吉原 泰介

株式会社ミガロ.

RAD事業部 技術支援課 顧客サポート

Delphi/400によるネイティブ資産の応用活用

ネイティブ資産を有効活用するための実践的なテクニックを紹介する。
SQL では実現が難しいが
ネイティブ資産やコマンドが利用できる Delphi/400 では
簡単に実現できる。

略歴
1978 年 3 月 26 日生れ
2001 年龍谷大学法学部卒
2005 年 07 月株式会社ミガロ. 入社
2005 年 07 月システム事業部配属
2007 年 04 月 RAD 事業部配属

現在の仕事内容
Delphi/400 や JACi400 の製品試験、および月 100 件に及ぶ問い合わせサポートとセミナー講師などを担当している。

1.ネイティブ資産･コマンド

Delphi/400 でアプリケーション開発を行う利点として、IBM i の特有のネイティブ資産やコマンドを活用できるということが挙げられる。例えば、RPG や COBOL プログラムを Delphi/400 から利用できることが、Delphi/400 の大きな特長（機能）である。

もちろん、Delphi/400 では SQL も自由に扱えるため、これらネイティブ資産を使わずともアプリケーションを開発することは可能だ。だが、ネイティブ資産を有効に活用すると、さらにアプリケーション開発の幅を広げることができるのである。

次のようなことを考えたことはないだろうか。

●ライブラリ環境を自由に切り替えられたら・・・
● SQL でメンバが扱えたら・・・
● QUERY 資産を利用できたら・・・
●ライブラリやファイルのリストが取得できたら・・・
●スプールファイルを利用できたら・・・

これらは、IBM i 上では簡単にできることだが、SQL などでは単純に実現できない。逆にいうと、ネイティブ資産やコマンドを利用できる Delphi/400 では、簡単に実現することができるのである。

本稿では、こうした Delphi/400 からネイティブ資産を有効に活用するための実践的なテクニックを紹介していきたい。

2.ネイティブコマンドの活用

2-1.Delphi/400からのコマンド実行

Delphi/400 では、IBM i 上のコマンドを直接実行する機能がある。

具体的には、TAS400 コンポーネントの RemoteCmd メソッド、あるいは TCMD400 コンポーネントからコマンドを実行できる。

この 2 つのコンポーネントの用途の違いは、次の通りである。

● TAS400 コンポーネント RemoteCmd メソッド：パラメータなしのコマンド
● TCMD400 コンポーネント：パラメータを扱うコマンド

ここでは、TAS400 コンポーネントを使って説明する。例えば TAS400 コンポーネントでは、Name を AS4001 とすると、以下のようなコーディングだけで実行できる。

```
AS4001.RemoteCmd（' ネイティブコマンド ');
```

ここまで、Delphi/400 からの、IBM i 上のコマンド実行を紹介した。以降からは、ネイティブコマンドが有効となるような活用実例をいくつか紹介しよう。

2-2.ライブラリ環境に対するコマンド活用例

Delphi/400 で接続しているセッショ

図1

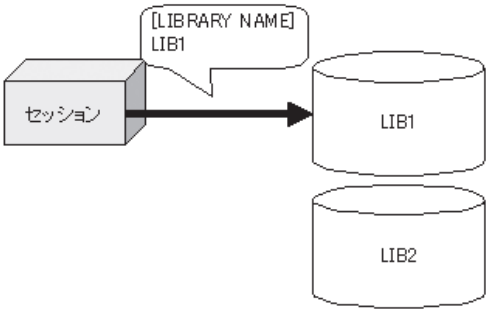
[LIBRARY NAME]
LIB1
セッション
LIB1
LIB2

図2

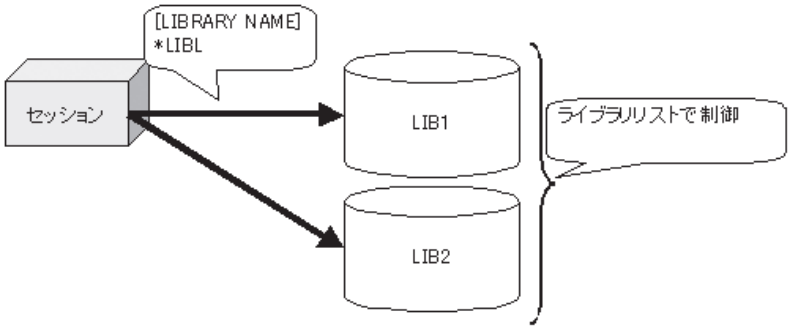
[LIBRARY NAME]
*LIBL
セッション
LIB1
LIB2
ライブラリリストで制御

