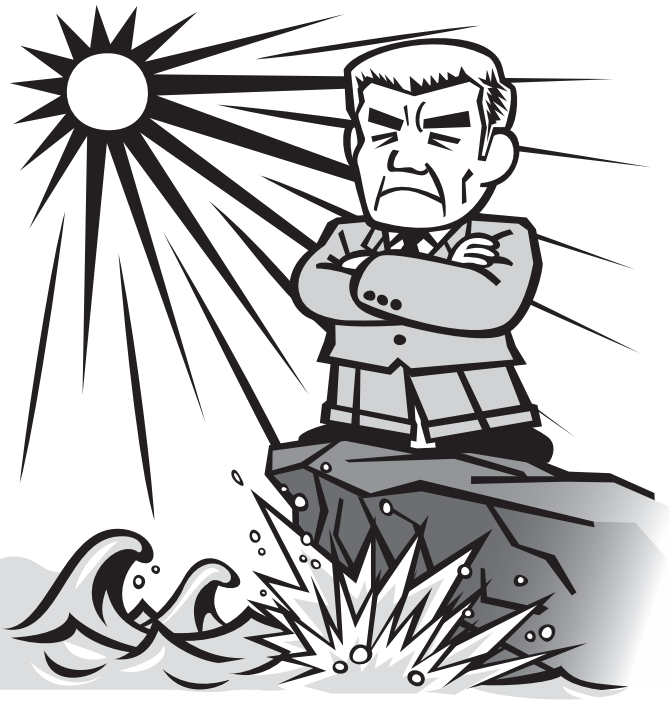
頑固一徹の 天気晴朗なれど波高し

西田 雅昭
NISHIDA, Masaaki



Visual Basic .NET奮戦記

第9回 データベース処理—その3—

Illustration : Toshiyuki Ido

Level

1 2 3 4 5

Technology Tools

- [x] Visual Basic
- [ ] Visual C#
- [ ] Visual C++
- [x] SQL Server
- [ ] Oracle
- [x] Access
- [ ] ASP.NET
- [ ] Other:

Samples

この記事で取り上げたソースコードおよびサンプルプログラムは、http://www.shoeisha.com/mag/windev/からダウンロード可能です。

＊）サンプルを動かすには、SqlConnection コントロール（scnNWind または ocnNWind）の ConnectionString プロパティをご自分の環境に合わせて変更してください。

我輩の名前は「頑固一徹」。Visual Basic一筋のベテランプログラマである。前回は、VB.NETのデータベース処理を理解するために、「データアダプタ」を使わずにデータベースを直接制御する方法について説明した。長年VB6に親しんできた我輩としては、非接続型のデータベース処理（DataAdapterとDataSet）にはどうしてもなじめない。そこで、頑固一徹、接続型の処理（直接処理）を勉強することで、ADO.NETのデータベース処理の実態を理解しようとしたのである。そして我輩は、この直接処理が有利な場面があることに気がついた。

「ExecuteReader」メソッドをもっと詳しく

前回は、直接制御が有利な例として、リストボックスにデータを読み込み、名前を選択するとコードを取得するサンプルプログラムを作成した（図1）。これは、「Command」クラスの「ExecuteReader」メソッドを利用している。

今回は、この「ExecuteReader」メソッドの働きを、もう少し詳しく説明することからはじめよう。

「ExecuteReader」メソッドによるデータの取得は、おそらく最もパフォーマンスの高い方法であり、我輩はもっと頻繁に使われるべきだと思っている。

ただ、注意しなければならないことがある。一度「ExecuteReader」メソッドを実行すると、「xxxDataReader.Close」メソッドを呼び出すまでは、関連付けられた「xxxConnection」オブジェクトは、ビジー状態になる。

この状態では、「xxxConnection」オブジェクトに対して、閉じる以外の操作を実行できない。したがって、「xxxDataReader」の処理が終了したら、ただちに「xxxDataReader.Close」メソッドを実行することを忘れてはならない。

しかし実は、「ExecuteReader」メソッドには、引数を指定できるオーバーロード（表1）があり、これをうまく使えば「xxxDataReader.Close」メソッドを実行する手間を省くことができる。

