4D for OCI: Error Handling

By Josh Fletcher, Technical Support Engineer, 4D Inc.

TN 05-45

Abstract

このテクニカルノートでは、4D for OCI で利用できる 2 種類のエラーハンドリング方法を説明しています。OCI(Oracle Call Interface)における基本的なエラーハンドリングの考え方についても取り上げます。サンプルデータベースは、紹介されている技法の例題です。

Overview of 4D for OCI Error Handling

4D for OCI では、2 種類のエラーハンドリング方法が利用できます。

- 標準的なエラーハンドリング
- 4D エラーハンドリング

前者の標準的なエラーハンドリングは、4D for OCI の関数で返されるエラーコードを調べ、必要に応じて処理をするというものです。一般に、コードの量が多くなる傾向があります。

後者の 4D エラーハンドリングは、ON ERR CALL コマンドを使用する 4D のエラーハンドリングと似ています。4D for OCI プラグインには、OCIOnErrCall というコマンドがあります。

OCIOnErrCall (methodName)

パラメータ	タイプ
methodName	文字列

コマンドによって、OCI をトラップするプロジェクトメソッドがインストールされます。インストールされたプロジェクトメソッドは、エラーハンドリングメソッドと呼ばれます。この方法を用いれば、エラーが発生すると自動的にエラーハンドリングメソッドがコールされるので、余分なコードを記述する手間が省けます。

それぞれの方法については、後で詳述しますが、はじめに OCI プログラミングにおけるエラーハンドリングの考え方について考察したいと思います。

OCI Error Handling

OCI エラーハンドリングには、手がかりとなるふたつの要素が関係しています。

- OCI ファンクションコールの返り値
- OCI が作成するエラーレコードにアクセスするためのエラーハンドル

4D for OCI の返り値は、コマンドの実行に成功したかどうかを確かめるための指標です。返り値の意味については、次の図にまとめてあります。

(出典: "Oracle Call Interface Programmer's Guide Release 8.1.6", December 1999)

OCI Return Code	値	説明
OCI_SUCCESS	0	コマンドの実行は成功
OCI_SUCCESS_WITH_ INFO	1	コマンドの実行は成功: OCIErrorGet()で警告を含む付加的な診断情報を取得
OCI_NO_DATA	100	コマンドの実行は完了: 続くデータなし
OCI_ERROR	-1	コマンドの実行は失敗: OCIErrorGet()で付加的な診断情報を取得
OCI_INVALID_HANDLE	-2	無効なハンドルがパラメータに渡された、またはユーザコールバックに無効なハンドルあるいはコンテキストを渡している: 付加的な診断情報なし
OCI_NEED_DATA	99	アプリケーションのランタイムデータが必要
OCI_STILL_EXECUTING	-3123	サービスコンテキストが非ブロッキングモードで確立され、カレントオペレーションを完了することができない。OCIErrorGet()には ORA-03123 が返される
OCI_CONTINUE	-24200	OCI ライブラリは通常の処理に戻っていない

実際の OCI プログラミングでは、上記の情報だけでは不十分な場合が多く、エラーハンドルの使用が求められることになります。

ほとんどすべての 4D for OCI コマンドはパラメータとしてエラーハンドルを必要としています。エラーハンドルは、エラーレコードを参照するために使用します。

ある種のデータベースエラー(例: OCI_ERROR)が発生すると、OCI はエラーレコードと呼ばれるものを生成します。これらはエラーコードよりも詳細な情報を記録するためのものです。たとえば、エラーコード(例:10003)とエラーテキスト(例:"Invalid table name")が含まれます。単一のエラーハンドルが複数のエラーレコードを指す場合があることに注意してください。

4D for OCI で正確にエラーハンドリングをするには、エラーをチェックするためにエラーハンドルを使用する必要があります。

Standard Error Handling

標準的なエラーハンドリングには、次に挙げる基本的なステップが関係しています。

- 各 4D for OCI コマンドの返り値を調べる
- 必要に応じてエラー処理をする

たとえば次のように記述します。

```
$l_returnValue:=OCIStmtPrepare($l_stmthp;$l_stmt_errhp;$t_SQL;Length($t_SQL))
If ($l_returnValue=OCI_SUCCESS )
    ` Continue OCI calls...
Else
  My_OCI_Error_Handler ($l_returnValue;$l_stmt_errhp)
End if
```