図3

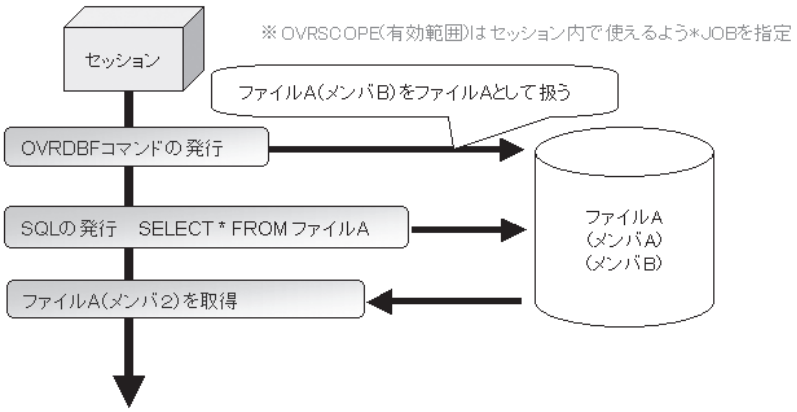
※OVRSCOPE(有効範囲)はセッション内で使えるよう*JOBを指定
セッション
ファイルA(メンバB)をファイルAとして扱う
OVRDBFコマンドの発行
SQLの発行　SELECT * FROM ファイルA
ファイルA(メンバ2)を取得
ファイルA
(メンバA)
(メンバB)

図4


セッション A - [24 x 80]
ファイル(F) 編集(E) 表示(V) 通信(C) アクション(A) ウィンドウ(W) ヘルプ(H)
報告書の表示
QUERY. . . : D4TECLIB/QRYTEST 報告書の幅 . . . . . : 261
行の位置指定 . . . . . 桁移動 . . . . . .
行 ...+...1...+...2...+...3...+...4...+...5...+...6...+...7..
部課 CD 受注No. 営業担当者 受注日 納期日 得意先 CD
000001 101 100,001 1 20,090,815 20,090,907 10,000,000
000002 101 100,002 2 20,090,815 20,090,906 20,000,000
000003 101 100,003 1 20,090,816 20,090,918 10,000,000
000004 101 100,004 1 20,090,817 20,090,901 10,000,000
000005 101 100,005 1 20,090,818 20,090,908 10,000,000
000006 102 100,006 3 20,090,820 20,090,910 30,000,000
000007 102 100,007 3 20,090,821 20,090,908 50,000,000
000008 101 100,008 1 20,090,821 20,090,908 10,000,000
000009 101 100,009 1 20,090,821 20,090,908 10,000,000
000010 102 100,010 3 20,090,822 20,090,924 40,000,000
000011 102 100,011 3 20,090,822 20,090,924 40,000,000
000012 102 100,012 3 20,090,822 20,090,906 50,000,000
000013 101 100,013 2 20,090,825 20,090,926 20,000,000
000014 102 100,014 3 20,090,826 20,090,925 30,000,000
****** ******** 報告書の終わり ********
終わり
F3= 終了 F12= 取り消し F19= 左 F20= 右 F21= 分割
英数 半角
03/032
D02 - セッションが正常に開始されました
FinePrint on FPR5:

ンでは、デフォルトのライブラリが設定されている。

例えばBDEの場合、BDEのエリアスまたはTDatabaseコンポーネントの［LIBRARY NAME］設定である。

通常はここに、アプリケーション上で使用したい参照ライブラリをデフォルトとして設定する。これを設定しておけば、例えばTTableコンポーネントなどでファイルの指定を行う場合、ライブラリ名の指定を省くことができる。【図1】

では、複数のライブラリを使用したい場合はどうすればよいか。もちろんTTableコンポーネントなどで、ファイルを’ライブラリ名/ファイル名’で直接指定することができる。だが、複数ライブラリ間でライブラリ指定を省略したい場合には、［LIBRARY NAME］に’*LIBL’と設定しておくとよいのである。