図1：前回作成したサンプル

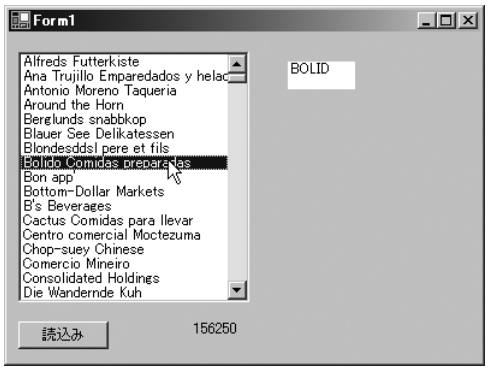


表1：「ExecuteReader」メソッドのオーバーロード

Overloads Public Function ExecuteReader(_ ByVal behavior As CommandBehavior) As SqlDataReader		
パラメータ	behavior	「CommandBehavior」列挙体

リスト1：前回サンプルのデータ読み込み処理部分

```
Private Sub btnRead_Click(ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnRead.Click
  Dim dtrCompanyName As SqlClient.SqlDataReader
  Dim lngStart, lngEnd, lngTime As Long

  scnNWind.Open()

  dtrCompanyName = scmCompanyName.ExecuteReader

  With dtrCompanyName
    While .Read
      lstCompanyName.Items.Add(.Item("CompanyName"))
    End While
    .Close()
  End With

  scnNWind.Close()
End Sub
```

＊）前回追加した読み込み時間の測定コードは割愛。

引数となる「CommandBehavior」列挙体は、クエリの結果と、それがデータソースに与える影響を指定するもので、なかなか便利だ。この列挙体には7つのメンバがあるが、ここではそのうちのひとつ「CloseConnection」について紹介しよう。

引数「CloseConnection」の効果

「ExecuteReader」メソッドに「CommandBehavior」列挙体のメンバ「CloseConnection」を指定して実行した場合は、関連付けられているDataReaderオブジェクトを終了すると、関連付けられているConnectionオブジェクトが終了する。

ここで前回のサンプルプログラムを振り返ってみよう。[読込み]ボタンのクリックイベントプロシージャでは、リスト1のように記述した。

しかし、「ExecuteReader」メソッドに引数「CloseConnection」を加えて、

```
dtrCompanyName = _
 scmCompanyName.ExecuteReader( _
 CommandBehavior.CloseConnection)
```

と記述すれば、「scnNWind.Close()」を記述する必要がない。

うっかり「Close」の書き忘れをしないために、我輩はいつも引数「CloseConnection」を使うことにした。

Memo

http://www.shoeisha.com/mag/windev/からダウンロードできる、本稿のサンプルの中に、前回のサンプルプログラムもあわせて収録している。我輩とともに、前回のサンプルプログラムに修正を加えながら本稿を読み進めてもらいたい。

データの取得を最適化するには？

序数の利用

前回のサンプルプログラムでは、「DataReader」データの取得は、

```
lstCompanyName.Items.Add( _
 .Item("CompanyName"))
```

のように、「DataReader」の「Item」プロパティを使っている。この際、列を指定するのに列名を使っていることに注意してもらいたい。

コレクションの中からひとつのオブジェクトを取得する場合、名前で呼び出すと、その名前を文字列で検索することになる。これでは、オブジェクトの数が多い場合には、多くの時間を要してしまう。

実は、コレクション中のインデックスとして序数が存在しているので、これを使ってオブジェクトを取得すれば、パフォーマンスが向上するのである。

この場合には、

```
lstCompanyName.Items.Add(.Item(1))
```

と記述するほうがスピードが速いはずである。

しかし、求める列（オブジェクト）の序数がわからない場合はどうすればよいであろうか。幸い、「xxxDataReader」クラスの「GetOrdinal」メソッドで、列の名前から列の序数を取得することができる。これを利用すると、リスト2のように記述することができる。

なお、「GetOrdinal」メソッドとは逆に、列の序数から列の名前を取得する「GetName」メソッドも用意されている。