この場合、4D for OCI コマンド OCIStmtPrepare の返り値が調べられているほか、付加的なエラーハンドル($l_stmt_errhp)が渡されており、コマンドの実行に成功しなかった場合のカスタムエラーハンドリングメソッド My_OCI_Error_Handler)で利用されています。

このようなコマンドの成否とハンドルの処理は、OCI コマンドをコールするたびに実行されることになります。したがって、コマンドの数だけ If 文が必要です。

この手法によるエラーハンドリングの長所は、デベロッパが完全にコントロールできるという点です。短所としては、コード量が多く、エラーハンドリングメソッドの変更や複数のエラーハンドルに対応する際の煩わしさが挙げられます。

4D Error Handling

4D エラーハンドリングで必要なのは、エラーハンドリングメソッドをインストールするためのコードだけです。

```
OCIOnErrCall ("My_OCI_Error_Handler")
```

OCI で発生したエラーは、すべてエラーハンドリングメソッドを自動的にコールするトリガとなります。したがって OCI コマンドは次の様式で実行することができます。

$l_returnValue:=OCIStmtPrepare ($l_stmthp;$l_stmt_errhp;$t_SQL;Length($t_SQL))

エラーが発生すれば、自動的にエラーハンドリングメソッドがコールされるので、これ以上のコードは必要ありません。とはいえ、少なくともコマンドの成否を調べるくらいの処理はしておきたいものです。

```
$l_returnValue:=OCIStmtPrepare
($l_stmthp;$l_stmt_errhp;$t_SQL;Length($t_SQL))
If ($l_returnValue=OCI_SUCCESS )
  ` Continue OCI calls...
End if
```

この手法では、エラーが発生した時点で残る OCI の実行は中止されることになります。

この手法の長所は、コード量が少なく、エラーハンドルの変更も簡単な点です。短所としては、エラー処理を 4D for OCI に依存しており、完全にコントロールできない点が挙げられます。

The Custom OCI Error-Handling Method

エラーハンドリングには、OCI コマンドの返り値とエラーハンドルが関係していることについては取り上げました。問題は、取得した情報を利用する方法です。このテクニカルノードのサンプルコードでは、*My_OCI_Error_Handler* メソッドでエラーハンドリングをしています。以下は、メソッドについての詳細です。

Parameters

My_OCI_Error_Handler (returnCode, errorHandle)

パラメータ	タイプ
returnCode	倍長整数
errorHandle	倍長整数

このメソッドは、いずれのエラーハンドリングでも使用することができます。

標準的なエラーハンドリングの場合、パラメータは明示的に渡す必要があります。

```
$l_returnValue:=OCIStmtPrepare
($l_stmthp;$l_stmt_errhp;$t_SQL;Length($t_SQL))
If ($l_returnValue=OCI_SUCCESS )
  ` Continue OCI calls...
Else
  My_OCI_Error_Handler ($l_returnValue;$l_stmt_errhp)
End if
```

この例では、返り値が$l_returnValue に代入され、エラーハンドルは$l_stmt_errhp として渡しています。

4D エラーハンドリングでは、量パラメータは 4D for OCI によって自動的に渡されます。

The Return Code and Error Handle

メソッド側では、エラーコードを調べることによって、エラーレコードの有無を特定しておくと便利です。

```
Case of
  ¥ ($l_returnCode=OCI_ERROR )
  ¥ ($l_returnCode=OCI_SUCCESS_WITH_INFO )
  ¥ ($l_returnCode=OCI_NO_DATA )
  ¥ ($l_returnCode=OCI_INVALID_HANDLE )
  ¥ ($l_returnCode=OCI_NEED_DATA )
  ¥ ($l_returnCode=OCI_STILL_EXECUTING )
  ¥ ($l_returnCode=OCI_CONTINUE )
End case
```

エラーレコードが存在しない場合、一般的なエラーメッセージを出力することができます。

```
¥ ($l_returnCode=OCI_INVALID_HANDLE )
   ALERT("Error:   OCI_INVALID_HANDLE")
```

エラーレコードが利用できれば、そのエラーメッセージを使用することができます。エラーコードが OCI_ERROR または OCI_SUCCESS_WITH_INFO であれば､エラーレコードが存在するはずです。