’*LIBL’と設定した場合、デフォルトの参照ライブラリはどこになるかというと、その接続セッションのライブラリリストが対象となる。【図2】

ここで、ネイティブコマンドを非常に有効に使うことができる。

例えば、LIB1, LIB2というライブラリを参照ライブラリにしたい場合、以下のようなコマンド実行を行うことで、接続中のセッションのライブラリリストにLIB1, LIB2というライブラリを追加することができ、これをデフォルトライブラリとして参照できるようになる。

```
AS4001.RemoteCmd('ADDLIBLE LIB1');
AS4001.RemoteCmd('ADDLIBLE LIB2');
```

つまり、このセッションのライブラリリストをデフォルトライブラリとして使用するので、ネイティブコマンドによって、例えば、本番環境のライブラリと試験環境のライブラリを簡単に切り替えるといったことができるようになるのである。

もちろん、これらのライブラリリストの設定を行うCLプログラムを用意しておき、TCall400コンポーネントを使用して制御することも可能である。

また、ライブラリリストを編集する際にCHGLIBコマンドを使用する場合は、すでに設定されているライブラリがリストから外されてしまう可能性があるので、注意が必要である。

2-3.SQLからメンバを扱うためのコマンド活用例

ファイルのメンバを利用したシステムの場合、SQLでの制約が問題となる。

TTableコンポーネントでファイルを指定する場合、TableNameプロパティに、次のように指定することができる。

```
ライブラリ名/ファイル名(メンバ名)
```

しかし、TQueryコンポーネントなど、SQL上では次のような指定になる。

```
ライブラリ名/ファイル名
```

この場合、メンバを指定することができないため、扱われるメンバは必ずファーストメンバがデフォルトになってしまう。つまり、メンバを利用したシステムにおいて、SQLは使える範囲が限定されてしまう。これは開発効率上、非常に問題がある。

ではSQLで、メンバを扱うためにはどうしたらよいだろうか。方法としては、SQL上ではメンバが指定できないのであれば、その指定をセッション上で事前に設定を行うことで可能にするというのはどうだろう。

ここで、ネイティブコマンドを非常に有効に使うことができる。データベース・ファイル一時変更（OVRDBF）コマンドを使うことで、セッション上でのファイル名の認識をメンバを含めて制御できるのである。

例えば、次のようなOVRDBFコマンドを実行する。

```
AS4001.RemoteCmd(‘OVRDBF
 FILE(ファイルA)
  TOFILE(ライブラリA/ファイルA)
  MBR(メンバB) OVRSCOPE(*JOB)’);
```

こうすると、セッション上でファイルAを扱うと、ファイルA（メンバB）を扱うことができるようになる。これによって、SQL内でメンバが指定できなくとも、実際には特定のメンバに対して処理を行うことができる。【図3】

なお、ここではコマンドでOVRSCOPEを指定しているが、これは、セッション上でOVRDBFを有効にするためである。これを指定しておかないと、実際にSQLで処理をする際に有効とならないので注意が必要である。

2-4.Queryを扱うためのコマンド活用例

IBM iのユーザーは、Query（ここではDelphi/400のTQueryコンポーネントではなくIBM i上のオブジェクト）を、データ抽出や集計といった業務で使用していることが非常に多い。【図4】

もちろんDelphi/400から、同じようなSQLなどを実行すれば、データを取得・集計することはできる。だが、Queryと同じ内容のものを、SQLとしてプログラムを新規に作成する必要がある。しかし、同じ内容のプログラムであれば、すでにIBM i上に存在しているQueryをそのまま利用できるほうがよいだろう。

実は、これもネイティブコマンドを応用活用することで実現することができる。

ただし、Delphi/400から直接Queryを利用するには、1つの課題がある。それは、Queryが対話型ジョブで利用する機能であるという点だ。Delphi/400は対話型CPW値を使用しないバッチ型のジョブであるため、対話型の処理を行うことができない。

では、どうやってDelphi/400からQueryを使用するか。RUNQRYというネイティブコマンドが利用できるのである。

通常、Queryは5250画面上においてRUNQRYを実行すると、結果が表示される。これは対話型で画面情報が返されているからである。しかしこの画面情報を、Delphi/400側では受け取ることができない。そのため、情報のアウトプットをファイルで出力できるように、RUNQRYコマンドのオプションで指定する必要がある。

