SYBASE®

ユーザーズ・ガイド

Adaptive Server® Enterprise OLE DB Provider by Sybase

15.5

[Microsoft Windows 版]

ドキュメント ID：DC00499-01-1550-01
改訂：2009 年 10 月

Copyright © 2010 by Sybase, Inc. All rights reserved.

このマニュアルは Sybase ソフトウェアの付属マニュアルであり、新しいマニュアルまたはテクニカル・ノートで特に示されないかぎりは、後続のリリースにも付属します。このマニュアルの内容は予告なしに変更されることがあります。このマニュアルに記載されているソフトウェアはライセンス契約に基づいて提供されるものであり、無断で使用することはできません。

このマニュアルの内容を弊社の書面による事前許可を得ずに、電子的、機械的、手作業、光学的、またはその他のいかなる手段によっても、複製、転載、翻訳することを禁じます。

マニュアルの注文

マニュアルの注文を承ります。ご希望の方は、サイベース株式会社営業部または代理店までご連絡ください。マニュアルの変更は、弊社の定期的なソフトウェア・リリース時にのみ提供されます。

Sybase の商標は、Sybase trademarks ページ (http://www.sybase.com/detail?id=1011207) で確認できます。Sybase およびこのリストに掲載されている商標は、米国法人 Sybase, Inc. の商標です。® は、米国における登録商標であることを示します。

Java および Java 関連の商標は、米国およびその他の国における Sun Microsystems, Inc. の商標または登録商標です。

Unicode と Unicode のロゴは、Unicode, Inc. の登録商標です。

このマニュアルに記載されている上記以外の社名および製品名は、当該各社の商標または登録商標の場合があります。

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目次

はじめに

対象読者

このマニュアルは、Adaptive Server Enterprise OLE DB プロバイダを使用して、Microsoft Windows プラットフォームで Adaptive Server® Enterprise のデータへアクセスする必要のあるアプリケーション開発者を対象としています。

このマニュアルの内容

このマニュアルは、次のように構成されています。

- 「第 1 章 Adaptive Server OLE DB プロバイダの概要」では、OLE DB プログラミングおよびプログラム・サンプルについて説明します。
- 「第 2 章 データベースへの接続」では、Adaptive Server OLE DB プロバイダを使用して Adaptive Server に接続する方法について説明します。
- 「第 3 章 サポートされている Adaptive Server の機能」では、Adaptive Server OLE DB プロバイダでサポートされる Adaptive Server の高度な機能について説明します。

関連マニュアル

詳細については、これらのマニュアルを参照してください。

- 使用しているプラットフォームの『Software Developer's Kit リリース・ノート』には、Adaptive Server OLE DB プロバイダおよび Software Developer's Kit (SDK) に関する重要な最新情報が記載されています。
- 『Software Developer's Kit/Open Server インストール・ガイド』では、SDK および Adaptive Server OLE DB プロバイダのコンポーネントのインストールについて説明します。
- 『ASE インストール・ガイド』では、Adaptive Server のインストールについて説明します。
- 使用しているプラットフォームの Adaptive Server Enterprise の『リリース・ノート』では、Adaptive Server の既知の問題および更新の詳細について説明します。

その他の情報

Sybase® Getting Started CD、SyBooks™ CD、Sybase Product Manuals Web サイトを利用すると、製品について詳しく知ることができます。

- Getting Started CD には、PDF 形式のリリース・ノートとインストール・ガイド、SyBooks CD に含まれていないその他のマニュアルや更新情報が収録されています。この CD は製品のソフトウェアに同梱されています。Getting Started CD に収録されているマニュアルを参照または印刷するには、Adobe Acrobat Reader が必要です (CD 内のリンクを使用して Adobe の Web サイトから無料でダウンロードできます)。

- SyBooks CD には製品マニュアルが収録されています。この CD は製品のソフトウェアに同梱されています。Eclipse ベースの SyBooks ブラウザを使用すれば、使いやすい HTML 形式のマニュアルにアクセスできます。

  一部のマニュアルは PDF 形式で提供されています。これらのマニュアルは SyBooks CD の PDF ディレクトリに収録されています。PDF ファイルを開いたり印刷したりするには、Adobe Acrobat Reader が必要です。

  SyBooks をインストールして起動するまでの手順については、Getting Started CD の『SyBooks インストール・ガイド』、または SyBooks CD の『*README.txt*』ファイルを参照してください。

- Sybase Product Manuals Web サイトは、SyBooks CD のオンライン版であり、標準の Web ブラウザを使用してアクセスできます。また、製品マニュアルのほか、EBFs/Updates、Technical Documents、Case Management、Solved Cases、ニュース・グループ、Sybase Developer Network へのリンクもあります。

  Technical Library Product Manuals Web サイトにアクセスするには、Product Manuals (http://www.sybase.com/support/manuals/) にアクセスしてください。

Web 上の Sybase 製品の動作確認情報

Sybase Web サイトの技術的な資料は頻繁に更新されます。

❖ **製品認定の最新情報にアクセスする**

1. Web ブラウザで Technical Documents を指定します。(http://www.sybase.com/support/techdocs/)
2. [Partner Certification Report] をクリックします。
3. [Partner Certification Report] フィルタで製品、プラットフォーム、時間枠を指定して [Go] をクリックします。
4. [Partner Certification Report] のタイトルをクリックして、レポートを表示します。

❖ **コンポーネント認定の最新情報にアクセスする**

1. Web ブラウザで Availability and Certification Reports を指定します。(http://certification.sybase.com/)
2. [Search By Base Product] で製品ファミリとベース製品を選択するか、[Search by Platform] でプラットフォームとベース製品を選択します。
3. [Search] をクリックして、入手状況と認定レポートを表示します。

❖ **Sybase Web サイト (サポート・ページを含む) の自分専用のビューを作成する**

MySybase プロファイルを設定します。MySybase は無料サービスです。このサービスを使用すると、Sybase Web ページの表示方法を自分専用にカスタマイズできます。

1 Web ブラウザで Technical Documents を指定します。(http://www.sybase.com/support/techdocs/)

2 [MySybase] をクリックし、MySybase プロファイルを作成します。

Sybase EBF とソフトウェア・メンテナンス

❖ **EBF とソフトウェア・メンテナンスの最新情報にアクセスする**

1 Web ブラウザで Sybase Support Page (http://www.sybase.com/support) を指定します。

2 [EBFs/Maintenance] を選択します。MySybase のユーザ名とパスワードを入力します。

3 製品を選択します。

4 時間枠を指定して [Go] をクリックします。EBF/Maintenance リリースの一覧が表示されます。

鍵のアイコンは、「Technical Support Contact」として登録されていないため、一部の EBF/Maintenance リリースをダウンロードする権限がないことを示しています。未登録でも、Sybase 担当者またはサポート・コンタクトから有効な情報を得ている場合は、[Edit Roles] をクリックして、「Technical Support Contact」の役割を MySybase プロファイルに追加します。

5 EBF/Maintenance レポートを表示するには [Info] アイコンをクリックします。ソフトウェアをダウンロードするには製品の説明をクリックします。

表記規則

このマニュアルで使用されている表記規則は次のとおりです。

- 関数、コマンド名、コマンド・オプション名、プログラム名、プログラム・フラグ、プロパティ、キーワード、文、ストアド・プロシージャは次の形式で表記されます。

  IDBCreateSession::CreateSession() を使用してセッションを作成できます。

- 変数、パラメータ、ユーザが指定する語は、構文内と本文中では次のように斜体で表記されます。

  例：文 `int` ***`RowCount;`*** *RowCount*; は int 型の変数。

- データベース、テーブル、カラム、データ型などのデータベース・オブジェクトの名前は、次のように表記されます。

  pubs2 オブジェクトの値。

- 関数の用途を示す例は、次のように表記されます。

```
ICommandText* pICommandText = NULL;
HRESULT hr = pIDBCreateCommand->CreateCommand(NULL,
   IID_ICommandText, (IUnknown**)&pICommandText);
pIDBCreateCommand->Release();
```

次の表は、構文の表記規則をまとめたものです。

表 *1:* 構文の表記規則

キー	定義
{ }	中カッコは、その中のオプションを 1 つ以上選択しなければならないことを意味する。コマンドには中カッコは入力しない。
[ ]	角カッコは、オプションを選択しても省略してもよいことを意味する。コマンドには角カッコは入力しない。
|	縦線は、中カッコまたは角カッコの中の複数のオプションのうち 1 つだけを選択できることを意味する。
,	カンマは、中カッコまたは角カッコの中のオプションをいくつでも選択できることを意味する。複数のオプションを選択する場合には、オプションをカンマで区切る。 カンマは他の構文内容で必須になることもある。
( )	このカッコはコマンドの一部として入力する。
. . .	省略記号 (...) は、直前の要素を必要な回数だけ繰り返し指定できることを意味する。省略記号はコマンドには入力しない。

アクセシビリティ機能

このマニュアルには、アクセシビリティを重視した HTML 版もあります。この HTML 版マニュアルは、スクリーン・リーダーで読み上げる、または画面を拡大表示するなどの方法により、その内容を理解できるよう配慮されています。

Software Developer's Kit バージョン 15.5 と HTML マニュアルは、連邦リハビリテーション法第 508 条のアクセシビリティ規定に準拠していることがテストにより確認されています。第 508 条に準拠しているマニュアルは通常、World Wide Web Consortium (W3C) の Web サイト用ガイドラインなど、米国以外のアクセシビリティ・ガイドラインにも準拠しています。

この製品のオンライン・ヘルプは HTML でも提供され、スクリーン・リーダーの読み上げで内容を理解できる機能があります。

注意 アクセシビリティ・ツールを効率的に使用するには、設定が必要な場合もあります。一部のスクリーン・リーダーは、テキストの大文字と小文字を区別して発音します。たとえば、すべて大文字のテキスト (ALL UPPERCASE TEXT など) はイニシャルで発音し、大文字と小文字の混在したテキスト (Mixed Case Text など) は単語として発音します。構文規則を発音するようにツールを設定すると便利かもしれません。詳細については、ツールのマニュアルを参照してください。

Sybase のアクセシビリティに対する取り組みについては、Sybase Accessibility (http://www.sybase.com/accessibility) を参照してください。Sybase Accessibility サイトには、第 508 条と W3C 標準に関する情報へのリンクもあります。

不明な点があるときは

Sybase ソフトウェアがインストールされているサイトには、Sybase 製品の保守契約を結んでいるサポート・センタとの連絡担当の方 (コンタクト・パーソン) を決めてあります。マニュアルだけでは解決できない問題があった場合には、担当の方を通して Sybase のサポート・センタまでご連絡ください。

第 1 章

Adaptive Server OLE DB プロバイダの概要

この章では、OLE DB インタフェースを使用して Microsoft ADO (ActiveX Data Objects) プログラミング環境から Adaptive Server 機能への完全なアクセス権を取得する方法を説明します。

OLE DB インタフェースを使用する多くのアプリケーションでは、直接ではなく、ADO プログラミング・モデルを通じて処理を行います。この章では、Adaptive Server に対する ADO プログラミングについても説明します。

OLE DB の概要

OLE DB は、Microsoft のデータ・アクセス・モデルです。OLE DB は、COM (Component Object Model) インタフェースを使用します。ODBC とは異なり、データ・ソースでの SQL クエリ・プロセッサの使用を前提としません。

各 OLE DB プロバイダはダイナミック・リンク・ライブラリです。アクセスするデータ・ソースの種類ごとに、個別の OLE DB プロバイダが必要です。Adaptive Server にアクセスするには、次の 2 つの OLE DB プロバイダを使用できます。

- **Sybase Adaptive Server OLE DB プロバイダ** Adaptive Server OLE DB プロバイダは、OLE DB 2.5 およびそれ以降で動作するように設計されています。ODBC コンポーネントを使用せずに、OLE DB データ・ソースとして Adaptive Server へのアクセスを実現します。このプロバイダの短い名は ASEOLEDB です。
- **Microsoft OLE DB provider for ODBC** Microsoft が提供する OLE DB プロバイダです。短い名前は MSDASQL です。MSDASQL プロバイダは、ODBC データ・ソースを OLE DB データ・ソースのように見せます。そのためには、Adaptive Server ODBC ドライバが必要です。

Adaptive Server OLE DB プロバイダを使用することにより、次の利点が得られます。

- 配備時に ODBC は必要ありません。
- OLE DB プログラミング環境から Adaptive Server 機能への完全なアクセス権を取得できます。MSDASQL プロバイダにより、OLE DB クライアントで任意の ODBC ドライバを使用できますが、各 ODBC ドライバの全範囲の機能の可用性が保証されるわけではありません。

サポートされるプラットフォーム

Adaptive Server OLE DB プロバイダを使用できるプラットフォームのリストは、『Open Server および SDK 新機能 』(各 Windows、Linux、UNIX、Mac OS X 版) を参照してください。

Adaptive Server OLE DB プロバイダによる ADO プログラミング

ActiveX Data Objects (ADO) は、Automation インタフェースを通じて公開されるデータ・アクセス・オブジェクト・モデルであり、クライアント・アプリケーションでオブジェクトについて事前に想定せずに実行時にオブジェクトのメソッドとプロパティを検出できます。Automation により、Visual Basic などのスクリプト言語で標準のデータ・アクセス・オブジェクト・モデルを使用できます。ADO では、異なるデータベース上のデータへのアクセスを提供するために OLE DB を使用します。

Adaptive Server OLE DB プロバイダを使用すると、ADO プログラミング環境から Adaptive Server の機能への完全なアクセス権を取得できます。

この項では、Visual Basic から ADO を使用して基本タスクを実行する方法について説明します。ここでは、ADO を使用するプログラミングの詳細については説明しません。ADO でのプログラミングについては、使用する開発ツールのマニュアルを参照してください。