エラーレコードの取得には OCIErrorGet コマンドを使用します。

OCIErrorGet (hndlp; recordno; errcodep; bufp) _ status

パラメータ	タイプ
hndlp	倍長整数
recordno	倍長整数
errcodep	倍長整数
bufp	文字列
status	倍長整数

OCIErrorGet の第 1 パラメータはエラーハンドルです。第 2 パラメータは取得するエラーレコード番号(1 番から開始)、第 3 パラメータは Oracle エラーコード番号の数値部分(例:エラーコードが ORA-00936 であれば 936)、第 4 パラメータはエラーテキスト(例:"Invalid table name”)です。以下の例では、エラーハンドルを使用してエラーメッセージを生成しています。

```
$l_status:=OCIErrorGet ($l_errorHandle;1;$l_errorCode;$t_errorText)
ALERT("Code:   "+String($l_errorCode)+"¥nMessage: "+$t_errorText)
```

例題では$l_errorHandle にエラーハンドルが渡されています。エラーレコード番号は 1 を直接的に渡しています。エラーコードは$l_errorCode に返され、エラーメッセージは$t_errorText に返されます。

ひとつのエラーハンドルに対して複数のエラーレコードが存在する場合もあるので、ループで処理するのがより確実です。エラーレコード番号を増価することによって、ループを実行することができますが、OCI ではエラーレコードの総数を調べることはできない点に注意してください。代わりに OCI_NO_DATA か返されるまで(OCIErrorGet が失敗する可能性を考えれば OCI_SUCCESS が返されないまでとしたほうがより確実かもしれません) *OCIErrorGet* を繰り返しコールする必要があります。エラーレコード番号がエラーレコードの総数よりも大きい値の場合、OCIErrorGet には OCI_NO_DATA が返されます。

Putting It All Together

以下に *My_OCI_Error_Handler* メソッドの全文を掲載します。

```
  ` ------------------------------------------------
  ` User name (OS): Joshua Fletcher
  ` Date and time: 12/13/05, 15:30:57
  ` ------------------------------------------------
  ` Method: My_OCI_Error_Handler
  ` Description
  `    Error handling method that can be used with OCIOnErrCall().
  `    For error processing all this mehod does is display the errors in an alert.
  `
  ` Parameters
  ` ------------------------------------------------------------------------
  ` C_LONGINT($1) = Return code from the function that reported the error
  `                 (e.g. OCI_ERROR, OCI_INVALID_HANDLE, etc.)
  ` C_LONGINT($2) = OCI error handle (the error handle contains error
  `                 "records" that have the actual error data in them.
  ` ------------------------------------------------------------------------
C_LONGINT($1;$l_returnCode)
C_LONGINT($2;$l_errorHandle)

$l_returnCode:=$1
```

```
$l_errorHandle:=$2

   ` This long integer is used to index through the records diagnostic records
   ` in the error handle. In OCI it is possible for a single error handle to
   ` have multiple error records.
C_LONGINT($l_errRecNum)
   ` The error records start at 1.
$l_errRecNum:=1

   ` Stores return code from any OCI calls in this method.
C_LONGINT($l_status)

   ` The error code of the error in the error record.   E.g. for an
   ` ORA-00936 error the error code will be 936.
C_LONGINT($l_errorCode)

   ` The error message text.
C_TEXT($t_errorText)

   ` This is just used in the ALERT message...
C_TEXT($t_methodName)
$t_methodName:=Current method name

   ` A loop is used because there can be more than one error record...
Repeat
  Case of
    ¥ ($l_returnCode=OCI_ERROR )
        ` To use OCIErrorGet the address of the error handle is needed
        ` and the error record number to load (starts at 1).   The third
        ` and fourth parameters are output variables in which OCI will
        ` place values.
     $l_status:=OCIErrorGet
($l_errorHandle;$l_errRecNum;$l_errorCode;$t_errorText)

        ` If this is false, there are no more error records or another
        ` error occured.
     If ($l_status=OCI_SUCCESS )
       If ($t_errorText="")
         ALERT($t_methodName+"¥nError:                    OCI_ERROR¥nCode:
"+String($l_errorCode)+"¥nMessage: No further information.")
       Else
         ALERT($t_methodName+"¥nError:                    OCI_ERROR¥nCode:
"+String($l_errorCode)+"¥nMessage: "+$t_errorText)
       End if
     End if


    ¥ ($l_returnCode=OCI_SUCCESS_WITH_INFO )
        ` OCI_SUCCESS_WITH_INFO is used to indicate that, while your
        ` OCI operation executed successfully, there is extra
        ` information you may want.
     $l_status:=OCIErrorGet
($l_errorHandle;$l_errRecNum;$l_errorCode;$t_errorText)
     If ($l_status=OCI_SUCCESS )
       ALERT($t_methodName+"¥nCode:          "+String($l_errorCode)+"¥nInfo:
```