例えば、Queryを実行した結果をQtemp上に結果ファイルとして出力する場合、次のようなRUNQRYコマンドを実行する。

```
AS4001.RemoteCmd(‘RUNQRY
 QRY(ライブラリ名/Query名)
  OUTTYPE(*OUTFILE)
  OUTFILE(QTEMP/出力ファイル名
  *FIRST *RPLFILE)’);
```

この後に、Delphi/400のTTableコ

図5

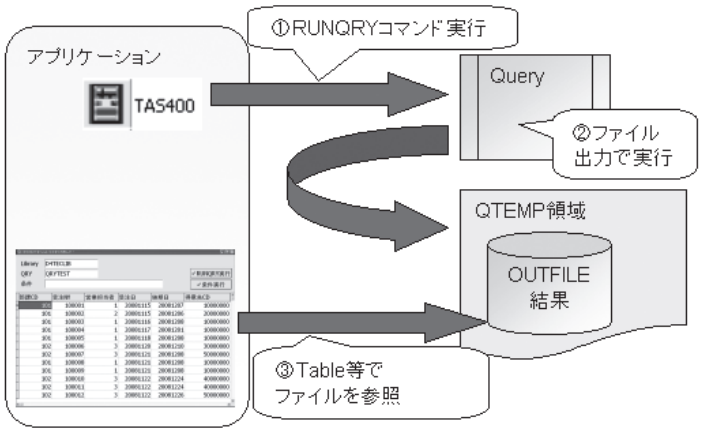

図6

```
procedure TfrmT1.Button1Click(Sender: TObject);
var
  LibraryName, QryName, RUNQRY : String; //Library名, QRY名, RUNQRY実行文
begin
  tblQRY.Close;                  // 使用ファイルをClose
  LibraryName  := EdtLIB.Text;   // Library名
  QryName      := EdtQRY.Text;   // QRY名
  //RUNQRY実行文の編集
  //RUNQRYを実行してQTEMPにQRY名の結果ファイルを作成
  RUNQRY :=  ('RUNQRY QRY(' + LibraryName + '/' + QryName + ') ' +
             'OUTTYPE(*OUTFILE) ' +  'OUTFILE(QTEMP/' + QryName + ' *FIRST *RPLFILE)');
  //RUNQRYの実行
  DMmain.As400.RemoteCmd(RUNQRY);
  //RUNQRYの実行結果ファイルをTableで取得
  tblQRY.TableName := 'QTEMP/' +  QryName;
  tblQRY.Open;
end;
```

図7

関数	機能
TcGetListLib	ライブラリのリストを取得
TcGetListFile	ファイルのリストを取得
TcGetListMbr	メンバのリストを取得
TcGetListDataArea	データエリアのリストを取得
TcGetListDataQueue	データキューのリストを取得
TcGetListOutQueue	アウトキューのリストを取得
TcGetListProg	プログラムのリストを取得

図8

```
procedure TForm1.FormCreate(Sender: TObject);
var
   List1:TStringList;            //Description用
begin
  AS4001.Active := true;         //ASへ接続
  List1 := TStringList.Create;   //Description用リストを作成
  ComboBox1.Items.Clear;         //コンボボックスクリア
  //関数を利用してライブラリのリストをコンボボックスへ設定
  TcGetListLib(AS4001.GetHandle, '*ALL', ComboBox1.Items, List1, 32000);
  List1.Free; //StringListの破棄
end;
```

ンポーネントなどで出力したファイルを読み込めば、Query の実行結果を画面に表示することができる。

なお、出力ファイルを Qtemp 上で扱うことで、出力ファイルの削除などの後処理を不要としている。【図 5】

この仕組みのコーディングは、図 6 のように非常に簡単に実現することができる。【図 6】

ポイントとしては、2 回目以降の実行のことを考慮して、RUNQRY コマンドで *RPLFILE を指定しておく。これにより、同じ Query を実行した場合、結果ファイルを上書きするようにできる。また、RUNQRY コマンド実行時にファイルがつかまれているとエラーの原因となる。そのため、処理の最初に、TTable コンポーネント（このサンプルコードでは tblQRY）は Close しておく必要がある。