Connection オブジェクトを使用したデータベースへの接続

この項では、データベースに接続する簡単な Visual Basic ルーチンについて説明します。

サンプル・コード

Command1 という名前のコマンド・ボタンをフォームに配置し、このルーチンをコマンド・ボタンの Click イベントに貼り付けることにより、ルーチンをテストできます。プログラムを実行し、[Command1] をクリックして接続した後、切断します。

```
Private Sub cmdTestConnection_Click()
   ' Declare variables
   Dim myConn As New ADODB.Connection
   On Error GoTo HandleError

   ' Establish the connection
   myConn.Provider = "ASEOLEDB"
   myConn.ConnectionString = "Data Source=MANGO:5000;User ID=sa;Pwd=;"
   myConn.Open
   MsgBox "Connection succeeded"
   myConn.Close
   Exit Sub

HandleError:
   MsgBox "Connection failed"
   Exit Sub
End Sub
```

注意

このサンプルは、次のタスクを実行します。

- ルーチン内で使用する変数を宣言します。
- Adaptive Server OLE DB プロバイダを使用して、サンプル・データベースへの接続を確立します。
- 接続をクローズします。

ASEOLEDB プロバイダは、インストール時に自身を登録します。この登録プロセスでは、レジストリの COM セクションにレジストリ エントリを作成し、ASEOLEDB プロバイダが呼び出されたときに ADO がこの DLL を見つけられるようにします。DLL の場所を変更した場合は、次の手順に従ってその場所を登録する必要があります。

❖ Adaptive Server OLE DB プロバイダの登録

1. コマンド・プロンプトを開きます。
2. Adaptive Server OLE DB プロバイダがインストールされているディレクトリに移動します。
3. 次のコマンドを実行してプロバイダを登録します。

```
regsvr32 sybdrvoledb.dll
```

Command オブジェクトを使用した文の実行

この項では、データベースに簡単な SQL 文を送信する簡単なルーチンについて説明します。

サンプル・コード

Command2 という名前のコマンド・ボタンをフォームに配置し、このルーチンをコマンド・ボタンの Click イベントに貼り付けることにより、ルーチンをテストできます。プログラムを実行し、[Command2] をクリックして接続し、データベース・サーバ・ウィンドウにメッセージを表示した後、切断します。

```
Private Sub cmdUpdate_Click()
   ' Declare variables
   Dim myConn As New ADODB.Connection
   Dim myCommand As New ADODB.Command
   Dim cAffected As Long

   ' Establish the connection
   myConn.Provider = "ASEOLEDB"
   myConn.ConnectionString = "Data Source = MANGO:5000; User ID=sa;PWD=;"+_
      "Initial Catalog=pubs2;"
   myConn.Open

   'Execute a command
   myCommand.CommandText = "INSERT INTO publishers values" +_
      "('7777', 'American Books', 'Boston', 'MA')"
   Set myCommand.ActiveConnection = myConn
   myCommand.Execute cAffected
   MsgBox CStr(cAffected) + " rows affected.", vbInformation
   myConn.Close
End Sub
```

注意

サンプル・コードは、接続が確立された後、Command オブジェクトを作成し、その CommandText プロパティに insert 文、ActiveConnection プロパティに現在の接続を設定します。次に、insert 文を実行し、更新の影響を受けた行数をメッセージ・ボックスに表示します。

この例では、insert 文がデータベースに送信され、実行されるとすぐにコミットされます。

Recordset オブジェクトによるデータベースのクエリ

ADO Recordset オブジェクトは、クエリの結果セットを表します。これを使用して、データベースのデータを表示できます。

サンプル・コード

cmdQuery という名前のコマンド・ボタンをフォームに配置し、このルーチンをコマンド・ボタンの Click イベントに貼り付けることにより、ルーチンをテストできます。プログラムを実行し、[CmdQuery] をクリックして接続します。次に、データベース・サーバ・ウィンドウにメッセージを表示し、クエリを実行して最初の数行をメッセージ・ボックスに表示した後、切断します。

```
Private Sub cmdQuery_Click()
   ' Declare variables
   Dim myConn As New ADODB.Connection
   Dim myCommand As New ADODB.Command
   Dim myRS As New ADODB.Recordset
   On Error GoTo ErrorHandler

   ' Establish the connection
   myConn.Provider = "ASEOLEDB"
   myConn.ConnectionString = "Data Source = MANGO:5000; User ID=sa;PWD=;" +
      "Initial Catalog=pubs2;"
   myConn.Open

   'Execute a query
   Set myRS = New Recordset
   myRS.CacheSize = 50
   myRS.Source = "Select * from customer"
   myRS.ActiveConnection = myConn
   myRS.LockType = adLockOptimistic
   myRS.Open

   'Scroll through the first few results
   For i = 1 To 5
     MsgBox myRS.Fields("company_name"), vbInformation
     myRS.MoveNext
   Next
   myRS.Close
   myConn.Close
   Exit Sub
ErrorHandler:
   MsgBox Error(Err)
   Exit Sub
End Sub
```

注意

この例の Recordset オブジェクトは、Customer テーブルに対するクエリの結果を保持します。For ループは、最初の数行をスクロールし、各行の “company_name” 値を表示します。

次に、ADO からカーソルを使用する簡単な例を示します。

Rowset オブジェクトの使用

Adaptive Server を使用する場合、ADO の Rowset がカーソルを表します。Rowset をオープンする前に Rowset オブジェクトの CursorType プロパティを宣言することにより、カーソルのタイプを選択できます。カーソル・タイプを選択することにより、Rowset に対して適用できる動作を制御できます。また、この選択はパフォーマンスに影響します。

カーソル・タイプ

Adaptive Server によりサポートされる一連のカーソル・タイプについては、『Adaptive Server Enterprise Transact-SQL ユーザーズ・ガイド』を参照してください。

ADO は、カーソル・タイプについて独自の命名規則を持ちます。次に、使用可能なカーソル・タイプと対応するカーソル・タイプの定数、それらに相当する Adaptive Server タイプを示します。

ADO カーソル・タイプ	ADO 定数	ASE 型
静的カーソル	adOpenStatic	非反映型カーソル
前方向のみ	adOpenForwardOnly	非スクロール・カーソル
スクロール可能	adOpenStatic	スクロール可能

サンプル・コード

次のコードは、ADO の Rowset オブジェクトのカーソル・タイプを設定しています。

```
Dim myRS As New ADODB.Rowset myRS.CursorType=_
   adOpenForwardOnly
```

カーソルを使用したデータの更新

Adaptive Server OLE DB プロバイダを使用すると、カーソルを使って結果セットを更新することができます。この機能は、MSDASQL プロバイダでは使用できません。

レコード・セットの更新

レコード・セットを使用して、データベースを更新することができます。

```
Private Sub Command6_Click()
   Dim myConn As New ADODB.Connection
   Dim myRS As New ADODB.Recordset
   Dim SQLString As String
   ' Connect
   myConn.Provider = "ASEOLEDB"
   myConn.ConnectionString = "Data Source=MANGO:5000;User ID=sa;Pwd=;"
   myConn.Open
   myConn.BeginTrans
   SQLString = "Select * from customer"
   myRS.Open SQLString, myConn, adOpenDynamic, adLockBatchOptimistic
   If myRS.BOF And myRS.EOF Then
      MsgBox "Recordset is empty!", 16, "Empty Recordset"
   Else
      MsgBox "Cursor type:" + CStr(myRS.CursorType), vbInformation
      myRS.MoveFirst
         For i = 1 To 3
            MsgBox "Row:" + CStr(myRS.Fields("id")), vbInformation
            If i = 2 Then
               myRS.Update "City", "Toronto"
               myRS.UpdateBatch
            End If
      myRS.MoveNext
```

```
        Next i '
        myRS.Close
    End If
    myConn.CommitTrans
    myConn.Close
End Sub
```

注意

レコード・セットの設定に adLockBatchOptimistic を使用する場合、myRS.Update メソッドではデータベース自体を変更できません。代わりに、Recordset のローカル・コピーを更新します。

myRS.UpdateBatch メソッドは、データベース・サーバを更新しますが、トランザクション内なのでその処理をコミットしません。UpdateBatch メソッドがトランザクション外から呼び出された場合は、変更がコミットされます。

myConn.CommitTrans メソッドは、変更をコミットします。Recordset オブジェクトはその時点までにクローズされるため、データのローカル・コピーが変更されるかどうかは問題ではありません。

トランザクションの使用

デフォルトでは、ADO を使用してデータベースに加えた変更はすべて、実行するとすぐにコミットされます。これには、Recordset の UpdateBatch メソッドと同様、明示的な更新が含まれます。ただし前の項では、Connection オブジェクトの BeginTrans メソッドと RollbackTrans メソッド、または CommitTrans メソッドを使用してトランザクションを使用できることを示しました。

トランザクションの独立性レベルは、Connection オブジェクトのプロパティとして設定されます。IsolationLevel プロパティは、次のいずれかの値になります。

ADO 独立性レベル	定数	ASE レベル
未指定	adXactUnspecified	適用されない。0 に設定。
Chaos	adXactChaos	サポートされない。0 に設定。
Browse	adXactBrowse	0
Read uncommitted	adXactReadUncommitted	0
Cursor stability	adXactCursorStability	1
Read committed	adXactReadCommitted	1
Repeatable read	adXactRepeatableRead	2
Isolated	adXactIsolated	3
Serializable	adXactSerializable	3

サポートされている OLE DB インタフェース

OLE DB API は、一連のインタフェースで構成されます。表 1-1 に、Adaptive Server OLE DB プロバイダ内の各インタフェースに対するサポートについて示します。

表 *1-1:* サポートされている *OLE DB* インタフェース

インタフェース	目的	制限事項
IAccessor	クライアント・メモリとデータ・ストア値の間のバインドを定義する。	DBACCESSOR_PASSBYREF はサポートされない。 DBACCESSOR_OPTIMIZED はサポートされない。
IColumnsInfo	rowset のカラムに関する簡単な情報を取得する。	適用せず
IColumnsRowset	rowset 内のオプションのメタデータ・カラムに関する情報を取得し、カラムのメタデータの rowset を取得する。	適用せず
ICommand	SQL コマンドを実行する。	設定できなかったプロパティを検索するために、DBPROPSET_PROPERTIESINERROR を指定して ICommandProperties: GetProperties を呼び出すことはサポートされない。
ICommandPrepare	コマンドを準備する。	適用せず
ICommandProperties	コマンドにより作成される rowset の Rowset プロパティを設定する。最も一般的には、rowset がサポートするインタフェースを指定するために使用する。	適用せず
ICommandText	ICommand の SQL コマンド・テキストを設定する。	DBGUID_DEFAULT SQL 言語のみがサポートされる。
ICommandWithParameters	コマンドのパラメータ情報を設定または取得する。	スカラ値のベクトルとして保存されたパラメータはサポートされない。
IConvertType		適用せず
IDBCreateCommand	セッションからコマンドを作成する。	適用せず
IDBCreateSession	データ・ソース・オブジェクトからセッションを作成する。	適用せず
IDBInfo	このプロバイダに固有のキーワードに関する情報を検索する (つまり、非標準の SQL キーワードを検索する)。 また、リテラル、クエリに一致するテキスト内で使用されている特殊文字、その他のリテラルに関する情報を検索する。	適用せず
IDBInitialize	データ・ソース・オブジェクトと列挙型を初期化する。	適用せず
IDBProperties	データ・ソース・オブジェクトまたは列挙型のプロパティを管理する。	適用せず

インタフェース	目的	制限事項
IDBSchemaRowset	システム・テーブルに関する情報を標準的な形式 (rowset) で取得する。	適用せず
IErrorLookup IErrorRecords	ActiveX エラー・オブジェクトをサポートする。	適用せず
IGetDataSource	セッションのデータ・ソース・オブジェクトへのインタフェース・ポインタを返す。	適用せず
IMultipleResults	コマンドから複数の結果 (rowset またはロー・カウント) を取得する。	適用せず
IOpenRowset	SQL 以外の方法で、データベース・テーブルにその名前によりアクセスする。	GUID ではなく、名前によるテーブルのオープンがサポートされる。
IRowset	rowset にアクセスする。	適用せず
IRowsetChange、IRowsetUpdate	rowset へのデータ変更をデータ・ストアに反映させる。 BLOB に対する InsertRow/SetData は未実装。	適用せず
IRowsetIdentity	ロー・ハンドルを比較する。	適用せず
ISequentialStream	BLOB カラムを取得する。	読み込みのみでサポートされる。 このインタフェースで SetData はサポートされない。
ISessionProperties	セッション・プロパティ情報を取得する。	適用せず
ISourcesRowset	データ・ソース・オブジェクトと列挙型の rowset を取得する。	適用せず
ITableDefinition	制約付きのテーブルを作成、削除、変更する。	適用せず
ITransaction	トランザクションをコミットまたはアボートする。	すべてのフラグがサポートされるわけではない。
ITransactionLocal	セッションに対するトランザクションを処理する。 すべてのフラグがサポートされるわけではない。	適用せず

Adaptive Server OLE DB プロバイダによる OLE DB プログラミング

この項では、Adaptive Server OLE DB プロバイダの使用中に OLE DB の基本タスクを実行する方法について説明します。

OLE DB を使用したデータ・ソースへの接続

次に、OLE DB インタフェースを使用して Adaptive Server データベースへの接続を確立する方法について説明します。

次に説明するように、OLE DB を使用した接続の設定には 2 つの方法があります。

- **IDBInitialize を使用したデータ・ソースへの接続**
  1. CoCreateInstance を呼び出します。
  2. CLSIDFromProgID("ASEOLEDB") から取得した clsid を渡します。
  3. IDBInitialize を使用して接続プロパティを設定します。
- **IDataInitialize を使用したデータ・ソースへの接続**
  1. CoCreateInstance を呼び出します。
  2. MSDAINITIALIZE から取得した clsid を渡します。
  3. IDataInitialize を使用して接続プロパティを設定します。