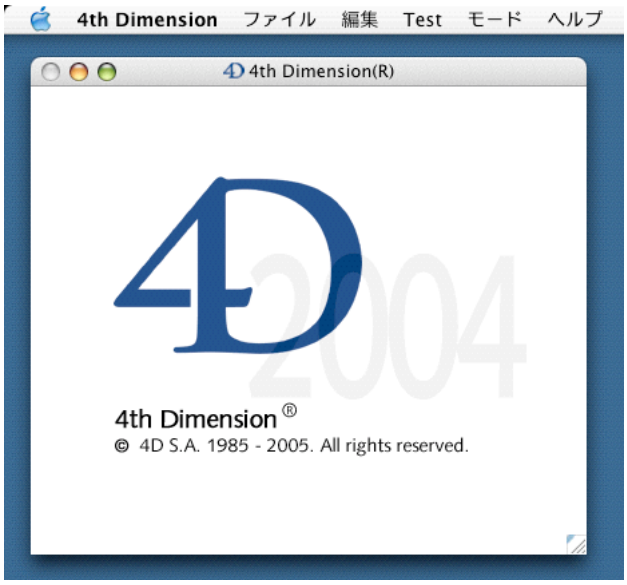
```
"+$t_errorText)
    End if


      ` ------------------------------------------------
      ` The other return codes are "self-explanatory" in that the
      ` name of the constant is meant to convey sufficient
      ` information.   No error records are created for these
      ` errors.   To determine under which conditions these errors
      ` might occur, consult the OCI documentation.
      ` In these cases a generic error message is used...
      ` ------------------------------------------------
  ¥ ($l_returnCode=OCI_NO_DATA )
    ALERT($t_methodName+"¥nError: OCI_NO_DATA")
    $l_status:=OCI_NO_DATA

  ¥ ($l_returnCode=OCI_INVALID_HANDLE )
    ALERT($t_methodName+"¥nError: OCI_INVALID_HANDLE")
    $l_status:=OCI_NO_DATA

  ¥ ($l_returnCode=OCI_NEED_DATA )
    ALERT($t_methodName+"¥nError: OCI_NEED_DATA")
    $l_status:=OCI_NO_DATA

  ¥ ($l_returnCode=OCI_STILL_EXECUTING )
    ALERT($t_methodName+"¥nError: OCI_STILL_EXECUTING")
    $l_status:=OCI_NO_DATA

  ¥ ($l_returnCode=OCI_CONTINUE )
    ALERT($t_methodName+"¥nError: OCI_CONTINUE")
    $l_status:=OCI_NO_DATA
 End case

    ` Increment to the next error record.   You must do this in order
    ` to exit the loop.   There is no way to tell if there are more
    ` error records until you call OCIErrorGet and receive a result
    ` of OCI_NO_DATA.
 $l_errRecNum:=$l_errRecNum+1
Until ($l_status#OCI_SUCCESS )
```

Using the Sample Database

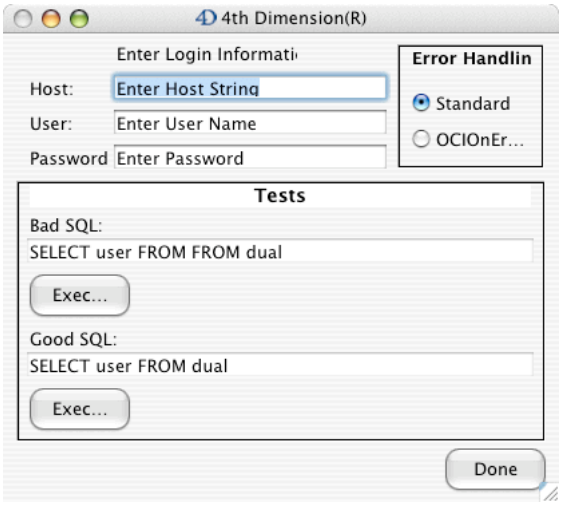
サンプルデータベースに含まれている主な点は次のとおりです。

- Oracle データベースに接続し、SQL ステートメント(良い例および悪い例)を実行するためのダイアログ画面。エラーハンドリング方法は切り替え可能。
- OCI エラーハンドリングのカスタムメソッド。
- 標準的なエラーハンドリングの例題。
- 4D エラーハンドリングの例題。

