薬師 尚之

Delphi/400 初級

株式会社ミガロ.

システム事業部 システム2課

[Delphi/400] OLEを利用したExcel出力のパフォーマンス向上手法

略歴
1985年11月22日生まれ
2008年3月 阪南大学 流通学部卒業
2008年4月 株式会社ミガロ. 入社
2008年4月 システム事業部配属

現在の仕事内容：
Delphi/400を利用したシステムの受託開発を担当し、基本設計から納品・フォロー、保守作業に至るまで、システム開発全般に携わっている。

1.はじめに

Delphi/400にバンドルされる帳票ツールを利用して帳票機能を開発することが多いが、Delphi/400のバージョンアップによって帳票ツールが変更された場合、帳票機能を移行または作り直しの検討が必要となる（バンドルされる帳票ツールは、Delphi/400 Version 5～7が「QuickReport」、7～XEが「Rave Reports」、XE 3以降は「FastReport」である）。

このような帳票ツールに依存した変更を解決する方法の1つとして、OLE（＊）を利用しExcelをベースにした帳票機能を実装することもできる。

OLEでのExcel利用は、Delphi/400から比較的簡単に実現できるが、大量データの処理には向いておらず、パフォーマンスが落ちることもある。この課題についてはプログラムロジックを工夫することで解決が可能である。本稿では、OLEを利用した基本的なExcel出力方法から、大量データを出力する場合のパフォーマンス向上手法を解説する。

2.OLEを利用した基本的なExcel出力

本章では、Excel出力プログラムの作成例を題材に、OLEの基本的な操作方法について解説する。

Excel出力プログラムの概要

・IBM iより売上情報を取得し、明細をExcelに出力する。
・明細出力時、各営業所単位で合計金額を出力する。
・Excel出力後に保存ダイアログを表示し保存する。

開発環境：Delphi/400 10.2 Tokyo および dbExpress

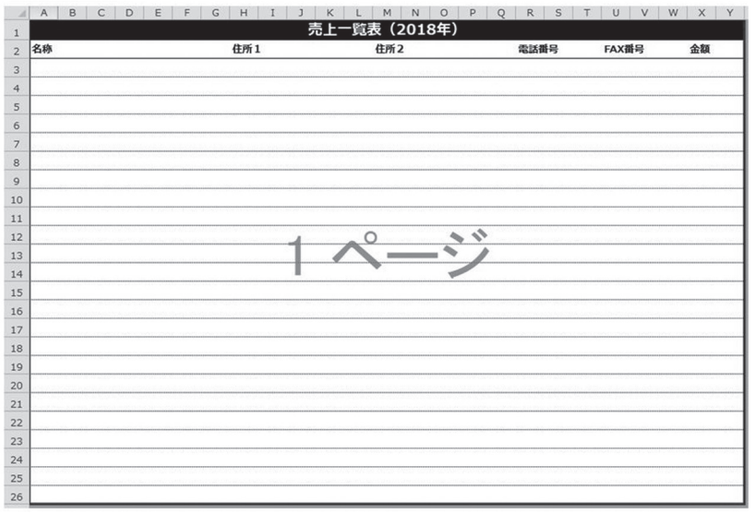
なお本稿では、帳票テンプレートは事前にExcelで作成したものを利用する。Delphi/400から罫線や書式設定の操作を行うことは可能だが、その回数が多くなるほどパフォーマンスの低下に繋がる。そのため、本稿ではあらかじめExcelテンプレート【図1】を作成し、Delphi/400からの操作回数を極力減らしている。またExcelでテンプレートを作成することで、帳票項目の書式設定もExcel側に持たせることができるため、そうした書式変換のプログラミングも不要となる。

【図1】のExcelテンプレートを利用したExcel出力のロジックが【ソース1】、【ソース2】となる。

＊OLE：Object Linking and Embeddingの略称

マイクロソフトが提供する機能の1つで、複数のアプリケーション間でデータの転送や共有を行うための仕組みを指す。Excelは OLE サーバーとなり、他のアプリケーションから操作可能（操作のためのメソッドが用意されている）。実行のためには、Excelが導入されている環境が前提となる。

図1　Excel テンプレート(例)