コード例　次は、OLE DB 接続を確立するためのコード例です。

```
wchar_t* szInitializationString = L"Provider=ASEOLEDB;
   User ID=sa;Password=;Initial Catalog=pubs2;
   Data Source=MANGO:5000;"

IDataInitialize* pIDataInitialize = NULL;
HRESULT hr = CoCreateInstance(
     __uuidof(MSDAINITIALIZE), NULL, CLSCTX_ALL,
     __uuidof(IDataInitialize), (void**)&pIDataInitialize);

IDBInitialize* pIDBInitialize = NULL;
hr = pIDataInitialize->GetDataSource(NULL, CLSCTX_ALL,
     szInitializationString,
     __uuidof(IDBInitialize), (IUnknown**)&pIDBInitialize);
hr = pIDBInitialize->Initialize();

IDBCreateSession* pIDBCreateSession = NULL;
hr = pIDBInitialize->QueryInterface(
     IID_IDBCreateSession, (void**)&pIDBCreateSession);

IDBCreateCommand* pIDBCreateCommand = NULL;
hr = pIDBCreateSession->CreateSession(NULL,
     IID_IDBCreateCommand,
     (IUnknown**)&pIDBCreateCommand);

ICommandText* pICommandText = NULL;
hr = pIDBCreateCommand->CreateCommand(NULL,
     IID_ICommandText, (IUnknown**)&pICommandText);

// use the command object
// ...

pICommandText->Release();

pIDBCreateSession->Release();
pIDBCreateCommand->Release();
pIDBInitialize->Release();
pIDataInitialize->Release();
```

OLE DB アプリケーション内でのスレッドと接続の使用

Adaptive Server 用のマルチスレッド OLE DB アプリケーションを開発できます。スレッドごとに個別の接続を使用することをおすすめします。ただし、複数のスレッド間でオープンな接続を共有できます。

SQL 文の実行

OLE DB には、SQL 文を実行するための関数がいくつか含まれます。

- **直接の実行**　Adaptive Server は SQL 文を解析したうえで、実行プランを準備し、SQL 文を実行します。解析および実行プランの準備を文の準備といいます。
- **バインドされたパラメータの実行**　実行時に文パラメータの値を設定するには、バインドされたパラメータを使用して SQL 文を構築し実行します。複数回実行する文のパフォーマンスを向上させるために、準備された文でバインドされたパラメータを使用することもできます。
- **準備された実行**　文の準備は、実行とは別に行われます。繰り返し実行する文の場合、これにより準備の繰り返しが回避され、その結果としてパフォーマンスが向上します。

直接の文の実行

ICommandText::Execute() 関数は、SQL 文を準備して実行します。この項のコード・サンプルは、パラメータを使用しない文を実行する方法を示します。オプションで、文にパラメータを含めることができます。

❖ **パラメータを使用しない文の実行**

1　セッションから Command オブジェクトを取得します。

```
ICommandText* pICommandText;
hr = pIDBCreateCommand->CreateCommand(
     NULL, IID_ICommandText,
     (IUnknown**)&pICommandText);
```

2　コマンドが実行する SQL 文を設定します。

```
hr = pICommandText->SetCommandText(
     DBGUID_DBSQL,
     L"DELETE FROM publishers where pub_id = '7777' ");
```

3 コマンドを実行します。cRowsAffected には、コマンドにより挿入、削除、または更新されたローの数が含まれます。次に示すように、pIRowset はコマンドにより作成された Rowset オブジェクトに割り付けられます。

```
DBROWCOUNT cRowsAffected;
IRowset* pIRowset;
hr = pICommandText->Execute(
     NULL, IID_IRowset, NULL,
     &cRowsAffected, (IUnknown**)&pIRowset);
```

バインドされたパラメータを持つ文の実行

この項のコード・サンプルは、バインドされたパラメータを使用して、実行時に文パラメータの値を設定する SQL 文を構築し実行する方法を示します。

❖ SQL 文の構築と実行

1 セッションから Command オブジェクトを作成します。

```
ICommandText* pICommandText;
hr = pIDBCreateCommand->CreateCommand(
     NULL, IID_ICommandText,
     (IUnknown**)&pICommandText);
```

2 実行する SQL 文を設定します。

```
hr = pICommandText->SetCommandText(
     DBGUID_DBSQL,
     L"DELETE FROM department WHERE dept_id = ?");
```

3 パラメータを記述する配列を作成します。

```
DB_UPARAMS paramOrdinal[1] = { 1 };
DBPARAMBINDINFO paramBindInfo[1] = {
     {
           L"DBTYPE_I4",
           NULL,
           sizeof(int),
           DBPARAMFLAGS_ISINPUT,
            0,
          0
     }
};
```

4 Command オブジェクトから ICommandWithParameters インタフェースを取得します。このコマンドのパラメータ情報を設定します。

```
ICommandWithParameters* pi;
hr = pICommandText->QueryInterface(
     IID_ICommandWithParameters, (void**)&pi);
hr = pi->SetParameterInfo(1, rgParamOrdinals,
rgParamBindInfo);
pi->Release();
```

5　次に、すべてのパラメータのデータを保持する構造を示します。この場合、次に示すように単一の int パラメータを使用します。

```
struct Parameters {
     int dept_id;
};
```

6　次の配列は、パラメータ構造内のフィールドを記述します。

```
static DBBINDING ExactBindingsParameters [1] = {
{
     1,           // iOrdinal
     offsetof (Parameters,dept_id), // obValue
     0,           // No length binding
     0,           // No Status binding
     NULL,  // No TypeInfo
     NULL,  // No Object
     NULL,    // No Extensions
     DBPART_VALUE,
     DBMEMOWNER_CLIENTOWNED,   // Ignored
     DBPARAMIO_INPUT,
     sizeof (int),
     0,
     DBTYPE_I4,
     0,              // No Precision
      0                  // No Scale
}
};
```

7　次のインタフェースは、Command オブジェクトからの IAccessor インタフェースです。

```
IAccessor* pIAccessor;
hr = pICommandText->QueryInterface(
     IID_IAccessor, (void**)&pIAccessor);
```

8　パラメータのアクセサを Command オブジェクト上に作成します。

```
DBBINDSTATUS status[1];
HACCESSOR hAccessor;
HRESULT hr = pIAccessor-
>CreateAccessor(DBACCESSOR_PARAMETERDATA,
     1, ExactBindingsParameters,
sizeof(ExactBindingsParameters),
     &hAccessor, status);
pIAccessor->Release();
```

9 パラメータの配列を作成します。配列内の各要素は、パラメータの完全なセットです。Execute メソッドは、次に示すように配列内の各パラメータ・セットについて SQL 文を 1 回実行します。

```
Parameters param = { 1 };
DBPARAMS params[1] = {
     {
          &param,
          1,
           hAccessor
     }

};
```

10 コマンドを実行します。

```
DBROWCOUNT cRowsAffected;
IRowset* pIRowset;
hr = pICommandText->Execute(
     NULL, IID_IRowset, params,
     &cRowsAffected, (IUnknown**)&pIRowset);
```

準備された文の実行

Adaptive Server OLE DB プロバイダは、準備された文を使用するための関数の完全なセットを提供します。これにより、繰り返し使用される文のパフォーマンスが向上します。次のコード・サンプルは、準備された文を使用する方法を示します。

注意 Adaptive Server での文のコンパイルと準備を有効にするには、DynamicPrepare=1 を設定します。

❖ 準備された文の使用

1 セッションから Command オブジェクトを取得します。

```
ICommandText* pICommandText;
hr = pIDBCreateCommand->CreateCommand(
     NULL, IID_ICommandText,
     (IUnknown**)&pICommandText);
```

2 実行する SQL 文を設定します。

```
hr = pICommandText->SetCommandText(
     DBGUID_DBSQL,
     L"DELETE FROM department WHERE dept_id = ?");
```

3　Command オブジェクトから ICommandPrepare インタフェースを取得します。次に示すように、Prepare を呼び出してコマンドを準備します。

```
ICommandPrepare* pICommandPrepare;
hr = pICommandText->QueryInterface(
     __uuidof(ICommandPrepare),
     (void**)&pICommandPrepare);
hr = pICommandPrepare->Prepare(cExpectedRuns);
pICommandPrepare->Release();
```

4　パラメータを記述する配列を作成します。

```
DB_UPARAMS paramOrdinal[1] = { 1 };
DBPARAMBINDINFO paramBindInfo[1] = {
     {
          L"DBTYPE_I4",
          NULL,
          sizeof(int),
          DBPARAMFLAGS_ISINPUT,
          0,
          0

};
```

5　Command オブジェクトから ICommandWithParameters インタフェースを取得し、パラメータ情報を設定します。

```
ICommandWithParameters* pi;
hr = pICommandText->QueryInterface(
     IID_ICommandWithParameters, (void**)&pi);
hr = pi->SetParameterInfo(1, rgParamOrdinals,
rgParamBindInfo);
pi->Release();
```

6　パラメータ・データを保持するための構造体を作成します。この構造体には、次に示すようにこのコマンドのすべてのパラメータが含まれます。

```
struct Parameters {
     int dept_id;
};
```

次に、コマンドに対する構造体について説明します。

```
static DBBINDING ExactBindingsParameters [1] = {
{
     1, // iOrdinal
     offsetof (Parameters,dept_id), // obValue
     0, // No length binding
     0, // No Status binding
     NULL, // No TypeInfo
     NULL, // No Object
     NULL, // No Extensions
     DBPART_VALUE,
     DBMEMOWNER_CLIENTOWNED, // Ignored
```

```
        DBPARAMIO_INPUT,
        sizeof (int),
        0,
        DBTYPE_I4,
        0, // No Precision
        0 // No Scale
    }
    };

    IAccessor* pIAccessor;
    hr = pICommandText->QueryInterface(IID_IAccessor,
    (void**)&pIAccessor);

    DBBINDSTATUS status[1];
    HACCESSOR hAccessor;
    HRESULT hr = pIAccessor->CreateAccessor(
        DBACCESSOR_PARAMETERDATA, 1,
        ExactBindingsParameters,
    sizeof(ExactBindingsParameters),
        &hAccessor, status);
    pIAccessor->Release();

    Parameters param = { 1 };
    DBPARAMS params[1] = {
        {
            &param,
            1,
            hAccessor
        }
    };

    DBROWCOUNT cRowsAffected;
    IRowset* pIRowset;
    hr = pICommandText->Execute(
        NULL, IID_IRowset, params,
        &cRowsAffected, (IUnknown**)&pIRowset);
```

7 IAccessor インタフェースを使用して、パラメータ構造体のアクセサを作成します。

```
    IAccessor* pIAccessor;
    hr = pICommandText->QueryInterface(IID_IAccessor,
    (void**)&pIAccessor);

    DBBINDSTATUS status[1];
    HACCESSOR hAccessor;
    HRESULT hr = pIAccessor->CreateAccessor(
        DBACCESSOR_PARAMETERDATA, 1,
        ExactBindingsParameters,
    sizeof(ExactBindingsParameters),
        &hAccessor, status);
    pIAccessor->Release();
```

次に、パラメータ・セットの配列を示します。

```
Parameters param = { 1 };
DBPARAMS params[1] = {
     {
          &param,
          1,
          hAccessor
     }
};
```

8　コマンドを実行します。

```
DBROWCOUNT cRowsAffected;
IRowset* pIRowset;
hr = pICommandText->Execute(
     NULL, IID_IRowset, params,
     &cRowsAffected, (IUnknown**)&pIRowset);
```

結果セットの使用

文を実行し、結果セットを操作する OLE DB 関数は、そのタスクを実行するためにカーソルを使用します。アプリケーションでは、結果セットを返す文を実行する場合に、カーソルを暗黙的にオープンします。

結果セット内を前方向にのみ移動し、結果セットを更新しないアプリケーションでは、カーソルの動作は比較的簡単になります。デフォルトでは、OLE DB アプリケーションはこの動作を要求します。OLE DB は、読み取り専用の前方向のみのカーソルを定義しています。また Adaptive Server OLE DB プロバイダは、この場合はパフォーマンスのために最適化されたカーソルを提供します。

ロー・セットで返されるロー数を制限するには、DBPROP_MAXROWS rowset プロパティを使用します。このプロパティのデフォルト値は 0 で、返される行数に制限がないことを示します。

注意　サーバ側・カーソルを有効にするには、UseCursor プロパティに 1 を設定します。

データの検索

次のコード例は、データを取得する方法を示します。

❖ **データの検索**

1　Command オブジェクトを作成します。

```
ICommandText* pICommandText;
hr = pIDBCreateCommand->CreateCommand(
     NULL, IID_ICommandText,
     (IUnknown**)&pICommandText);
```

2 SQL 文を設定します。

```
hr = pICommandText->SetCommandText(
     DBGUID_DBSQL,
     L"SELECT * FROM testReadStringData");
```

3 rowset データ構造体を作成し記述します。この構造体には、次に示すように、アクセスする各カラムのフィールドが含まれます。

```
IAccessor* pIAccessor;
hr = pICommandText->QueryInterface(IID_IAccessor,
(void**)&pIAccessor);

static DBBINDING ExactBindings [1] = {
{
     1, // iOrdinal
     offsetof (ExactlyTheSame,s), // obValue
     0, // No length binding
     0, // No Status binding
     NULL, // No TypeInfo
     NULL, // No Object
     NULL, // No Extensions
     DBPART_VALUE,
     DBMEMOWNER_CLIENTOWNED, // Ignored
     DBPARAMIO_NOTPARAM,
     sizeof(mystr), // number of bytes
     0,
     DBTYPE_WSTR | DBTYPE_BYREF,
     0, // No Precision
     0 // No Scale
}
};

DBBINDSTATUS status[1];
HACCESSOR hAccessor;
HRESULT hr = pIAccessor->CreateAccessor(
     DBACCESSOR_ROWDATA, 1, ExactBindings,
     sizeof(ExactlyTheSame), &hAccessor, status);
pIAccessor->Release();
```

4 rowset を実行します。

```
DBROWCOUNT cRowsAffected;
IRowset* pIRowset;
hr = pICommandText->Execute(
NULL, IID_IRowset, params,
&cRowsAffected, (IUnknown**)&pIRowset);
```