3.scdtoolsユニットの活用

3-1.scdtoolsとは

Delphi/400 には、TFile400 コンポーネントの LibraryName プロパティで、ライブラリのリストを検索するダイアログが表示されて選択できる機能がある。

この機能は設計画面上の動作だが、Delphi/400 で実現できる。こうした機能を利用するために、Delphi/400 が提供しているのが「scdtools」である。

3-2.scdtoolsの使い方

「scdtools」はコンポーネントではなく、共通関数を提供するユニットとして存在する。「scdtools」に用意されている主な関数を図 7 に示す。パラメータなど詳しい使い方は、HELP の scdtools にも記載されている。【図 7】

ここでは、ライブラリのリストを取得する例で基本的な活用方法を説明する。「scdtools」の TcGetListLib という関数を活用することで、簡単にライブラリのリストを取得することができる。

次のようなプログラムを作ってみよう。

① Uses 節に scdtools を追記。
② TAS400、TComboBox を画面に配置。
③ FormCreate のイベントにプログラムを記述。【図 8】

画面を起動すると、コンボボックスにライブラリのリストが表示できる。これによって、画面からユーザーが使用するライブラリを選択して、指定することが可能になる。【図 9】

また上記はライブラリのリスト取得の例であるが、同様の使い方でファイルやメンバ、データエリア、データキュー、アウトキュー、プログラムなどのオブジェクトのリストを取得することもできる。動的なプログラムを作成する場合に非常に便利である。

3-3.scdtoolsの応用活用例

この「scdtools」のオブジェクトリストの取得を応用して、スプールファイルの照会画面を作成しよう。完成画面は図 10 に示す。【図 10】

このスプールファイルを照会する画面を作るためには、次の機能の実装が必要となる。

①ライブラリのリスト取得
②アウトキューのリスト取得
③スプールのリスト取得
④スプールの取得

以下順番に、仕組みとコーディングのサンプルを提示する。

①ライブラリのリスト取得
（TComboBox のリストに設定）

これは「scdtools」の TcGetListLib 関数で取得することができる。前述(3-2)をそのまま参考にして実装が可能である。

②アウトキューのリスト取得
（TComboBox のリストに設定）

これは「scdtools」の TcGetListOutqueue 関数で取得することができる。関数の使用方法は TcGetListLib 関数とほぼ同じ。パラメータに、アウトキューを取得する対象のライブラリが増えているだけである。【図 11】

③スプールのリスト取得

スプールリストの取得には、Delphi/400 で専用の TListSpool400 というコンポーネントが用意されているので、これが利用できる。

TListSpool400 コンポーネントのプロパティで、ライブラリとアウトキューを設定する必要がある。①②のコンボボックスで選択されている値を設定して、Active プロパティを True にして接続すれば、TTable のようなデータセットの形でスプールのリストデータを取得することができる。【図 12】

これは TDataSource、TDBGrid コンポーネントでそのまま画面に表示することができる。

④スプールの取得

最後にスプール情報の取得には、これも Delphi/400 で専用の TSpool400 というコンポーネントが用意されているので、これを利用する。

TSpool400 コンポーネントのプロパティで、スプール名、スプールナンバー、ジョブ名、ジョブナンバー、ユーザー名を設定する必要がある。これらの情報は③で取得している TListSpool400 コンポーネントですべて項目として持っているので、その値をプロパティに転送するだけである。【図 13】

そして、TListSpool400 も同様に、Active プロパティを True にして接続すれば、Spool のデータを取得できる。これも TDataSource、TDBGrid コンポーネントでそのまま画面に表示する。

以上で、スプールファイルの照会画面が完成である。【図 10】

4.まとめ

このように Delphi/400 を使用する際にひと工夫すると、IBM i のネイティブ資産の活用範囲をさらに広げることができる。どれも SQL などでは実現が難しい内容だが、Delphi/400 のネイティブコマンドやコンポーネントを使用することで簡単に実現することができる。

本稿では、よく使われる実践的な応用テクニックを紹介した。Delphi/400 でのアプリケーション開発時に、IBM i の資産をより有効に活用する参考にしていただきたい。