ソース1

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ovExcel: OleVariant;
  ovWorkBook: OleVariant;
  ovWorkSheet: OleVariant;
  sFileName: String;
  iRow: Integer;
  sFL1: String;
  cShokei: Currency;
begin
  // 出力データ取得 …①
  SQLQuery1.Close;
  SQLQuery1.SQL.Text := 'SELECT * FROM TRURI ORDER BY URIFL1';
  SQLQuery1.Open;
  try
    // Excelオブジェクト生成 …②
    ovExcel := CreateOleObject('Excel.Application');
    try
      // Excel非表示 …③
      ovExcel.Visible := False;

      // フォーマット読み込み …④
      sFileName := IncludeTrailingPathDelimiter(ExtractFileDir(Application.ExeName)) + 'Format.xlsx';
      ovWorkBook := ovExcel.Workbooks.Open(sFileName);
      ovWorkSheet := ovWorkBook.WorkSheets[1];

      // 出力準備
      iRow := 3;
      cShokei := 0;
      sFL1 := SQLQuery1.FieldByName('URIFL1').AsString;  // 小計出力用営業所

      // データ出力
      while not SQLQuery1.Eof do
      begin
        // セルへ転送 …⑤
        ovWorkSheet.Cells[iRow,  1].Value := SQLQuery1.FieldByName('URIFL2').AsString;   // 名称
        ovWorkSheet.Cells[iRow,  8].Value := SQLQuery1.FieldByName('URIFL3').AsString;   // 住所１
        ovWorkSheet.Cells[iRow, 13].Value := SQLQuery1.FieldByName('URIFL4').AsString;   // 住所２
        ovWorkSheet.Cells[iRow, 18].Value := SQLQuery1.FieldByName('URIFL5').AsString;   // 電話番号
        ovWorkSheet.Cells[iRow, 21].Value := SQLQuery1.FieldByName('URIFL6').AsString;   // FAX番号
        ovWorkSheet.Cells[iRow, 24].Value := SQLQuery1.FieldByName('URIFL7').AsCurrency; // 金額