サンプルデータベースを起動すると 4D スプラッシュスクリーンが表示されます。



Test メニューから Test Dialog を選択するか、Control+T を入力すると Test_Dialog フォームが表示されます。



このダイアログからは 2 種類のテストケースを実行することができます。一方は不正な SQL であり、他方は正しい SQL です。正しい方の SQL はどんな Oracle データベースでも実行できるはずなので、特に Oracle 側をセットアップする必要はありません。

テストを実行するには次のようにします。

- Oracle ホストストリングを入力します。TNS エントリーまたはフルホストストリングのどちらでも構いません。(例: “1.2.3.10:1521/orcl”)
- ユーザ名を入力します。(デフォルトのユーザ名は"scott")
- パスワードを入力します。(デフォルトのパスワードは"tiger")

エラーハンドリングの方法を必要に応じて切り替えます。

Standard-標準的なエラーハンドリングを使用。
OCIOnErrCall-4D エラーハンドリングを使用。

サンプルのテストケースは、どちらのエラーハンドリングを使用しても結果に特筆するような違いはありません。カスタムエラーハンドリングメソッドは両者に共通のものです。違いは、それぞれにおけるコードの書き方にあります。

Tests エリアのインタフェースでは、2 種類のテストを実行することができます。

Bad SQL-不正な SELECT 文を実行
Good SQL-有効な SQL ステートメントを実行

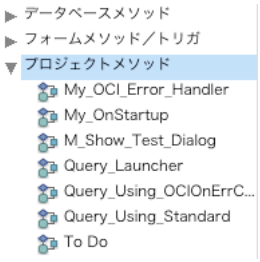
Bad SQL では、次のような警告メッセージが表示されます。

これは SQL ステートメント(“SELECT user FROM FROM dual”)の中に余計な FROM があるためです。この記述では、FROM が不正なテーブル名となってしまいます。もちろん、たまたま FROM というテーブルが存在すれば、別のエラーメッセージが表示されることになります。

Good SQL では、成功したステートメントの結果として次のような警告が表示されます。

テスト画面を終了するには、Done ボタンをクリックします。

次に Control+Y(Windows の場合)またはメニューを選択してデザインモードに移行します。エクスプローラを開いてプロジェクトメソッドを概観してください。



今回の題目と関連があるのは次のメソッドです。

- *My_OCI_Error_Handler*-テスト画面からアクセスできる両方の OCI テストでコールされる汎用エラーハンドリングメソッドです。
- *Query_Using_Standard*-標準的なエラーハンドリングの例題です。
- Query_Using_OCIOnErrCall-4D エラーハンドリングの例題です。

その他のプロジェクトメソッドは、テスト画面の処理に必要なだけで、OCI コマンドとは無関係なものばかりです。

Query_Using_Standard では、OCI コマンドをコールするたびにエラーハンドリングが発生している点に注目してください。

Query_Using_OCIOnErrCall では、一度も *My_OCI_Error_Handler* をコールしていない点に注目してください。これは、エラーが発生した場合、4D が自動的にエラーハンドリングメソッドをコールしてくれるためです。

4D エラーハンドリングにおいては、OCI_SUCCESS 以外はすべて「エラー」扱いになります。OCI_SUCCESS_WITH_INFO は、意味的にはエラーではありませんが、それでも必要な処理ができるようにエラーハンドリングメソッドがコールされます。

Conclusion

このテクニカルノートでは、4D for OCI のエラー処理に必要な知識として OCI エラーハンドリングの概要を取り上げました。4D for OCI で利用できるエラーハンドリングとして 2 種類の手法を比較しながら紹介しました。提供された情報は、4D デベロッパがいずれの手法を採用するか判断し、カスタムエラーハンドリング処理を構築する上で参考になることでしょう。