5　次のコードを使用して、一度に 1 つずつローを取得します。

```
DBCOUNTITEM cRowsReturned;
HROW hRow[1];
HROW* pRow = hRow;
hr = pIRowset->GetNextRows(NULL, 0, 1, &cRowsReturned,
&pRow);
```

6　IMalloc を使用して、GetData により割り付けられたメモリを解放します。

```
CComPtr<IMalloc> pIMalloc = NULL;
hr = CoGetMalloc( MEMCTX_TASK, &pIMalloc );

while (hr == S_OK)
{
```

7　指定したローのデータを取得します。次に例を示します。

```
ExactlyTheSame pData[1] = { {NULL} };
hr = pIRowset->GetData(hRow[0], hAccessor, pData);
wchar_t* value = pData[0].s;
```

8　割り付けられたメモリを解放します。

```
// client owned memory must be freed by the client
pIMalloc->Free(pData[0].s);
pData[0].s = NULL;
```

9　ローを解放します。

```
hr = pIRowset->ReleaseRows(1, pRow, NULL, NULL,
NULL);
```

10　次のローを取得します。

```
hr = pIRowset->GetNextRows(NULL, 0, 1,
     &cRowsReturned, &pRow);
}

pIRowset->Release();
pICommandText->Release();
```

データベースからローを取得するには、ICommandText::Execute を使用して SELECT 文を実行します。これにより、文のカーソルがオープンされます。次に、IRowset::GetNextRows を使用してカーソルによりローをフェッチします。アプリケーションが rowset を解放することにより文を解放すると、カーソルがクローズされます。

スクロール可能なカーソルの使用

IRowset::GetNextRows はスクロール可能なカーソルをサポートし、rowset 内で前後に移動できます。GetNextRows の lRowsOffset または cRows パラメータに負の値を指定すると、rowset は結果セットを後方に移動します。

Adaptive Server OLE DB プロバイダは、Static Insensitive スクロール可能カーソルをサポートしています。このプロバイダは、IRowset::GetNextRows() メソッドを実装しています。これは、MSDN ライブラリに含まれる『Microsoft Open Database Connectivity Software Development Kit Programmer's Reference Volume 2』で定義された標準メソッドです。詳細については、Microsoft Web Site (http://msdn.microsoft.com/) を参照してください。

OLE DB プロバイダは、次のスクロール・タイプをサポートします。

- Next － 次のローが返される。
- Prior － 前のローが返される。
- Relative *n* rows － 現在の rowset から *n* 番目の rowset が返される。

UseCursor 接続プロパティの設定

クライアント側またはサーバ側のスクロール可能なカーソルが使用されているかどうかを特定するには、UseCursor プロパティを設定する必要があります。

- UseCursor 接続プロパティを 1 に設定すると、Adaptive Server バージョンが 15.0 以降の場合、サーバ側のスクロール可能なカーソルが使用されます。それ以前のバージョンの Adaptive Server の場合、サーバ側のスクロール可能なカーソルは使用できません。
- UseCursor 接続プロパティを 0 に設定すると、クライアント側のスクロール可能なカーソル (キャッシュされた結果セット) が Adaptive Server のバージョンにかかわらず使用されます。

警告！ クライアント側のスクロール可能なカーソルを使用する場合、リソースを多量に消費します。

スクロール可能なカーソル属性の設定

スクロール可能カーソルを使用するには、次の属性を設定する必要があります。

- DBPROP_CANSCROLLBACKWARDS – VARIANT_TRUE に設定すると、GetNextRows の lRowsOffset パラメータに負の値を指定できます。
- DBPROP_CANFETCHBACKWARDS – VARIANT_TRUE に設定すると、GetNextRows の cRows パラメータに負の値を指定できます。

スクロール可能カーソルの実行

❖ スクロール可能カーソルの実行

1　rowset でスクロール可能なカーソル・プロパティを設定します。

```
DBPROP RowsetProperties[2];
for(int i = 0; i < 2; i++)
   VariantInit(&RowsetProperties[i].vValue);

RowsetProperties[0].dwPropertyID = DBPROP_CANFETCHBACKWARDS;
RowsetProperties[0].vValue.vt      = VT_BOOL;
RowsetProperties[0].vValue.boolVal = VARIANT_TRUE;
RowsetProperties[0].dwOptions      = DBPROPOPTIONS_REQUIRED;
RowsetProperties[0].colid          = DB_NULLID;
RowsetProperties[1].dwPropertyID   = DBPROP_CANSCROLLBACKWARDS;
RowsetProperties[1].vValue.vt      = VT_BOOL;
RowsetProperties[1].vValue.boolVal = VARIANT_TRUE;
RowsetProperties[1].dwOptions      = DBPROPOPTIONS_REQUIRED;
RowsetProperties[1].colid          = DB_NULLID;

DBPROPSET rgRowsetPropSet[1];
rgRowsetPropSet[0].guidPropertySet = DBPROPSET_ROWSET;
rgRowsetPropSet[0].cProperties     = 2;
rgRowsetPropSet[0].rgProperties    = RowsetProperties;
```

2　rowset をオープンします。

```
IRowset* pIRowset = ds.OpenRowset("book", 1, rgRowsetPropSet);
```

3　ローを前方向にフェッチします。

```
DBCOUNTITEM cRowsReturned;
HROW hRow[3];
HROW* pRows = hRow;
hr = pIRowset->GetNextRows(NULL, 0, 3, &cRowsReturned, &pRows);
```

4　ローを解放します。

```
hr = pIRowset->ReleaseRows(cRowsReturned, pRows, NULL, NULL, NULL);
```

5　ローを後方向にフェッチします。

```
DBCOUNTITEM cRowsReturned;
HROW hRow[3];
HROW* pRows = hRow;
hr = pIRowset->GetNextRows(NULL, 0, -3, &cRowsReturned, &pRows);
```

6　ローを解放します。

```
hr = pIRowset->ReleaseRows(cRowsReturned, pRows, NULL, NULL, NULL);
```

7　rowset を解放します。

```
pIRowset->Release()
```

結果の参照

結果および結果セットの解釈を確認するには、スクロール可能カーソルを実行後、Microsoft MSDN library (http://msdn.microsoft.com/) を参照してください。

スクロール可能な静的で非依存なカーソル・プログラムの例

スクロール可能な静的で非依存なカーソル・プログラムの例については、「スクロール可能カーソルの実行」(21 ページ) を参照してください。

ストアド・プロシージャの呼び出し

この項では、ストアド・プロシージャを呼び出し、OLE DB アプリケーションからの結果を処理する方法について説明します。

ストアド・プロシージャとトリガの詳細については、『ASE リファレンス・マニュアル』を参照してください。

❖ ストアド・プロシージャの呼び出しとその結果の処理

1 コマンドを作成します。

```
ICommandText* pICommandText;
hr = pIDBCreateCommand->CreateCommand(
     NULL, IID_ICommandText,
     (IUnknown**)&pICommandText);
```

2 コマンドのテキストを設定します。

```
hr = pICommandText->SetCommandText(
     DBGUID_DBSQL,
     L"{ call sp_foo(?) }");
```

3 パラメータを定義します。

```
DB_UPARAMS paramOrdinal[1] = { 1 };
DBPARAMBINDINFO paramBindInfo[1] = {
     {
          L"DBTYPE_I4",
          NULL,
          sizeof(int),
          DBPARAMFLAGS_ISINPUT,
          0,
          0
     }
};
```

4　コマンドのパラメータ情報を設定します。

```
ICommandWithParameters* pi;
hr = pICommandText->QueryInterface(
     IID_ICommandWithParameters, (void**)&pi);
hr = pi->SetParameterInfo(1, rgParamOrdinals,
rgParamBindInfo);
pi->Release();
```

5　パラメータのデータ構造体を定義します。

```
struct Parameters {
int dept_id;
};

static DBBINDING ExactBindingsParameters [1] = {
     {
          1, // iOrdinal
          offsetof (Parameters,dept_id), // obValue
          0, // No length binding
          0, // No Status binding
          NULL, // No TypeInfo
          NULL, // No Object
          NULL, // No Extensions
          DBPART_VALUE,
          DBMEMOWNER_CLIENTOWNED, // Ignored
          DBPARAMIO_INPUT,
          sizeof (int),
          0,
          DBTYPE_I4,
          0, // No Precision
          0 // No Scale
     }
};
```

6　パラメータのアクセサを作成します。

```
IAccessor* pIAccessor;
hr = pICommandText->QueryInterface(IID_IAccessor,
(void**)&pIAccessor);
DBBINDSTATUS status[1];
HACCESSOR hAccessor;
HRESULT hr = pIAccessor->CreateAccessor(
     DBACCESSOR_PARAMETERDATA, 1,
     ExactBindingsParameters,
sizeof(ExactBindingsParameters),
     &hAccessor, status);
pIAccessor->Release();
```

7　パラメータ・データを定義します。

```
Parameters param = { 1 };
DBPARAMS params[1] = {
     {
          &param,
          1,
          hAccessor
     }
};

DBROWCOUNT cRowsAffected;
IRowset* pIRowset;
hr = pICommandText->Execute(
     NULL, IID_IRowset, params,
     &cRowsAffected, (IUnknown**)&pIRowset);
```

エラーの処理

メソッドから失敗を返すことにより、エラーがレポートされます。すべてのメソッドが HRESULT を返します。失敗が発生したかどうかを判別するには、FAILED(hr) を呼び出します。エラーに関する情報を取得するには、GetErrorInfo を呼び出します。

例　次のコード例では、FAILED(hr) と GetErrorInfo を使用します。

```
if (FAILED(hr))
{
     IErrorInfo* pIErrorInfo;
     GetErrorInfo(0, &pIErrorInfo);
     BSTR desc;
     pIErrorInfo->GetDescription(&desc);
     // use the desc
     SysFreeString(desc);
     pIErrorInfo->Release();
}
```

データ型のマッピング

次の表は、Adaptive Server OLE DB プロバイダのデータ型のマッピングを示します。

表 *1-2: Adaptive Server のデータ型と OLE DB のデータ型*

ASE のデータ型	OLE DB のデータ型	C++ 言語のデータ型
bigdatetime	DBTYPE_DBTIMESTAMP	TIMESTAMP_STRUCT
bigint	DBTYPE_I8	long long
bigtime	DBTYPE_DBTIME	TIME_STRUCT
binary	DBTYPE_BYTES	unsigned char[]
bit	DBTYPE_BOOL	BOOL
char	DBTYPE_STR、 DBTYPE_BSTR	char[]、BSTR
date	DBTYPE_DBDATE	DATE_STRUCT
datetime	DBTYPE_DBTIMESTAMP	TIMESTAMP_STRUCT
decimal	DBTYPE_DECIMAL	SQL_NUMERIC
double	DBTYPE_R8	double
float (<16)	DBTYPE_R4	float
float (>=16)	DBTYPE_R8	double
image	DBTYPE_IUNKNOWN、 DBTYPE_BYTES	IUnknown、unsigned char[] **注意** IUnknown インタフェースを通じてストリームを使用することをおすすめします。また、unsigned char[] としてバインドすることもできます。
int[eger]	DBTYPE_I4	long
money	DBTYPE_CY	long long
nchar	DBTYPE_STR、 DBTYPE_BSTR	char[]、BSTR
numeric	DBTYPE_NUMERIC	SQL_NUMERIC
nvarchar	DBTYPE_STR、 DBTYPE_BSTR	char[]、BSTR
real	DBTYPE_R4	float
smalldatetime	DBTYPE_DBTIMESTAMP	TIMESTAMP_STRUCT
smallint	DBTYPE_I2	short
smallmoney	DBTYPE_CY	long long
text	DBTYPE_IUNKNOWN、 DBTYPE_STR	IUnknown、char[] **注意** IUnknown インタフェースを通じてストリームを使用することをおすすめします。また、char[] としてバインドすることもできます。
time	DBTYPE_DBTIME	TIME_STRUCT
timestamp	DBTYPE_BYTES	unsigned char[]

ASE のデータ型	OLE DB のデータ型	C++ 言語のデータ型
tinyint	DBTYPE_UI1	unsigned char
unichar	DBTYPE_WSTR、 DBTYPE_BSTR	wchar_t[]、BSTR
unitext	DBTYPE_IUNKNOWN、 DBTYPE_WSTR	IUnknown、wchar_t[] **注意**　IUnknown を使用することをおすすめします。
univarchar	DBTYPE_WSTR、 DBTYPE_BSTR	wchar_t[]、BSTR
unsignedbigint	DBTYPE_UI8	unsigned long long
unsignedint	DBTYPE_UI4	unsigned long
unsignedsmallint	DBTYPE_UI2	unsigned short
varbinary	DBTYPE_BYTES	unsigned char[]
varchar	DBTYPE_STR、 DBTYPE_BSTR	char[]、BSTR

計算カラムの使用

Adaptive Server OLE DB プロバイダは計算カラムをサポートします。計算カラムを使用すると、“Salary + Commission” に対する “Pay” のように、式に対する簡単な表現を作成でき、インデックスを作成できるデータ型の場合は、カラムをインデックス可能にすることができます。**計算カラム**は、同じローの通常のカラム、関数、算術演算子、パス名、それらのメタデータ情報などの式によって定義されます。

データベース・オブジェクトの長い識別子の使用

Adaptive Server OLE DB プロバイダは、最長 255 バイトのオブジェクト名または識別子をサポートします。Adaptive Server の長い識別子の詳細については、『Adaptive Server Enterprise 15.0 新機能ガイド』を参照してください。

警告！ C++ プログラムまたはクライアント・アプリケーションで**長い識別子**を使用する場合は、データのトランケーションを防ぐため、十分な長さのバッファを割り付ける必要があります。

Adaptive Server OLE DB プロバイダのサンプル

Adaptive Server OLE DB プロバイダを使用した Visual Basic のサンプルは、*%SYBASE%¥DataAccess¥OLEDB¥samples* ディレクトリにあります。