        // 小計を計算
        cShokei := cShokei + SQLQuery1.FieldByName('URIFL7').AsCurrency;
```

①出力データ取得

TSQLQueryを利用してIBM iのファイルよりExcel出力用のデータを取得する。

② Excel オブジェクト生成

CreateOleObjectの引数に「Excel.Application」を指定し、Excelのオブジェクトを生成する。また、変数ovExcelに代入することで生成したオブジェクトをOleVariant型で操作可能にする。前提として、Delphi/400でOLEの各メソッドを利用するにはuses節に「ComObj」を追加する必要がある。

③ Excel 非表示

Excelを表示したままにすると、プログラムのExcel操作が画面ですべて表示されてしまいパフォーマンスも低下するため、ovExcel.VisibleをFalseにして非表示にする。

④フォーマット読込

Excelテンプレートのパスとファイル名を取得（変数sFileName）し、Workbooks.Openで開いたブックを変数ovWorkBookに代入する。ovWorkBookのWorkSheetsプロパティでブックのシート番号を指定し、シートオブジェクトを取得する。これで、①と同様に、ブックとそのシートをOleVariant型で操作可能にする。

⑤セルへ転送

ここでExcelのセルへ値を出力する。

ovWorkSheetのCellsプロパティでセル位置（行および列のインデックス）を指定し、IBM iのデータベース・ファイルより取得した値を代入する。

代入した値はセル側に設定されている書式が適用される。

⑥ Excel 保存

TSaveDialogを配置して、Excel出力後に保存ダイアログを開く。そのダイアログで指定したパスをSaveAsメソッドの引数に渡すことで任意の場所にファイルを保存できる。またFilterプロパティの設定により、保存時に選択できる拡張子の制御が可能である。本稿では「*.xlsx,*.xls」の2種類を指定可能にしている。【図2】

ファイル保存時の注意点を補足しておく。

ダイアログ上で「.xls」を指定して保存した場合、保存したファイルを開く際に拡張子が正しくない旨の警告が表示される。【図3】

これは、Officeのバージョン2007以降、規定のデータ保存フォーマットが変わり、過去のOfficeとは互換性のない形式で保存されることになったためである。対応方法としては、拡張子に「.xls」が指定された場合、SaveAsメソッドの引数にファイル形式を表す定数「56」（Excel 97-2003 ブック）を指定することで後方互換に対応可能となる。これにより保存したファイルは、開いた際に警告が表示されなくなる。この定数はほかにもPDFやCSV形式で出力可能であり、その一覧は以下のWebサイトにて紹介されているので、参考にしてほしい。

●参考URL：

https://msdn.microsoft.com/ja-jp/vba/excel-vba/articles/xlfileformat-enumeration-excel

（Googleで”XlFileFormat列挙”を検索すると上位に表示される。）

⑦ Excel 表示

処理終了後にovExcel.VisibleをTrueにしてExcelを表示する。データ出力が完了した状態でExcelが表示される。

⑧オブジェクト解放

生成したOleVariant型の変数をUnassignedで解放する。

これでExcel出力処理は完成である。このプログラムで実際にExcelでの帳票出力を行った結果が【図4】である。

ソースを見ればわかるとおり、OLEを利用したExcelの出力自体は比較的簡単に実装できる。しかし、この方法では、大量のデータを出力する際にパフォーマンスがかなり悪くなってしまう。その原因としては、Delphi/400からセルに値をセットする際にアプリケーション間で通信が発生する（以下、通信と表記）からだが、実はこれが処理時間が長くなる大きな要因となっている。

本稿のテンプレートを例にすれば、1明細あたり6項目存在するため、1行出力する度に6回の通信が発生する。これが数百、数千件と、扱うデータ件数や項目数が増えると、その分通信が繰り返されるため、パフォーマンスに影響するのは明白である。パフォーマンスを向上させるには、いかにDelphi/400とExcelとの通信回数を減らすかが重要である。その手法については次章で触れる。

3.Excelの出力パフォーマンス

2章では基本的なExcel出力の手法を説明した。本章ではパフォーマンスを向上させる手法、つまりExcelとの通信回数を低減する方法を紹介する。

Excelに値を出力する際、各項目を1セルずつ出力するのではなく、出力する値を2次元配列などに記憶させ、特定のタイミング（改ページ時など）で一括出力することにより、通信回数を格段に低減させることができる。

その方法は2種類あり、クリップボード、もしくはOLEバリアント配列を利用することで実現できる。

しかし、前者のクリップボードについては以下の課題点がある。

●クリップボードを利用する上での課題点

①クリップボードの内容がプログラムで書き換えられてしまう

② WindowsVista以降、クリップボードの動作が不安定

①の課題点

クリップボードは、列ごとに「#9」（タブコード）、改行ごとに「#13#10」（改行コード）のリテラルを挿入することで、複数行・列の内容を格納でき、またその内容をExcelに一括で出力できる。しかし、この手法はユーザーの意図しないところでアプリケーション側からクリップボードの内容が書き換えられてしまい、ユーザーのコピー&ペースト操作などに影響を与えてしまうことがある。

②の課題点

Windowsでは、Excel上でコピー&ペーストを繰り返し実施しているとクリップボードのエラーが発生するという事象がある。

発生する条件は不定だが、特に

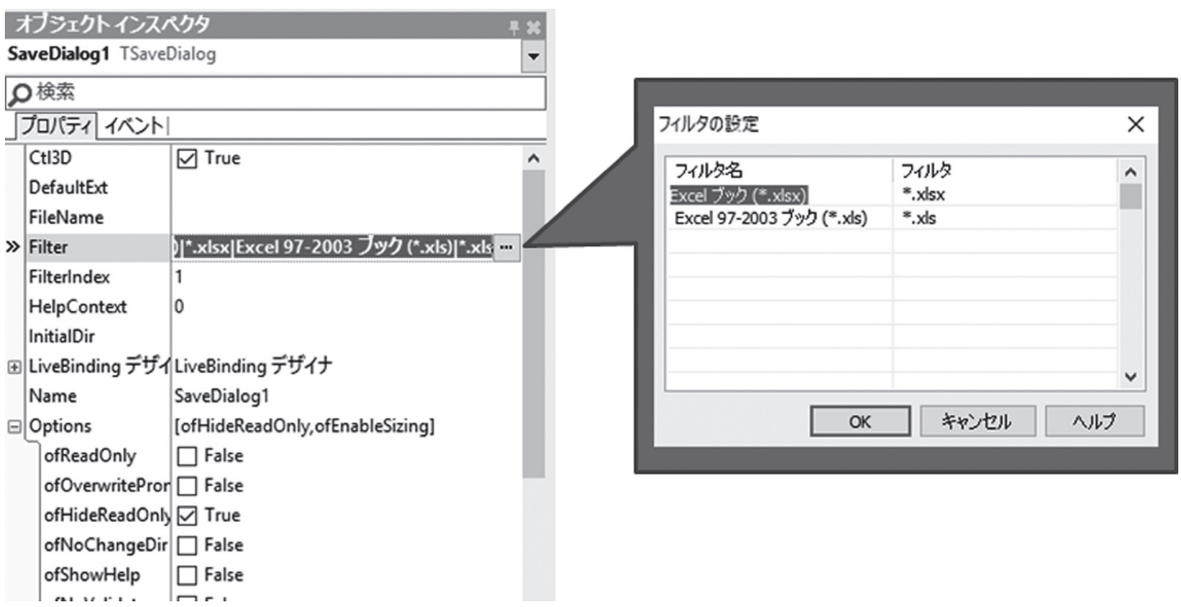
ソース2

