBy 使用的字段

在 VFP8 中这是不行的，你必须用列号代替。

优化 TOP N 的性能

在 VFP8 或更早的版本中，TOP N [PERCENT]是这样运行的：所有记录先排序，然后由 TOP N [PERCENT]萃取。 在 VFP9 中，我们从排序进程中尽早地除掉记录，而不让她进入 TOP N [PERCENT]的萃取进程。

优化 LIKE “sometext%” 的性能

要查找 VFP 本地数据某个字段由”sometext”开头，我们可以用 Rushmore 优化的查询：

SELET * FROM tablel WHERE somefield = “sometext”

然而，许多 SQL 后端并不支持这样的句子。 VFP9 现在使用更通用的表达式：

SELECT * FROM tablel WHERE somefield LIKE “sometext%”

FROM 子句支持 子 SELECT

我们经常提到衍生表。衍生表就是 FROM 子句中的 SELECT 段生成的别名。

SELECT ... FROM (SELECT ...) ...

允许 无关联子查询 使用 TOP N/ORDER BY

VFP9 现将允许 无关联的子查询 使用 TOP N/ORDER BY。如果使用 TOP N，同时也该使用 ORDER BY。这是这一许可的唯一条件。

SELECT ... WHERE ... ;

(SELECT ... TOP nExpr ... FROM ... ORDER BY ...) ...

允许 无关联子查询 使用 GROUP BY

许多查询可以通过“执行一次子查询、将子查询的一个或多个结果替换入外部查询的 Where 子句中”来运算出结果。在包含一个相关的子查询的查询（以“重复查询”而闻名）中，子查询的值依靠外部查询提供。这就意味着子查询是被重复执行的，对外部查询的每一行都要执行一次子查询。

SELECT ... WHERE ... ;

(SELECT ... WHERE ... GROUP BY ...) ...

允许多个子查询嵌套

语法：

SELECT ... WHERE ... ;

(SELECT ... WHERE ... (SELECT) ...) ...

允许在 SELECT 字段列表中使用子查询

VFP9 中 SELECT 字段列表可以包含子查询产生的结果，比如：

SELECT T1.f1, ;

(SELECT f2 FROM foo2 T2 WHERE T2.f1=T1.f1) AS foo2 ;

FROM foo1 T1

允许在 INSERT INTO …SELECT 中使用 UNION

我们将允许在 INSERT INTO 的 FROM 段中使用 UNION ：

INSERT INTO ... ;

(SELECT ... FROM ... UNION SELECT...) ...

允许在 UPDATE SET 列表中使用 子查询

UPDATE foo1 SET f2=100+(SELECT f2 FROM foo2 ;

WHERE foo2.f1=foo1.f1) WHERE f1>5

对 DELETED()标记 进行 Rushmore 优化

我们将优化含有 DELETED()/NOT DELETED()/FOR DELETED()表达式的索引，还有，DELETED 标记中含有 MAX()/MIN() 时，我们也将优化她。

不久我们就会提供更多的源码例程

在下一期 Foxtalk，我们将提供许多 提示/技巧，包括源码，让你马上可以在 VFP9 Beta 中使用。