OLE DB DSN のマイグレーション

OLE DB DSN のマイグレーション・ツールを使用すると、データ・ソース定義 (DSN) を OLE DB ドライバ･キットから Sybase 製の Adaptive Server OLE DB プロバイダにマイグレートできます。マイグレートされた DSN は、OLE DB ドライバ・キットではなく、Sybase 製の Adaptive Server OLE DB プロバイダを使用します。

Sybase 製 Adaptive Server OLE DB プロバイダへのマイグレート

Sybase 製の Adaptive Server OLE DB プロバイダを使用するように OLE DB アプリケーションをマイグレートするには、OLE DB クライアント・アプリケーションが使用する接続文字列を変更する必要があります。Adaptive Server OLE DB プロバイダの短い名前は、“ASEOLEDB” です。

注意 Adaptive Server OLE DB プロバイダの接続文字列の構文は、OLE DB ドライバ・キットの構文とは異なります。OLE DB ドライバ・キットの構文もサポートされていますが、可能な場合は Adaptive Server OLE DB プロバイダの接続文字列の構文にマイグレートすることをおすすめします。

Sybase ドライバへのデータ・ソース名のマイグレート

OLE DB ドライバ・キットから Sybase が作成したドライバにデータ・ソース名 (DSN) をマイグレートするには、次の 2 つの方法があります。

- Sybase Adaptive Server データ・ソース・アドミニストレータの使用
- DSN マイグレーション・ツールの使用

Sybase Adaptive Server データ・ソース・アドミニストレータの使用

Sybase Adaptive Server データ・ソース・アドミニストレータは、既存の OLE DB ドライバ・キットのデータ・ソースのマイグレートと、Adaptive Server OLE DB プロバイダ用の新しいデータ・ソースの作成を可能にします。

❖ データ・ソース・アドミニストレータによるデータ・ソースのマイグレート

1　“Sybase Data Source Administrator” というタイトルのメイン・ウィンドウで、データ・ソースを選択します。

2　[マイグレート] をクリックします。

Sybase データ・ソース・アドミニストレータを使用して、OLE DB データ・ソースを追加、削除、設定、またはテストできます。

DSN マイグレーション・ツールの使用

DSN マイグレーション・ツールを使用して、OLE DB ドライバ・キットから Sybase 製の OLE DB ドライバにデータ・ソースをマイグレートできます。

dsnmigrate ツールでは、どの DSN をマイグレートするかを制御するスイッチを使用します。コマンド・ラインで次のように入力します。

```
dsnmigrate.exe [/?|/h|/help][/oledb]
[/l|/ul|/sl][/a|/ua|/sa] [[/dsn|/udsn|/sdsn]=dsn]
[/suffix=suffix]
```

変換されたすべての OLE DB DSN に対して、新しい Sybase DSN では変換前と同じ名前が使用されます。

変換スイッチ

表 1-3 に、変換で使用されるスイッチを示し、各スイッチについて説明します。

表 *1-3:* 変換スイッチ

スイッチ	結果の説明
/?、/h、/help	使用可能な dsnmigrate スイッチを表示します。これらのスイッチは、コマンド・ライン引数を指定せずに dsnmigrate を呼び出した場合にも表示されます。
/oledb	dsnmigrate を OLEDB モードにします。デフォルトでは、ODBC DSN がマイグレートされます。
/l	OLE DB ドライバ・キットのすべてのユーザ DSN とシステム DSN を一覧表示します。
/ul	OLE DB ドライバ・キットのすべてのユーザ DSN を一覧表示します。
/sl	OLE DB ドライバ・キットのすべてのシステム DSN を一覧表示します。
/a	OLE DB ドライバ・キットのすべてのユーザ DSN とシステム DSN を変換します。
/ua	OLE DB ドライバ・キットのすべてのユーザ DSN を変換します。
/sa	OLE DB ドライバ・キットのすべてのシステム DSN を変換します。

スイッチ	結果の説明
/dsn	OLE DB ドライバ・キットの特定のユーザ DSN またはシステム DSN を変換します。
/udsn	OLE DB ドライバ・キットの特定のユーザ DSN を変換します。
/sdsn	OLE DB ドライバ・キットの特定のシステム DSN を変換します。
dsn	変換される DSN の名前。
/suffix	DSN に名前を付ける方法を変更するオプション・スイッチ。このスイッチを使用すると、元の DSN が保持され、新しい DSN に “<dsn>-<suffix>” という名前が付けられます。
suffix	新しい DSN に名前を付けるために使用されるサフィックス。

第 2 章

データベースへの接続

この章では、クライアント・アプリケーションが Adaptive Server OLE DB プロバイダを使用して Adaptive Server Enterprise に接続する方法を説明します。

接続の概要

Adaptive Server を使用するクライアント・アプリケーションは、まず、サーバへの接続を確立する必要があります。接続は、クライアント・アプリケーションのすべてのアクティビティを実行するチャネルとなります。たとえば、ユーザがデータベース上で実行できる操作はユーザ ID によって決定されますが、このユーザ ID は接続確立要求の一部として送信され、データベース・サーバに渡されます。

Adaptive Server OLE DB プロバイダは、クライアント・アプリケーションからの呼び出しに含まれている接続情報を (初期化ファイル内のディスクに格納されている情報と併用して)、必要なデータベースを実行している Adaptive Server サーバを見つけて接続します。

OLE DB メタデータ・ストアド・プロシージャのインストール

OLE DB プロバイダを使用して接続するどの Adaptive Server にも OLE DB メタデータ・ストアド・プロシージャをインストールする必要があります。

❖ **Adaptive Server への OLE DB ストアド・プロシージャのインストール**

1 OLE DB インストール・ディレクトリ内の *sp* ディレクトリに移動します。

- OLE DB 32 ビット版の場合：*%SYBASE%¥DataAccess¥oledb¥sp*
- OLE DB 64 ビット版の場合：*%SYBASE%¥DataAccess64¥oledb¥sp*

2 *install_oledb_sprocs* スクリプトを次のように実行します。

```
install_oledb_sprocs ServerName username [password]
```

各パラメータの意味は、次のとおりです。

- ***ServerName*** は、Adaptive Server の名前。
- ***username*** は、サーバに接続するユーザの名前。
- ***[password]*** は、ユーザ名のパスワード。値が null の場合は、パラメータを空にする。

接続パラメータの構造

アプリケーションからデータベースに接続する場合は、サーバ名、データベース名、ユーザ ID などの一連の接続パラメータを使用して接続を定義します。各接続パラメータは、キーワードと値の組み合わせとして、parameter=value という形式で指定されます。たとえば、ユーザ ID の接続パラメータは、次のように指定します。

```
User ID=sa
```

接続文字列として渡される接続パラメータ

接続パラメータを組み合わせて接続文字列を作ります。接続文字列では、各接続パラメータを、次に示すようにセミコロンで区切ります。

```
parameter1=value1;parameter2=value2;...
```

通常、アプリケーションによって構築され、ドライバに渡される接続文字列は、ユーザが情報を入力する方法と直接対応していません。代わりに、ユーザがダイアログに入力するか、アプリケーションが初期化ファイルから接続情報を読み込むことができます。

次に、接続文字列は Adaptive Server OLE DB プロバイダに渡されます。

OLE DB データ・ソースへの接続パラメータの保存

データベースに接続するとき、OLE DB アプリケーションは 1 つのファイルに格納された接続パラメータのセットである OLE DB データ・ソースを使用します。

OLE DB データ・ソースを設定するには、Sybase Data Source Administrator を使用します。

データ・ソースを使用した接続

OLE DB アプリケーションは通常、接続先の各データベースのクライアント・コンピュータのデータ・ソースを使用します。Adaptive Server Enterprise の接続パラメータのセットを OLE DB データ・ソースとして格納できます。データ・ソースがある場合、DataSource 接続パラメータを使用して、接続文字列には目的のデータ・ソースを指定するだけで済みます。

```
Data Source=my data source;
```

接続パラメータの使用

次は、Adaptive Server OLE DB プロバイダに提供できる接続パラメータのリストです。

表 *2-1:* 接続パラメータ

プロパティ名	説明	必須	デフォルト値
AnsiNull	厳格に準拠し、“= NULL” を使用不可とする。代わりに “IsNull” を使用する。	いいえ	1
ApplicationName	クライアント・アプリケーションを識別するために Adaptive Server が使用する名前。	いいえ	空
BufferCacheSize	入出力バッファをプール内に保持する。結果が大きくなる場合、この値を大きくするとパフォーマンスが向上する。	いいえ	20
CharSet	Adaptive Server との通信に使用する文字セットを指定する。デフォルトでは、Adaptive Server OLE DB ドライバは Adaptive Server と同じデフォルトの文字セットをネゴシエートする。	いいえ	空
ClientCharset	クライアント文字セットを指定する。ClientCharset が CharSet と異なる場合、Adaptive Server OLE DB プロバイダは、クライアントの文字セットを CharSet で指定された文字セットに変換する。	いいえ	オペレーティング・システムが現在使用する文字セット。
ClientHostName	ログイン・レコードでサーバに渡されたクライアント・ホストの名前。	いいえ	空
ClientHostProc	ログイン・レコードでサーバに渡されたこのホスト・マシン上のクライアント・プロセスの ID。	いいえ	空
CodePageType	使用する文字コードの種類を指定する。有効な値は、ANSI と OEM。	いいえ	ANSI

プロパティ名	説明	必須	デフォルト値
CRC	ストアド・プロシージャで複数の update 文が実行されたとき、ドライバはデフォルトで、更新されたレコード総数を返す。このカウントには、update や insert に設定されたトリガで実行されるすべての更新操作も含まれる。 ドライバが最後の更新カウントだけを返すようにする場合は、このプロパティを 0 に設定する。	いいえ	1
DataIntegrity	Kerberos のデータの整合性を有効にする。	いいえ	0 (無効)
DataSource	*Server:Port* のフォーマットで接続するデータ・ソース。	いいえ (サーバとポートが指定されている場合)。	空
DSPassword	LDAP サーバで匿名アクセスが許可されない場合に、LDAP サーバでの認証に使用するパスワード。パスワードは DSURL でも指定可能。	いいえ	空
DSPrincipal	LDAP サーバで匿名アクセスが許可されない場合に、LDAP サーバでの認証に使用するユーザ名。プリンシパルは DSURL でも指定可能。	いいえ	空
DSURL	LDAP サーバへの URL。	いいえ	空
DynamicPrepare	1 に設定した場合、ドライバはコンパイルまたは準備のために SQLPrepare 呼び出しを Adaptive Server に送信する。これにより、同じクエリを繰り返し使用する場合に、パフォーマンスを向上させることができる。	いいえ	0
EnableServerPacketSize	最適なパケット・サイズを選択するために、Adaptive Server 15.0 以降を許可する。	いいえ	1
EncryptedPassword	パスワードが暗号化フォーマットで転送されるかどうかを指定する。 • 0 − プレーン・テキスト形式のパスワードを使用。 • 1 − 暗号化されたパスワードを使用。サポートされていない場合、エラー・メッセージを返す。 • 2 − 暗号化されたパスワードを使用。サポートされていない場合、プレーン・テキスト形式のパスワードを使用する。 **注意** パスワードの暗号化が有効になっていて、サーバが非対称暗号化をサポートしている場合、非対称暗号化が対称暗号化の代わりに使用される。	いいえ	0
Encryption	指定されている暗号化方式。指定できる値：ssl。	いいえ	空

プロパティ名	説明	必須	デフォルト値
HASession	高可用性を有効にするかどうかを指定する。0 は高可用性を無効にし、1 は高可用性を有効にする。	いいえ	0
Initial Catalog、Database	接続するデータベース。	いいえ	空
Language	Adaptive Server が返すエラー・メッセージの言語。	いいえ	空 - デフォルトでは英語を使用。
LoginTimeOut	アプリケーションへ戻る前に、ログインの試行を待機する秒数。0 に設定するとタイムアウトが無効になり、接続の試行を永久的に待機する。	いいえ	15
MutualAuthentication	Kerberos 相互認証を有効にする。	いいえ	0 (無効)
oldpassword	現在のパスワード。oldpassword に null や空の文字列以外の値が含まれている場合、現在のパスワードは PWD に含まれる値に変更される。	いいえ	空
PacketSize	Adaptive Server とクライアント間で送受信されるネットワーク・パケット 1 つあたりのバイト数。	いいえ	ドライバが Adaptive Server 15.0 以降に接続したときに判別されたサーバ。より古い Adaptive Server では、デフォルトは 512 になる。
Port	Adaptive Server サーバのポート番号。	いいえ (データ・ソースが指定されている場合)	空
PWD、Password	パスワードの値が含まれる。通常のログインを実行すると、oldpassword が設定されず PWD に現在の値が含まれる。パスワードを変更すると、oldpassword に現在の値が設定され、PWD には新しいパスワードの値が含まれる。	いいえ (ユーザ名にパスワードが不要な場合)。	空
QuotedIdentifier	Adaptive Server で二重引用符で囲まれた文字列を識別子として処理するかどうかを指定する。0 は引用符付き識別子を無効にし、1 は引用符付き識別子を有効にする。	いいえ	0
ReplayDetection	Kerberos リプレイ検出を有効にする。	いいえ	0
RestrictMaximum PacketSize	EnableServerPacketSize に 1 を設定した場合にメモリに制約があるときは、このプロパティに 512 の倍数 (最大 65536) の int 値を設定する。	いいえ	0
SecondaryPort	アクティブ／アクティブ設定またはアクティブ／パッシブ設定でフェールオーバ・サーバとして機能する Adaptive Server のポート番号。	はい (HASession が 1 に設定されている場合)	空