```
        // 次データへ
        SQLQuery1.Next;
        // 行数をインクリメント
        Inc(iRow);
        // 営業所が変更されたタイミングまたは最終行で小計を出力
        if (sFL1 <> SQLQuery1.FieldByName('URIFL1').AsString) or (SQLQuery1.Eof) then
        begin
          // 小計を出力
          ovWorkSheet.Cells[iRow, 20].Value := sFL1 + '  小計：';
          ovWorkSheet.Cells[iRow, 24].Value := cShokei;
          // 営業所を保持
          sFL1 := SQLQuery1.FieldByName('URIFL1').AsString;
          // 小計をクリア
          cShokei := 0;
          // 行数をインクリメント
          Inc(iRow);
        end;
      end;

      // Excel保存 …⑥
      if SaveDialog1.Execute then
      begin
        if (LowerCase(ExtractFileExt(SaveDialog1.FileName)) = '.xls') then
        begin
          ovWorkBook.SaveAs(SaveDialog1.FileName, 56);
        end
        else
        begin
          ovWorkBook.SaveAs(SaveDialog1.FileName);
        end;
        ovWorkBook.Saved := True;
      end;

      // Excel表示 …⑦
      ovExcel.Visible := True;
    finally
      // オブジェクト開放 …⑧
      ovWorkSheet := Unassigned;
      ovWorkBook := Unassigned;
      ovExcel := Unassigned;
    end;
  finally
    SQLQuery1.Close;
  end;
end;
```

図2　TSaveDialog設定



Windows10 では動作が安定しないことが多い。現時点（2018 年 8 月現在）では、マイクロソフトより解決方法は明示されておらず、Windows Update による修正も実施されていない。

Delphi/400 からクリップボードを操作する際もこの影響を受ける可能性があるため、本稿ではクリップボードの利用、解説は割愛する。

後者の OLE バリアント配列は、2 次元のバリアント配列を生成し、配列に順番に値をセットすることで、複数列・行の情報を一括で Excel に出力する手法である。

この手法であれば、クリップボードの課題点の影響を受けずにパフォーマンス向上を実現することができるため、本稿では OLE バリアント配列を利用した Excel 出力の方法を解説する。

4.OLEバリアント配列を利用した実装

本章では、実際に OLE バリアント配列を利用したプログラムの実装例として、2 章のソースとの相違点を中心に解説する。（【ソース 3】、【ソース 4】）

①２次元配列用 OLE バリアント変数を定義

変数 ovArray を OleVariant 型で宣言する。

②明細転送用の配列を準備

VarArrayCreate 関数で、変数 ovArray に明細転送用の配列を設定する。1 番目の引数は配列の要素（行および列のインデックス）を指定する。2 番目の引数は配列の要素型（varVariant）を指定する。ここでは Excel テンプレート 1 ページ分（行：24、列：25）で定義している。

③配列へ格納

ここでは Excel のセルに直接値をセットするのではなく、②で準備した配列に格納する。変数 ovArray に行および列のインデックスを指定（Excel のセル位置に該当する箇所）し、IBM i のデータベースより取得した値を順番に配列へ格納する。

④配列より Excel に転送

ovWorkSheet の Range プロパティで配列の行・列の数に合わせて Excel のセル範囲を指定し、変数 ovArray を代入することで、③で配列に格納した値を一括で出力することができる。これにより、複数行・列の出力を 1 回の通信で完了させることが可能である。

通常の静的配列では Range で指定した範囲に配列を代入することができない（型違いでコンパイルエラーとなる）。静的配列でも 1 行単位であれば出力可能だが、その場合は Excel との通信回数が多くなってしまう。そのため、配列は必ず OleVariant 型で定義する必要がある。