M

図9

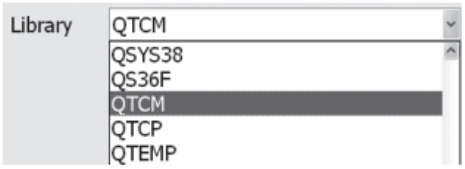

図10

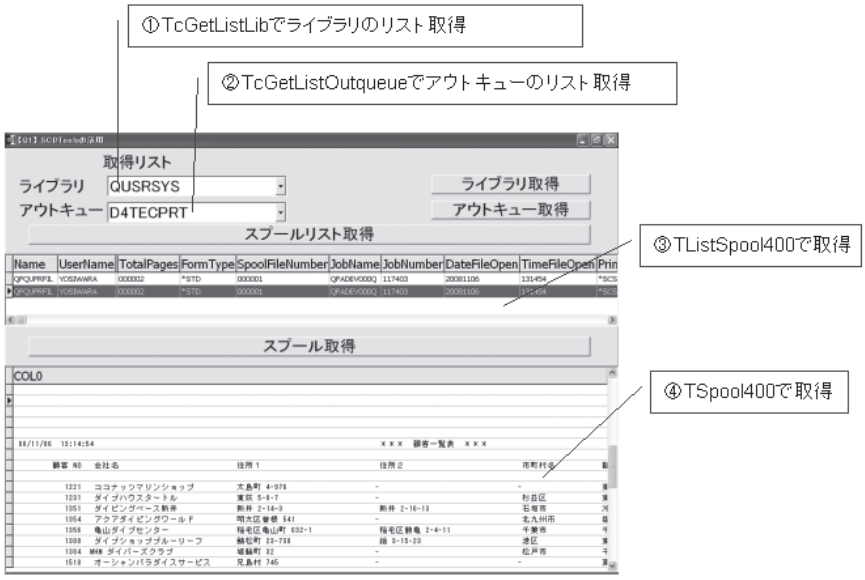

図11

使用コンポーネント
cbLIB:　TcomboboxコンポーネントcbOUTQ:　TcomboboxコンポーネントAS4001:　TAS400コンポーネント

```
procedure TForm1.btnOUTQClick(Sender: TObject);
var
  List1 : TStringList;   //Discription格納用StringList
begin
  //Discription格納用StringListの生成
  List1 := TStringList.Create;
  //アウトキュー用コンボボックスの初期化
  cbOUTQ.Items.Clear;
  //アウトキューリストの取得
  TcGetListOutqueue(As4001.GetHandle,          //接続ハンドル
                    '*ALL'                     //絞込み文字列
                     Trim(cbLIB.Text),         //ライブラリ名
                    cbOUTQ.Items,              //アウトキューリスト（戻り）
                    List1,                     //アウトキュー記述リスト（戻り）
                    32000);                    //バッファサイズ
  List1.Free; //StringListの破棄
end;
```

図12

```
procedure TForm1.btnListSpoolClick(Sender: TObject);
begin
  //TListSpool400のプロパティを設定してリストを取得
  with ListSpool4001 do
  begin
    Active      := false;              //切断
    LibraryName := Trim(cbLIB.Text);   //ライブラリ名
    OutQName    := Trim(cbOUTQ.Text);  //アウトキュー名
    Active      := true;               //接続
  end;
end;
```

使用コンポーネント
ListSpool4001：TListSpool400コンポーネント

図13

```
procedure TForm1.btnSpoolClick(Sender: TObject);
begin
  with Spool4001 do
  begin
    Active      := false;  //切断
    //※ワーク名をクリアしておかないと2回目同じワークとなります。
    WorkFile    := '';
    //スプール名
    SpoolName   := ListSpool4001.FieldByName('Name').AsString;
    //スプールナンバー
    SpoolNumber := ListSpool4001.FieldByName('SpoolFileNumber').AsString;
    //ジョブ名
    JobName     := ListSpool4001.FieldByName('JobName').AsString;
    //ジョブナンバー
    JobNumber   := ListSpool4001.FieldByName('JobNumber').AsString;
    //ユーザー名
    User        := ListSpool4001.FieldByName('UserName').AsString;
    Active      := true;   //接続
  end;
end;
```

使用コンポーネント
Spool4001：TSpool400コンポーネント