プロパティ名	説明	必須	デフォルト値
SecondaryServer	アクティブ／アクティブ設定またはアクティブ／パッシブ設定でフェールオーバ・サーバとして機能する Adaptive Server の名前または IP アドレス。	はい (HASession が 1 に設定されている場合)	空
Server	Adaptive Server の名前または IP アドレス。	いいえ (データ・ソースが指定されている場合)	空
ServerInitiated Transactions	ServerInitiatedTransactions に 1 を設定した場合、Adaptive Server は必要に応じてトランザクションの管理を開始する。ドライバは、接続に対して set chained on コマンドを発行する。接続で「連鎖」トランザクション・モードを使用しないことを要求する場合は、このプロパティに 0 を設定する。	いいえ	1
TextSize	ネットワークで送信されるバイナリまたはテキスト・データの最大サイズ。	いいえ	空。Adaptive Server のデフォルトは 32 KB。
TrustedFile	Encryption を ssl に設定した場合、このプロパティに信頼されたファイルへのパスを設定する。	いいえ	空
UseCursor	ドライバでカーソルを使用するかどうかを指定する。0 はカーソルを使用しない。1 はカーソルを使用する。	いいえ	0
User ID、UserID、UID	Adaptive Server サーバへの接続に必要なユーザ ID。大文字と小文字を区別する。	はい	なし

ADO からの接続

Microsoft ADO (ActiveX Data Object) は、オブジェクト指向のプログラミング・インタフェースです。ADO では、データ・ソースとのユニーク・セッションを Connection オブジェクトで表します。接続を開始するには、Connection オブジェクトの次の機能を使用します。

- プロバイダの名前を表す Provider プロパティ。プロバイダ名を指定しない場合は、MSDASQL プロバイダが使用されます。
- Adaptive Server 接続文字列を表す *ConnectionString* プロパティ。OLE DB データ・ソース名を指定することも、他の接続文字列と同様に、UserID、Password、DatabaseName などのパラメータを明示的に指定することもできます。
- 接続オブジェクトを使用して接続を開始する Open メソッド。

例

次の Visual Basic コードは、接続オブジェクトを使用して、Adaptive Server への OLE DB 接続を開始します。

```
' Declare the connection object
Dim myConn as New ADODB.Connection
myConn.Provider = "ASEOLEDB"
myConn.ConnectionString ="Data Source=MANGO:5000; User ID=sa"
myConn.Open
```

第 3 章

サポートされている Adaptive Server の機能

この章では、Adaptive Server OLE DB プロバイダで使用できる Adaptive Server の機能について説明します。

マイクロ秒の精度の time データ

Adaptive Server OLE DB プロバイダは、SQL データ型の bigdatetime と bigtime をサポートすることで、マイクロ秒レベルの精度の time データを提供します。

bigdatetime と bigtime は同様に機能し、SQL データ型の datetime および time とデータ・マッピングが同じです。

- bigdatetime は、Adaptive Server のデータ型 bigdatetime に対応し、0000 年 1 月 1 日の 0:00:00.000000 から経過したマイクロ秒数を示します。有効な bigdatetime 値の範囲は、0001 年 1 月 1 日の 0:00:00.000000 から 9999 年 12 月 31 日の 23:59:59.999999 までです。
- bigtime は、Adaptive Server のデータ型 bigtime に対応し、当日の午前 0 時ちょうどから経過したマイクロ秒数を示します。有効な bigtime 値の範囲は、00:00:00.000000 から 23:59:59.999999 までです。

使用法

- Adaptive Server 15.5 への接続時に、Adaptive Server OLE DB プロバイダは bigdatetime および bigtime データ型を使用してデータを転送します。受信した Adaptive Server カラムが datetime および time として定義されている場合でも同様です。

  これは、Adaptive Server が、Adaptive Server カラムに合わせるために、Adaptive Server OLE DB プロバイダから取得した値を暗黙的にトランケートする可能性があることを意味します。たとえば、bigtime の値 23:59:59.999999 は、time データ型の Adaptive Server カラムに 23:59:59.996 として保存されます。
- Adaptive Server 15.0.x 以前のバージョンへの接続時には、Adaptive Server OLE DB プロバイダは datetime および time データ型を使用してデータを転送します。
- 以前のバージョンの Adaptive Server OLE DB プロバイダは、引き続き datetime データ型と time データ型を使用します。

サポートされている Adaptive Server クラスタ・エディションの機能

この項では、クラスタ・エディション環境をサポートする Adaptive Server OLE DB プロバイダの機能について説明します。クラスタ・エディション環境では、複数の Adaptive Server が共有ディスクのセットと高速プライベート相互接続に接続します。この場合、複数の物理ホストと論理ホストを使用して、Adaptive Server を拡張できます。

クラスタ・エディションの詳細については、『Cluster ユーザーズ・ガイド』を参照してください。

ログインのリダイレクト

クラスタ・エディション環境では一般に、常にサーバ間で処理負荷の不均衡が発生しています。ビジー状態のサーバに対してクライアント・アプリケーションが接続を試みた場合、ログインのリダイレクト機能によって、サーバの負荷バランスが調整されます。具体的には、クラスタ内の負荷が少ない別サーバに対して、クライアント接続がリダイレクトされます。ログインのリダイレクトが発生するのはログイン・シーケンス中であり、リダイレクトが発生したことは、クライアント・アプリケーションには通知されません。ログインのリダイレクト機能をサポートしているサーバに対してクライアント・アプリケーションが接続した時点で、この機能は自動的に有効になります。

注意　クライアントをリダイレクトするように設定されているサーバに対してクライアント・アプリケーションが接続すると、ログインに時間がかかる場合があります。これは、クライアント接続が別サーバにリダイレクトされるたびに、ログイン・プロセスが再開されるからです。

接続マイグレーション

接続マイグレーションを使用すると、クラスタ・エディション環境内のサーバは動的に負荷を分散できます。さらに、既存のクライアント接続とそのコンテキストをクラスタ内の別サーバにシームレスにマイグレートできます。この機能によって、クラスタ・エディション環境では、最適なリソース配分と処理時間の短縮が実現します。サーバ間のマイグレーションはシームレスに行われるので、接続マイグレーション機能は、真に可用性が高い (HA: High Availability)「ダウン時間ゼロ」の環境を構築する場合にも役立ちます。接続マイグレーション機能をサポートしているサーバに対してクライアント・アプリケーションが接続した時点で、この機能は自動的に有効になります。

注意　接続マイグレーション中には、コマンドの実行に時間がかかる場合があります。状況に応じて、コマンドのタイムアウト値を増やすことをおすすめします。

接続フェールオーバの強化

既存の接続フェールオーバ機能を使用すると、停電やソケットの障害など、予想外の原因でプライマリ・サーバが使用不可になった場合に、クライアント・アプリケーションは接続先を別の Adaptive Server に切り替えることができます。この機能が強化され、クライアント・アプリケーションは動的なフェールオーバ・アドレスを使用して、複数のサーバに対して何度もフェールオーバできるようになりました。

高可用性に対応したシステムでは、フェールオーバ・ターゲットの候補をクライアント・アプリケーションにあらかじめ設定しておく必要はありません。Adaptive Server は、クラスタ・メンバシップ、論理クラスタの使用状況、負荷分散などに基づいて、最適なフェールオーバ・リストを常にクライアントに提供します。クライアントは、フェールオーバ時にフェールオーバ・リストの順序付けを参照して、再接続を試みます。ドライバがサーバに正常に接続した場合は、返されたリストに基づいて、ホスト値のリストが内部的に更新されます。それ以外の場合は、接続失敗例外が発生します。

クラスタ・エディションの接続フェールオーバの有効化

OLEDB ユーザ・インタフェースの使用

Adaptive Server OLE DB プロバイダで拡張接続フェールオーバを有効にする方法の 1 つは、そのユーザ・インタフェースを使用するものです。

❖ **ユーザ・インタフェースによる拡張フェールオーバの有効化**

1. Sybase Datasource Administrator の [データ・ソースの設定] ダイアログ・ボックスを開きます。
2. [接続] タブに移動します。
3. [高可用性の有効化] を選択します。
4. オプションで、[代替サーバ] フィールドに次の形式で代替サーバとポートを入力します。

```
server1:port1,server2:port2,...,serverN:portN;
```

接続を確立するとき、Adaptive Server OLE DB プロバイダは最初に、[データ・ソースの設定] ダイアログ・ボックスの [一般] タブで定義されているプライマリ・ホストとポートに接続を試みます。Adaptive Server OLE DB プロバイダが接続を確立できない場合、OLE DB は [Adaptive Server] フィールドのホストとポートのリストで次に指定されているホストとポートを試みます。

OLEDB 接続文字列の使用

また、OLE DB 接続文字列を使用して HASession 接続文字列プロパティを 1 に設定すると、拡張フェールオーバを有効にできます。次に例を示します。

```
Provider=ASEOLEDB;User ID=sa;Password=;
InitialCatalog=sdc;Data Source=server1:port1;
ProviderString='HASession=1;AlternateServers=
server2:port2,...,serverN:portN';
```

この例では、Data Source はプライマリ・サーバとポートを定義します。Adaptive Server OLE DB プロバイダは、最初にプライマリ・サーバとの接続を確立しようとします。接続に失敗した場合、代替サーバが定義されていれば、OLE DB は [代替サーバ] フィールドに列挙されているサーバへの接続を順番に試みます。接続に成功するか、リストの末尾に達するまで、この処理が繰り返されます。

注意 GUI または接続文字列で指定された代替サーバのリストは、初期接続時にのみ使用されます。使用可能なインスタンスとの接続の確立後、高可用性をサポートしているクライアントは、最適なフェールオーバ・ターゲットを含む最新のリストをサーバから受信します。この新しいリストは、指定されたリストを上書きします。

ディレクトリ・サービス

ディレクトリ・サービスを使用すると、Adaptive Server OLE DB プロバイダは中央にある LDAP サーバから接続やその他の情報を取得して Adaptive Server に接続できます。ここでは、DSURL (Directory Service URL) というプロパティを使用して、データを取得する LDAP サーバを示します。

ディレクトリ・サービスとしての LDAP

LDAP (Lightweight Directory Access Protocol) は、ディレクトリ・サービスへの業界標準のアクセス方法です。ディレクトリ・サービスを使用すると、コンポーネントは LDAP サーバから情報を DN (識別名) で検索できます。LDAP サーバは、企業またはネットワーク上で使用されるサーバ、ユーザ、ソフトウェアの情報を格納したり管理したりします。

LDAP サーバは、Adaptive Server やクライアントとは異なるプラットフォームに配置できます。LDAP は、クライアントとサーバが交換するメッセージの通信プロトコルと内容を定義します。LDAP サーバに格納され、取得が可能な情報は、次のとおりです。

- Adaptive Server に関する情報 (IP アドレス、ポート番号、ネットワーク・プロトコルなど)
- セキュリティ・メカニズムとフィルタ
- 高可用性コンパニオン・サーバ名

詳細については、『Adaptive Server Enterprise システム管理ガイド』を参照してください。

LDAP サーバの設定時に、次のアクセス制限を指定できます。

- 匿名認証 － すべてのユーザがあらゆる情報にアクセスできます。
- ユーザ名とパスワードによる認証 － Adaptive Server は、次のファイルで指定されているデフォルトのユーザ名とパスワードを使用します。

ユーザ名とパスワードによる認証のプロパティによって、LDAP サーバとのセッション接続が確立され、終了します。

ディレクトリ・サービスの使用

ディレクトリ・サービスを使用するには、ConnectString に次のプロパティを追加します。

```
DSURL= ldap://SYBLDAP:389/dc=sybase,dc=com??one?sybase
Servername=MANGO
```

URL は LDAP URL で、LDAP ライブラリを使用して URL を解決します。

LDAP サーバの高可用性をサポートするため、DSURL はセミコロンで区切られた複数の URL を受け入れます。次に例を示します。

```
DSURL={ldap://SYBLDAP:389/dc=sybase,dc=com??one?sybaseServern
ame=MANGO;
ldap://SYBLDAP1:389/dc=sybase,dc=com??one?sybaseServername=MA
NGO}
```

プロバイダは、LDAP サーバからプロパティを指定された順序で取得しようとします。

DSURL は次のように指定します。

```
ldap://hostport/dn[?attrs[?scope[?filter[?userdn?userpass]]]]
```

パラメータの意味は次のとおりです。

- *hostport* は、次のような、ホスト名とオプションの `portnumber` です。次に例を示します。`SYBLDAP1:389`。
- `dn` は検索ベースです。次に例を示します。
  `dc=sybase, dc-com`
- *attrs* は、LDAP に要求される属性のカンマ区切りリストです。これはブランクにします。Data Provider はすべての属性を必要とします。
- *scope* は、次の 3 つの文字列のいずれかになります。
  - `base` (デフォルト) － ベースを検索します。
  - `one` － 直下の子を検索します。
  - `sub` － サブ・ツリーを検索します。

- *filter* は検索フィルタであり、通常は sybaseServername です。ここをブランクにする場合は、Data Source または ConnectionString の Server Name プロパティを設定します。
- *userdn* は、ユーザの DN です。LDAP サーバが匿名ログインをサポートしていない場合は、ここでユーザの DN を設定するか、ConnectionString の DSPrincipal プロパティを設定します。
- *userpass* はパスワードです。LDAP サーバが匿名ログインをサポートしていない場合は、ここでパスワードを設定するか、ConnectionString の DSPassword プロパティを設定します。

interfaces ファイル

インタフェース・ファイルを使用した場合、次のいずれかのようになります。

- `file///mils1/sybase/ini/sql.ini`
- `file:///c:/sybase/ini/sql.ini`
- `file://c:¥sybase¥ini¥sql.ini`
- `file:///sql.ini`

この場合、URL はファイルで、そのインタフェース・ファイルのロケーションを解決します。デフォルトでは、Adaptive Server OLE DB プロバイダは、インタフェース・ファイルの完全なファイル URL があると見なします。完全なパスが指定されない場合、ドライバは *$SYBASE* ツリー構造でファイルを検索します。これは、すべての Sybase 製品のインタフェース・ファイルのデフォルトのロケーションです。

URL に *sybaseServername* を組み込むことも、Server Name プロパティを LDAP Sybase サーバ・オブジェクトのサービス名に設定することもできます。