⑤配列を解放

VarClear で生成した OLE バリアント配列（ovArray）を解放する。

これで、OLE バリアント配列を利用した Excel 出力処理が完成した。出力結果としては【図 4】と同じになる。

本章の①～⑤で解説した内容が、2 章のプログラムと異なる点である。別途、改ページが必要になる場合はテンプレートのシートを 1 ページ分コピーして最終行からペーストを行うか、もしくはあらかじめ 2 ページ目以降をテンプレート内に作成しておくことで対応が可能である。

また、配列に格納するデータ量に応じてメモリを消費するので、大量に格納してメモリ不足に陥らないためにも、一定のタイミングで出力するよう注意したい。

続いて、本章で作成したプログラムがどの程度パフォーマンスを向上させられたかを検証するため、実際に Excel 出力にかかる時間を計測、比較している。

検証用にデータを 2000 件準備し、それに伴いロジックを一部変更した（【ソース 3】の②で定義している配列の要素を行：2000 に変更）。

そして、2 章と本章のプログラム共に Excel オブジェクト生成からオブジェクト解放まで（＊ソース内コメント参照）にかかる時間を計測した結果が【図 5】である。

「セル単位で転送」ボタンの右側に、第 2 章のセル単位に出力した場合の計測値、「配列で一括転送」ボタンの右側には、本章の OLE バリアント配列を利用して出力した場合の計測値を表示している。

＜実行結果＞
「セル単位で転送」＝ 7.866 秒
「配列利用」＝ 0.964 秒

結果の差からわかるとおり、Excel との通信回数を減らすことで処理時間に明確な効果が出ている。

2 章のセル単位に出力する方法では、単純計算で明細 6 項目 × 2000 行で 12,000 回も Excel と通信を行う。これに対して、本章で解説した OLE バリアント配列を利用したプログラムならば通信回数が 1 回で済むため、パフォーマンスの向上にかなり貢献していることがわかる。

1 回の通信にかかる時間は微々たるものだが、扱うデータの件数や項目数が増えれば増えるほど、この差は顕著になってくるので、本章のテクニックが有効となる。

5.最後に

本稿では、OLE を利用した基本的な Excel 出力の方法と、Excel 出力のパフォーマンスを低下させる要因として、Excel 操作に伴って発生する通信があることを解説した。

OLE バリアント配列を利用する場合としない場合の計測値を比較すれば、Excel との通信回数の低減がパフォーマンス向上に役立つことがご理解いただけたと思う。これは、Excel を操作するプログラム全般で有効なテクニックである。今後、帳票機能を OLE で実装する際は、本稿で解説したパフォーマンス向上テクニックを役立てていただきたく思う。

M

図3　拡張子のエラー

図4　出力結果

売上一覧表（2018年）

名称	住所1	住所2	電話番号	FAX番号	金額
株式会社足立商店	東京都足立区	1－1－2	XXX-XXXX-XXX	XXX-XXXX-XXX	1,000
株式会社足立興業	東京都足立区	1－2－3	XXX-XXXX-XXX	XXX-XXXX-XXX	2,000
株式会社荒川商店	東京都荒川区	2－2－3	XXX-XXXX-XXX	XXX-XXXX-XXX	3,000
株式会社荒川興業	東京都荒川区	1－2－1	XXX-XXXX-XXX	XXX-XXXX-XXX	5,000
株式会社板橋商店	東京都板橋区	5－2－3	XXX-XXXX-XXX	XXX-XXXX-XXX	10,000
				東京営業所 小計：	21,000
株式会社池田商店	大阪府池田市	1－2－3	XXX-XXXX-XXX	XXX-XXXX-XXX	1,000
株式会社泉大津商店	大阪府泉大津市	12－2－3	XXX-XXXX-XXX	XXX-XXXX-XXX	2,000
株式会社泉佐野商店	大阪府泉佐野市	13－2－3	XXX-XXXX-XXX	XXX-XXXX-XXX	3,000
株式会社岸和田商店	大阪府岸和田市	14－2－1	XXX-XXXX-XXX	XXX-XXXX-XXX	4,000
				大阪営業所 小計：	10,000
八幡商店株式会社	京都府八幡市	1－5	XXX-XXXX-XXX	XXX-XXXX-XXX	3,600
株式会社亀岡	京都府亀岡市	5－2－3	XXX-XXXX-XXX	XXX-XXXX-XXX	5,500
				京都営業所 小計：	9,100
株式会社広島商店	広島県広島市	1－1－1	XXX-XXXX-XXX	XXX-XXXX-XXX	1,000
株式会社福山商店	広島県福山市	1－2－2	XXX-XXXX-XXX	XXX-XXXX-XXX	3,000
株式会社尾道商店	広島県尾道市	3－1	XXX-XXXX-XXX	XXX-XXXX-XXX	2,500
				広島営業所 小計：	6,500
株式会社津軽商店	青森県津軽市	1－2－2	XXX-XXXX-XXX	XXX-XXXX-XXX	1,500
株式会社北津軽商店	青森県きた津軽市	1－3－2	XXX-XXXX-XXX	XXX-XXXX-XXX	3,300
				青森営業所 小計：	4,800

ソース3