次のプロパティは、ディレクトリ・サービスを使用する場合に役立ちます。

- DSURL − LDAP URL に設定します。デフォルトは空の文字列です。
- Server − LDAP Sybase サーバ・オブジェクトのサービス名。デフォルトは空の文字列です。
- DSPrincipal − LDAP サーバが DSURL の一部ではなく匿名アクセスが許可されない場合に、このサーバにログオンするユーザ名。
- DSPassword または Directory Service Password − LDAP が DSURL の一部ではなく匿名アクセスが許可されない場合に、このサーバで認証するパスワード。

パスワードの暗号化

Adaptive Server OLE DB プロバイダはデフォルトで、ネットワークを介してプレーン・テキストのパスワードを Adaptive Server に送信して認証を求めます。ただし、Adaptive Server OLE DB プロバイダは、パスワードの対称／非対称暗号化もサポートしています。これによってデフォルトの動作を変更し、パスワードを暗号化してからネットワークに送信できます。

対称暗号化メカニズムでは、パスワードの暗号化と復号化に同じキーが使用されます。これに対して、非対称暗号化メカニズムでは、暗号化にはパブリック・キー、復号化には別のプライベート・キーが使用されます。プライベート・キーはネットワークを介して共有されないため、非対称暗号化の方が対称暗号化よりも安全であると考えられます。パスワードの暗号化が有効になっていて、サーバが非対称暗号化をサポートしている場合、非対称暗号化が対称暗号化の代わりに使用されます。

Sybase CSI (Common Security Infrastructure) を使用して、ログイン・パスワードとリモート・パスワードを暗号化できます。CSI 2.6 は、連邦情報処理標準 (FIPS: Federal Information Processing Standard) 140-2 に準拠しています。

パスワードの暗号化の有効化

パスワードの暗号化を有効にするには、EncryptPassword 接続プロパティを設定する必要があります。この接続プロパティでは、パスワードが暗号化フォーマットで転送されるかどうかを指定します。パスワードの暗号化が有効になっていると、ログインがネゴシエートされた場合にのみ、パスワードはネットワークに送信されます。パスワードは最初に暗号化されてから送信されます。EncryptPassword の値は次のとおりです。

- 0 − プレーン・テキスト形式のパスワードを使用します。これはデフォルトの値です。
- 1 − 暗号化されたパスワードを使用します。サポートされていない場合、エラー・メッセージを返します。
- 2 − 暗号化されたパスワードを使用します。サポートされていない場合、プレーン・テキスト形式のパスワードを使用します。

注意 非対称のパスワード暗号化を使用するには、非対称暗号化をサポートするサーバ (Adaptive Server 15.0.2 など) が必要です。非対称暗号化では、追加の処理時間が必要になり、ログインに若干の遅延が発生する可能性があります。

❖ **暗号化パスワード**

1　Sybase Data Source Administrator を起動します。

2　データ・ソースを選択します。

3　暗号化パスワードを確認します。

注意 ユーザ・インタフェースを使用して EncryptPassword を 0 または 1 に設定します。EncryptPassword を 2 に設定するには、接続文字列を使用します。

例

```
Data Source=MANGO:5000;UserID=sa;pwd=sybase;EncryptPassword=2
```

パスワード有効期限の処理

各企業は、自社のデータベース・システム用に独自のパスワード・ポリシーを設定しています。ポリシーに応じて、パスワードは特定の日時で期限切れになります。パスワードがリセットされない限り、データベースに接続した Adaptive Server ドライバはパスワード期限切れエラーをスローし、isql を使用してパスワードを変更するようユーザに要求します。パスワード変更機能を使用すると、別のツールを使用しなくても、期限切れになったパスワードを変更できます。

パスワードを変更するには、次の 2 つの接続プロパティを設定します。

- oldpassword － 現在のパスワード。oldpassword に null や空の文字列以外の値が含まれている場合、現在のパスワードは pwd に含まれる値に変更される。
- pwd － ユーザが入力した新しいパスワードの値が含まれる。oldpassword が存在しないか null の場合、pwd には現在のパスワード値が含まれる。

SSL を使用したデータの暗号化

SSL (Secure Sockets Layer) は、クライアントとサーバ間、およびサーバ同士の接続において、ワイヤ・レベルまたはソケット・レベルで暗号化されたデータを送信する業界標準です。サーバとクライアントが、安全な暗号化セッションをネゴシエートして合意してから、SSL 接続が確立されます。これは、“SSL ハンドシェイク ” と呼ばれています。

注意 安全なセッションの確立に追加のオーバヘッドが必要になります。データを暗号化するとサイズが増え、情報の暗号化と復号化に追加の計算が必要になるからです。一般に、SSL ハンドシェイク中に生じる I/O の増加によって、ユーザ・ログインにかかる時間が 10 ～ 20 倍になることがあります。

SSL ハンドシェイク

クライアント・アプリケーションが接続を要求すると、SSL 対応サーバは証明書を提示し、ID を証明してから、データを送信します。基本的に、SSL ハンドシェイクは次の手順によって構成されています。

1 クライアントがサーバに接続要求を送信します。要求には、クライアントがサポートしている SSL (または TLS: Transport Layer Security) オプションが含まれています。

注意 TSL (Transport Layer Security) は、SSL 3.0 の拡張バージョンで、SSL バージョン 3.0 CipherSuite のエイリアスでもあります。

2 サーバは、証明書とサポートされている CipherSuite (次項を参照してください) のリストを返します。これには、SSL/TLS サポート・オプション、キー交換で使用されるアルゴリズム、デジタル署名が含まれます。

3 クライアントとサーバの両者が 1 つの CipherSuite (次項を参照してください) について合意すると、安全で暗号化されたセッションが確立されます。

CipherSuite

SSL ハンドシェイク中に、クライアントとサーバは、CipherSuite を介して共通のセキュリティ・プロトコルをネゴシエートします。CipherSuite は、SSL プロトコルで使用されるキー交換アルゴリズム、ハッシュ方式、暗号化方式の優先順位リストです。

デフォルトでは、クライアントとサーバの両方がサポートしている最も強力な CipherSuite は、SSL ベースのセッションに使用される CipherSuite です。サーバ接続パラメータは、接続文字列か、LDAP などのディレクトリ・サービスによって指定されます。

Adaptive Server OLE DB プロバイダおよび Adaptive Server は、SSL Plus ライブラリ API と暗号エンジンである Security Builder (両方とも Certicom Corporation 製) で使用可能な暗号スイートをサポートしています。

注意 次に示す CipherSuite のリストは TLS 仕様に準拠しています。

次に、Adaptive Server OLE DB プロバイダでサポートされる CipherSuites を最も強力なものから順に示します。

- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_RC4_128_SHA
- TLS_RSA_WITH_RC4_128_MD5
- TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- TLS_DHE_DSS_WITH_RC4_128_SHA
- TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_DES_CBC_SHA

- TLS_DHE_DSS_WITH_DES_CBC_SHA
- TLS_DHE_RSA_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT1024_WITH_RC4_56_SHA
- TLS_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA
- TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT_WITH_RC4_40_MD5
- TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
- TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA

SSL ハンドシェイクと SSL/TLS プロトコルの詳細については、Internet Engineering Task Force Web Site (http://www.ietf.org) を参照してください。

CipherSuite の詳細については、IETF organization Web Site (http://www.ietf.org/rfc/rfc2246.txt) を参照してください。

Adaptive Server OLE DB プロバイダの SSL セキュリティ・レベル

Adaptive Server OLE DB プロバイダでは、SSL は次のセキュリティ・レベルを提供します。

- SSL セッションが確立されると、ユーザ名とパスワードが暗号化された安全で接続によって送信されます。
- SSL 対応サーバへの接続を確立すると、サーバは接続対象のサーバであることを自己認証し、暗号化された SSL セッションが開始され、データが送信されます。
- サーバ証明書のデジタル署名をチェックして、サーバから受信したデータが転送中に変更されたかどうかを判断します。

証明書によるサーバの検証

Adaptive Server OLE DB プロバイダのクライアントが SSL 対応サーバに接続する場合は、サーバには証明書ファイルが必要です。これには、サーバの証明書と暗号化プライベート・キーが含まれています。また、証明書は署名 / 認証局 (CA: Certification Authority) によってデジタル署名されている必要もあります。既存のクライアント接続が確立されるのと同じように、Adaptive Server OLE DB プロバイダのクライアント・アプリケーションは Adaptive Server へのソケット接続を確立します。ネットワークのトランスポート層の接続コールがクライアント・サイドで完了し、受け入れコールがサーバ・サイドで完了すると、SSL ハンドシェイクが行われます。それから、ユーザのデータが送信されます。

SSL 対応サーバに正しく接続するには、次のことが必要です。

1 クライアント・アプリケーションが接続要求を行った場合、SSL 対応サーバは証明書を提示しなければなりません。

2 クライアント・アプリケーションは、証明書に署名した CA を認識しなければなりません。「信頼された」CA すべてを含んだリストは、次に示す「信頼されたルート・ファイル」にあります。

信頼されたルート・ファイル

既知で信頼された CA のリストは、信頼されたルート・ファイルに保管されています。エンティティ (クライアント・アプリケーション、サーバ、ネットワーク・リソースなど) に既知の CA の証明書がある以外は、信頼されたルート・ファイルは証明書ファイルのフォーマットと同じです。システム・セキュリティ担当者が、標準 ASCII テキスト・エディタを使って、信頼された CA を追加したり、削除したりします。

アプリケーション・プログラムでは、ConnectString の TrustedFile=*trusted file path* プロパティを使用して、信頼されたルート・ファイルの位置を指定します。最も一般的に使用される CA (Thawte、Entrust、Baltimore、VeriSign、RSA) が記載された信頼されたルート・ファイルは *$SYBASE/config/trusted.txt* にインストールされています。

証明書の詳細については、『Open Client Client-Library C リファレンス・マニュアル』を参照してください。

SSL 接続の有効化

Adaptive Server OLE DB プロバイダで SSL を有効にするには、ConnectString に Encryption=*ssl* と TrustedFile=<*filename*> を追加します (*filename* は信頼されたルート・ファイルへのパスです)。これで、Adaptive Server OLE DB プロバイダが Adaptive Server と SSL 接続をネゴシエートするようになります。

注意 SSL を使用するように、Adaptive Server を設定してください。SSL の詳細については、『Adaptive Server Enterprise システム管理ガイド』を参照してください。

❖ Microsoft Windows での SSL 接続の有効化

1 接続文字列内の Encryption プロパティを `ssl` に設定します。

2 接続文字列内の TrustedFile プロパティに信頼されたルート・ファイルのファイル名を設定します。ファイル名には、そのファイルへのパスも含める必要があります。

OLE DB でのブックマークおよびバッチ・オペレーションのサポート

Sybase では、OLE DB プロバイダに対して、ブックマーク・オペレーションと SQL バッチ・オペレーションをサポートします。

IRowsetLocate インタフェースを使用するブックマーク・オペレーションでは、ブックマークの比較や、ブックマークに基づくローの取得を行うことができます。バッチ・オペレーションは IRowsetUpdate インタフェースを介してサポートされ、ローのバッチの更新、挿入、および削除の方法が提供されます。IRowsetLocate と IRowsetUpdate の使用方法については、Microsoft Developer Network (http://msdn.microsoft.com) を参照してください。

Adaptive Server OLE DB プロバイダでの高可用性 (HA) フェールオーバ

表 3-1 に、Adaptive Server OLE DB プロバイダの、高可用性 (HA) フェールオーバで使用される接続パラメータを示します。

表 *3-1: HA* フェールオーバ接続パラメータ

プロパティ名	説明	必須	デフォルト値
HASession	高可用性を有効にするかどうかを指定する。0 は高可用性を無効にし、1 は高可用性を有効にする。	いいえ	0
SecondaryPort	アクティブ／アクティブ設定またはアクティブ／パッシブ設定でフェールオーバ・サーバとして機能する Adaptive Server のポート番号。	はい (HASession が 1 に設定されている場合)	空
SecondaryServer	アクティブ／アクティブ設定またはアクティブ／パッシブ設定でフェールオーバ・サーバとして機能する Adaptive Server の名前または IP アドレス。	はい (HASession が 1 に設定されている場合)	空

HA システムにおけるフェールオーバの使用

HA クラスタには、2 つ以上のマシンが含まれます。これらのマシンは、1 つのマシン (またはアプリケーション) が中断した場合にもう 1 つのマシンが両方のマシンの負荷を処理するように設定されています。それぞれが、高可用性クラスタのノードです。HA クラスタはシステムが常に稼働していなければならないような環境で使用します。たとえば、クライアントが 1 年 365 日絶えず接続する銀行のシステムなどです。

マシンは、他のマシンのディスクを各マシンで読み込みできるように設定されています。ただし、同時読み込みはできません (フェールオーバで使用するすべてのディスクは共有ディスクに設定してください)。

たとえば、プライマリ・コンパニオン・サーバである Adaptive Server 1 に障害が発生したとします。セカンダリ・コンパニオン・サーバである Adaptive Server 2 は、Adaptive Server 1 を再起動できるようになるまでそのディスク (ディスク 1 ～ 4) を読み込んで、ディスク上のデータベースをすべて管理します。Adaptive Server 1 に接続していたクライアントは、自動的に Adaptive Server 2 に接続されます。

フェールオーバによって、Adaptive Server をアクティブ／アクティブ設定またはアクティブ／パッシブ設定の高可用性クラスタで運用できます。

フェールオーバが発生すると、プライマリ・コンパニオンに接続していたクライアントは、フェールオーバ・プロパティを使用して、自動的にセカンダリ・コンパニオンへのネットワーク接続を再確立します。フェールオーバを有効にするには、接続プロパティ HASession を 1 (デフォルト値は 0) に設定します。このプロパティを設定しないと、サーバでフェールオーバが設定されていても、セッションではフェールオーバが行われません。SecondaryServer (セカンダリ Adaptive Server サーバの IP アドレスまたはマシン名) と SecondaryPort (セカンダリ Adaptive Server サーバのポート番号) のプロパティも設定する必要があります。使用するシステムの高可用性設定の詳細については、Adaptive Server Enterprise の『高可用性システムにおける Sybase フェールオーバの使用』を参照してください。

Adaptive Server OLE DB プロバイダでプライマリ Adaptive Server サーバの接続エラーが検出されると、最初にプライマリ・サーバへの再接続が試行されます。再接続できない場合はフェールオーバが行われたと見なされます。次に、SecondaryServer と SecondaryPort に設定された接続プロパティを使用して、セカンダリ Adaptive Server への接続が自動的に試行されます。

フェールオーバの成功を確認する方法

セカンダリの Adaptive Server への接続が確立されると、Adaptive Server OLE DB プロバイダは関数のリターン・コード HRESULT に “E_FAIL” を返します。

フェールオーバが成功したことを確認するには、ERRORINFO の dwMinor フィールド (IErrorRecords::GetBasicErrorInfo の戻り値) または IErrorInfo::GetDescription から返される説明を確認してください。HA フェールオーバが成功していれば、dwMinor の値は “30130” になります。IErrorInfo::GetDescription では次の説明が返されます。文中の *ASEServerName* は、フェールオーバ先のサーバ名を表します。

```
"Sybase server is not available or has terminated your
connection, you have successfully connected to the next
available HA server ASEServerName.All transactions has
been rolled back."
```

注意 フェールオーバが成功したかどうかの確認には、エラー説明ではなく、dwMinor により返されるコードを確認することをおすすめします。

クライアントは、新しい接続を使用して、失敗したトランザクションを再適用しなければなりません。トランザクションのオープン中にフェールオーバが発生した場合、フェールオーバ前にデータベースにコミットされた変更のみが保持されます。

フェールオーバの失敗の確認

セカンダリ・サーバへの接続が確立されない場合でも、Adaptive Server OLE DB プロバイダは関数のリターン・コード HRESULT に “E_FAIL” を返します。ただし、ERRORINFO の dwMinor フィールド (IErrorRecords::GetBasicErrorInfo の戻り値) は “30131” になり、IErrorInfo::GetDescription から返される説明は次のようになります。

```
"Connection to Sybase server has been lost, connection
to the next available HA server also failed.All
transactions have been rolled back."
```

フェールオーバを確認するサンプル・コード

次のコードは、フェールオーバのコーディング方法を示しています。

```
/* Declare required variables */
...
/* Open Database connection */
...
/* Perform a transaction */
...
/*Check HRESULT  and dwMinor in ERRORINFO, handle failover */
if (FAILED(hr))
{
   IErrorInfo* pIErrorInfo;
   GetErrorInfo(0, &pIErrorInfo);
   IErrorRecords * pIErrorRecords;
   HRESULT hr1 = pIErrorInfo->QueryInterface(IID_IErrorRecords,
      (void **)&pIErrorRecords);
   if (SUCCEEDED(hr1))
   {
      ERRORINFO errorInfo;
      pIErrorRecords->GetBasicErrorInfo(0, &errorInfo);
      pIErrorRecords->Release();
      if (errorInfo.dwMinor == 30130)
      {
         //successful failover,
         //retry the transaction
      }
   }
}
```

Kerberos による認証

Kerberos は、簡単なログイン認証と相互のログイン認証を提供する業界標準のネットワーク認証システムです。Kerberos は、ユーザとサービスを認証します。Kerberos を使用して、さまざまなアプリケーションにわたるシングル・サインオンを非常に安全な環境内で行えます。ネットワークでパスワードを渡す代わりに、Kerberos サーバがユーザのパスワードと使用可能なサービスのパスワードの暗号化されたバージョンを保持します。

さらに Kerberos では、機密性とデータの整合性を維持するために暗号化を使用します。

Adaptive Server と Adaptive Server OLE DB プロバイダは、Kerberos 接続をサポートします。Adaptive Server OLE DB プロバイダは特に、MIT、CyberSafe、Active Directory の KDC をサポートします。

プロセスの概要

基本的に、Kerberos 認証プロセスは次のように機能します。

1 クライアント・アプリケーションは、特定のサービスにアクセスするための「チケット」を Kerberos サーバに要求します。

2 Kerberos サーバは、2 つのパケットを含むチケットをクライアントに返します。第 1 のパケットはユーザ・パスワードにより暗号化されます。第 2 のパケットはサービス・パスワードにより暗号化されます。これらの各パケット内に「セッション・キー」が含まれます。

3 クライアントは、セッション・キーを取得するためにユーザ・パケットを復号化します。

4 クライアントは新しい認証パケットを作成し、それをセッション・キーにより暗号化します。

5 クライアントは、認証パケットとサービス・パケットをサービスに送信します。

6 サービスは、セッション・キーを取得するためにサービス・パケットを復号化し、ユーザ情報を取得するために認証パケットを復号化します。

7 サービスは、認証パケットからのユーザ情報と、サービス・パケットにも含まれているユーザ情報を比較します。両者が一致する場合、ユーザは認証済みです。

8 サービスは、認証パケットに含まれる検証データに加えてサービス固有の情報を含む確認パケットを作成します。

9 サービスは、このデータをセッション・キーとともに暗号化し、それをクライアントに返します。

10　クライアントは、パケットを復号化するために Kerberos から受信したユーザ・パケット内のセッション・キーを使用し、サービスがそれ自身の主張に一致しているかどうかを検証します。

こうした方法で、ユーザとサービスは相互に認証されます。将来は、クライアントとサービス (この場合は Adaptive Server データベース・サーバ) の間の通信はすべて、セッション・キーにより暗号化されるようになります。これにより、サービスとクライアント間で送信されるすべてのデータが望ましくない閲覧者から正しく保護されます。

稼働条件

認証システムとして Kerberos を使用するには、Kerberos に認証を委任するように Adaptive Server Enterprise を設定します。詳細については、『Adaptive Server Enterprise システム管理ガイド』を参照してください。

Adaptive Server で Kerberos を使用するように設定されている場合、Adaptive Server と通信するすべてのクライアントが Kerberos クライアント・ライブラリをインストールする必要があります。これは、各種オペレーティング・システムのベンダごとに異なります。

- Microsoft Windows では、Windows Active Directory クライアント・ライブラリがクライアント・ライブラリとともにインストールされます。
- Microsoft Windows では、CyberSafe と MIT のクライアント・ライブラリを使用できます。

詳細については、ベンダのマニュアルを参照してください。

Kerberos 認証の有効化

ドライバに対して Kerberos を有効にするには、プログラムに次を追加します。

```
AuthenticationClient=<one of 'mitkerberos' or
'cybersafekerberos' or 'activedirectory'> and
ServerPrincipal=<ASE server name>
```

ここで *<ASE server name>* は、KDC (Key Distribution Center) 内で設定されたサーバの論理名またはプリンシパルです。ドライバはこの情報を使用して、設定された KDC および Adaptive Server サーバと Kerberos 認証をネゴシエートします。

Kerberos クライアントで別のキャッシュ内の TGT を検索する必要がある場合は、userprincipal メソッドを指定できます。

SQL_DRIVER_NOPROMPT とともに SQLDriverConnect を使用する場合、ConnectString は次のようになります。

```
char ConnectString[BUFSIZ];
strcpy(ConnectString, "Driver=Adaptive Server Enterprise;");
strcat(ConnectString, "UserID=sa;Password=;");
strcat(ConnectString, "Server=sampleserver;");
strcat(ConnectString, "Port=4100;Database=pubs2;");
strcat(ConnectString, "UseCursor=1;");
strcat(ConnectString, "AuthenticationClient=mitkerberos;");
strcat(ConnectString, " ServerPrincipal=MANGO;");
```

Key Distribution Center からの初期チケットの取得

Kerberos 認証を使用するには、Key Distribution Center から TGT (Ticket Granted Ticket) と呼ばれる初期チケットを生成します。このチケットを取得する手順は、使用する Kerberos ライブラリに応じて異なります。詳細については、ベンダのマニュアルを参照してください。

❖ MIT Kerberos クライアント・ライブラリ用の TGT の生成

1 コマンド・ラインに次のように入力して kinit ユーティリティを開始します。

```
% kinit
```

2 *your_name@YOUR.REALM* などの kinit ユーザ名を入力します。

3 your_name@YOUR.REALM のパスワード (“my_password” など) を入力します。kinit ユーティリティにより TGT (Ticket Granting Ticket) に対する要求が認証サーバに送信されます。

このパスワードは、キーの計算のために使用されます。そのキーは、応答の一部を復号化するために使用されます。この応答には、セッション・キーに加えて要求の確認が含まれます。パスワードを正しく入力していれば、この段階で TGT が取得されています。

4 TGT が取得されていることを確認するには、コマンド・ラインに次のように入力します。

```
% klist
```

klist コマンドの結果は次のようになるはずです。

```
Ticket cache:/var/tmp/krb5cc_1234
Default principal:your_name@YOUR.REALM
Valid starting       Expires              Service principal
24-Jul-95 12:58:02   24-Jul-95 20:58:15   krbtgt/YOUR.REALM@YOUR.REALM
```

結果の説明

チケット・キャッシュ　チケット・キャッシュ・フィールドにより、どのファイルにクレデンシャル・キャッシュが含まれているかがわかります。

デフォルトのプリンシパル　デフォルトのプリンシパルは、TGT を所有するユーザ (この場合は自身) のログインです。

有効な開始／期限／サービス・プリンシパル　以降の出力は、既存のチケットのリストです。これは要求した最初のチケットであるため、1 つのチケットのみがリストに含まれています。サービス・プリンシパル (`krbtgt/YOUR.REALM@YOUR.REALM`) は、このチケットが TGT であることを示しています。このチケットは、約 8 時間有効です。

分散トランザクションでの Adaptive Server OLE DB プロバイダの使用

この機能では、分散トランザクションを管理するトランザクション・コーディネータとして Microsoft 分散トランザクション・コーディネータ (MS DTC) が使用されている必要があります。

Sybase は次のプログラミング・モデルをサポートしています。

- MS DTC を直接使用するアプリケーション
- MTS (Microsoft Transaction Server) または (COM+) を使用するアプリケーション

MS DTC のプログラミング

❖ **Microsoft 分散トランザクション・コーディネータ (MS DTC) を使用したプログラミング**

1. DtcGetTransactionManager 関数を使用して MS DTC に接続します。MS DTC の詳細については、Microsoft 分散トランザクション・コーディネータのマニュアルを参照してください。
2. OLE DB の手順に従って、確立する Adaptive Server 接続ごとに IDBSession を取得します。
3. ITransactionDispenser::BeginTransaction 関数を呼び出して MS DTC トランザクションを開始し、そのトランザクションを表す OLE トランザクション・オブジェクトを取得します。

4 MS DTC トランザクション リストに登録する IDBSession (OLE DB 接続) に ITransactionJoin を問い合わせ、渡された (手順 3 で取得した) パラメータ punkTransactionCoord をトランザクション・オブジェクトとして使用して、JoinTransaction を呼び出します。現時点では、Sybase は、分散トランザクションの独立性レベルとして ISOLATION LEVEL_READCOMMITTED のみをサポートしています。ITransactionOptions はサポートしていません。

5 SQL Server を更新するには、IDBCommand を作成して実行するための OLE DB 手順に従います。

6 ITransaction::Commit 関数を呼び出して MS DTC トランザクションをコミットします。これでトランザクション・オブジェクトは無効になります。

MTS または COM+ に展開されるコンポーネントのプログラミング

次の手順は、MTS または COM+ で分散トランザクションに関与するコンポーネントの作成方法を説明しています。

❖ MTS または COM+ に展開されるコンポーネントのプログラミング

1 Adaptive Server 接続ごとに IDBSession を作成します。

2 実行する更新ごとに IDBCommand を作成して実行します。

3 コンポーネントを MTS または COM に展開し、必要に応じてトランザクション属性を設定します。

トランザクションの作成、トランザクションへの関与、トランザクションのコミットまたはロールバックは、COM+、OLE DB サービス、OLE DB プロバイダによって処理されます。

自動トランザクション登録を実行するには、OLE DB サービスが必要です。OLE DB サービスを有効にするには、ルールに従ってデータ・ソースを初期化する必要があります (MS OLE DB のマニュアルを参照)。自動トランザクション登録を有効にするには、*OLE_DB_SERVICES* レジストリと DBPROP_INIT_OLEDBSERVICES プロパティ値に DBPROPVAL_OS_TXNENLISTMENT ビットを設定するか、接続文字列で OLE DB Services = 2 を渡します。

分散トランザクションでの接続プロパティのサポート

ここでは、接続プロパティについて説明します。

- 分散トランザクション・プロトコル (DistributedTransactionProtocol) － 分散トランザクションをサポートするために使用されるプロトコルを指定するには、XA インタフェース標準または MS DTC OLE ネイティブ・プロトコルのいずれかを使用し、[OLE DB データ・ソース] ダイアログで [**分散トランザクション・プロトコル**] を選択し、OLE ネイティブ・プロトコルの接続文字列のプロバイダ文字列部分にプロパティ DistributedTransactionProtocol = *OLE* を設定するか、デフォルトのプロトコルである *XA* を使用します。
- 密結合トランザクション (TightlyCoupledTransaction) － 2 つのリソース・マネージャを使用する分散トランザクションで同一の Adaptive Server サーバを指定すると、「密結合トランザクション」という状態になります。この場合、このプロパティを 1 に設定していないと分散トランザクションが失敗することがあります。

つまり、同一の Adaptive Server サーバに対して 2 つのデータベース接続をオープンしてから、オープンした接続を同一の分散トランザクションに登録する場合は、TightlyCoupledTransaction=1 を設定することをおすすめします。このプロパティを設定するには、[OLE DB データ・ソース] ダイアログで [密結合トランザクション] を選択するか、接続文字列のプロバイダ文字列部分でプロパティ TightlyCoupledTransaction=1 を渡します。

索引

か

き

く

け

さ

し

す

せ

て

と

に

ね

は

ふ

ま

り