```
procedure TForm1.Button2Click(Sender: TObject);
var
  ～途中省略～(ソース1と同様)
  ovArray: OleVariant; ...①
begin

  // 出力データ取得
  SQLQuery1.Close;
  SQLQuery1.SQL.Text := 'SELECT * FROM TRURI ORDER BY URIFL1';
  SQLQuery1.Open;
  try
    // Excelオブジェクト生成
    ovExcel := CreateOleObject('Excel.Application');
    try
      // Excel非表示
      ovExcel.Visible := False;

      // 明細転送用の配列を準備 ...②
      ovArray := VarArrayCreate([0, 23, 0, 24], varVariant);

      // フォーマット読み込み
      sFileName := IncludeTrailingPathDelimiter(ExtractFileDir(Application.ExeName)) + 'Format.xlsx';
      ovWorkBook := ovExcel.Workbooks.Open(sFileName);
      ovWorkSheet := ovWorkBook.WorkSheets[1];

      // 出力準備
      iRow := 0;
      cShokei := 0;
      sFL1 := SQLQuery1.FieldByName('URIFL1').AsString;  // 小計出力用営業所

      // データ出力
      while not SQLQuery1.Eof do
      begin
        // 一旦配列に格納 ...③
        ovArray[iRow,  0] := SQLQuery1.FieldByName('URIFL2').AsString;   // 名称
        ovArray[iRow,  7] := SQLQuery1.FieldByName('URIFL3').AsString;   // 住所１
        ovArray[iRow, 12] := SQLQuery1.FieldByName('URIFL4').AsString;   // 住所２
        ovArray[iRow, 17] := SQLQuery1.FieldByName('URIFL5').AsString;   // 電話番号
        ovArray[iRow, 20] := SQLQuery1.FieldByName('URIFL6').AsString;   // FAX番号
        ovArray[iRow, 23] := SQLQuery1.FieldByName('URIFL7').AsCurrency; // 金額

        // 小計を計算
        cShokei := cShokei + SQLQuery1.FieldByName('URIFL7').AsCurrency;
```

ソース4

```
        // 次データへ
        SQLQuery1.Next;
        // 行数をインクリメント
        Inc(iRow);
        // 営業所が変更されたタイミングまたは最終行で小計を出力
        if (sFL1 ◇ SQLQuery1.FieldByName('URIFL1').AsString) or (SQLQuery1.Eof) then
        begin
          // 小計を配列に格納
          ovArray[iRow, 19] := sFL1 + '  小計：';
          ovArray[iRow, 23] := cShokei;

          // 営業所を保持
          sFL1 := SQLQuery1.FieldByName('URIFL1').AsString;

          // 小計をクリア
          cShokei := 0;
          // 行数をインクリメント
          Inc(iRow);
        end;
      end;

      // 配列よりExcelに転送 ...④
      ovWorkSheet.Range[ovWorkSheet.Cells[3, 1], ovWorkSheet.Cells[26, 25]].Value :=
        ovArray;

      // Excel保存
      ～途中省略～(ソース2と同様)

      // Excel表示
      ovExcel.Visible := True;
    finally
      // 配列を開放 ...⑤
      VarClear(ovArray);

      // オブジェクト開放
      ovWorkSheet := Unassigned;
      ovWorkBook := Unassigned;
      ovExcel := Unassigned;
    end;
  finally
    SQLQuery1.Close;
  end;
end;
```

図5　測定結果の比較