SYBASE®
An SAP® Company

ユーザーズガイド

Adaptive Server® Enterprise ADO.NET Data Provider 15.7 SP100

Microsoft Windows

ドキュメント ID： DC00067-01-1570100-01
改訂： 2013 年 5 月

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

目次

Adaptive Server ADO.NET Data Provider

Adaptive Server® ADO.NET Data Provider は、Adaptive Server Enterprise 用の ADO.NET プロバイダです。 Adaptive Server バージョン 12.5.x、15.0.x、15.5、15.7 でサポートされています。

Adaptive Server ADO.NET Data Provider を使用すると、C#、Visual Basic .NET、マネージ拡張を備えた C++、J# など、.NET でサポートされる任意の言語を使用して Adaptive Server 内のデータにアクセスできます。 .NET 共通言語ランタイム (CLR: Common Language Runtime) アセンブリであり、ADO.NET インタフェース全般の機能を提供する、一連の必要なクラスをすべて含んだクラスライブラリに相当します。 すべてのクラスはマネージコードで、任意のマネージクライアントコードからアクセスできます。 このような各言語間の通信は、Microsoft .NET Framework によって実現します。

Adaptive Server ADO.NET Data Provider を使用する主な利点としては、次のものが挙げられます。

- Adaptive Server ADO.NET Data Provider は、OLE DB プロバイダよりも高速です。
- .NET 環境において、Adaptive Server ADO.NET Data Provider は Adaptive Server に対するネイティブアクセスを提供します。 サポートされるその他のプロバイダとは異なり、Adaptive Server と直接通信できるため、ブリッジ技術を必要としません。

Adaptive Server ADO.NET Data Provider 配備の要件

Adaptive Server ADO.NET Data Provider の日々のための要件を確認してください。

サポートされているプラットフォームのリストについては、Sybase® Platform Certifications ページ (http://certification.sybase.com/ucr/search.do) を参照してください。

システム要件

Adaptive Server ADO.NET Data Provider のインストールの最小システム要件

コンピュータに以下がインストールされている必要があります。

- 開発時 - NET Framework SDK 2.0 以降および VisualStudio.NET 2005 以降
- 配備時 - NET Framework 2.0 以降

必要なファイル
Adaptive Server ADO.NET Data Provider は、クライアントコードによって参照される以下のプロバイダアセンブリファイルで構成されています。

- `Sybase.AdoNet2.AseClient.dll` - .NET 2.0、.NET 3.0、および .NET 3.5 の機能がサポートされます。
- `Sybase.AdoNet4.AseClient.dll` - .NET 4.0 以降の機能がサポートされます。
  これらのファイルの 32 ビット版は `C:¥Sybase¥DataAccess¥ADONET¥dll` ディレクトリにインストールされ、64 ビット版は `C:¥Sybase¥DataAccess64¥ADONET¥dll` ディレクトリにインストールされます。

グローバルアセンブリキャッシュへの Adaptive Server ADO.NET Data Provider アセンブリの配備

多くの場合、1 台のコンピュータ上の複数のアプリケーションで、Adaptive Server ADO.NET Data Provider アセンブリが共有されています。 この結果、アセンブリのコピーが重複して存在することになり、互換性やバージョン管理の問題が生じます。

このような状況を回避するため、Adaptive Server ADO.NET Data Provider アセンブリをグローバルアセンブリキャッシュ (GAC: Global Assembly Cache) に配備することをおすすめします。グローバルアセンブリキャッシュはコンピュータ全体を対象範囲とするキャッシュであり、同じコンピュータ上の複数のアプリケーションによって共有されているアセンブリを格納および管理します。 このような配備を行うことができない場合は、プロバイダを使用するアプリケーションが実行されるすべてのディレクトリに、Adaptive Server ADO.NET Data Provider アセンブリのコピーをインストールしてください。

アセンブリは Adaptive Server ADO.NET Data Provider のインストールプログラムによって自動的に GAC に配備されます。 インストールプログラムを使用しない場合は、手動でアセンブリを配備してください。 これを行うには、**AseGacUtility** または **AseGacUtility4** (.NET 4.0 以降の場合) を実行するか、.NET Framework 構成ツールを使用します。

AseGacUtility または AseGacUtility4 の実行によるアセンブリの配備

AseGacUtility または **AseGacUtility4** ユーティリティを実行して、アセンブリを配備します。

1. **AseGacUtility** または **AseGacUtility4** がインストールされているディレクトリに移動します。デフォルトのロケーションは、Adaptive Server ADO.NET Data Provider の 32 ビット版では `C:¥Sybase¥DataAccess¥ADONET¥dll`、

Adaptive Server ADO.NET Data Provider の 64 ビット版では C:¥Sybase ¥DataAccess64¥ADONET¥dll です。

2. **次のコマンドを実行します。**

```
AseGacUtility /i DLL_Name
```

また、次を使用することもできます。

```
AseGacUtility4 /i DLL_Name
```

DLL_Name では、GAC に配備する DLL を指定します。

たとえば、Sybase.AdoNet2.AseClient.dll の 32 ビット版を配備するには、次のコマンドを実行します。

```
AseGacUtility /i
C:¥Sybase¥DataAccess¥ADONET¥dll¥Sybase.AdoNet2.AseClient.dll
```

同様に 64 ビット版の Sybase.AdoNet4.AseClient.dll を配備するには、次のコマンドを実行します。

```
AseGacUtility4 /i
C:¥Sybase¥DataAccess64¥ADONET¥dll¥Sybase.AdoNet4.AseClient.dll
```

.NET Framework 構成ツールを使用したアセンブリの配備

.NET Framework 校正ツールを使用して、アセンブリを配備します。

1. [.NET Framework 構成ツール]を起動します。

   構成ツールの起動方法については、各オペレーティングシステムの Microsoft のマニュアルを参照してください。
2. 左側のツリービューで、[アセンブリ キャッシュ] を選択します。
3. [アセンブリ キャッシュにアセンブリを追加する] リンクをクリックします。
4. [アセンブリの追加] ダイアログボックスで、インストールディレクトリにある Adaptive Server ADO.NET Data Provider アセンブリを検索して、[開く] をクリックします。

   デフォルトのインストールディレクトリは、Adaptive Server ADO.NET Data Provider の 32 ビット版では C:¥Sybase¥DataAccess¥ADONET¥dll、Adaptive Server ADO.NET Data Provider の 64 ビット版では C:¥Sybase ¥DataAccess64¥ADONET¥dll です。 これで、Adaptive Server ADO.NET Data Provider アセンブリが GAC に配備されます。 キャッシュ内のアセンブリのリストを確認するには、[アセンブリ キャッシュのアセンブリ一覧の表示] リンクを選択します。

GAC からのアセンブリの削除

AseGacUtility または **AseGacUtility4** ユーティリティを実行して、GAC からアセンブリを削除します。

1. AseGacUtility または AseGacUtility4 がインストールされているディレクトリに移動します。デフォルトのロケーションは、Adaptive Server ADO.NET Data Provider の 32 ビット版では `C:¥Sybase¥DataAccess¥ADONET¥dll`、Adaptive Server ADO.NET Data Provider の 64 ビット版では `C:¥Sybase¥DataAccess64¥ADONET¥dll` です。
2. **次のコマンドを実行します。**

   ```
   AseGacUtility /u DLL_Name
   ```

   または、次のコマンドを実行します。

   ```
   AseGacUtility4 /i DLL_Name
   ```

   ここで、*DLL_Name* は、GAC から削除する DLL です。

   たとえば、`Sybase.AdoNet2.AseClient.dll` の 32 ビット版を GAC から削除するには、次のコマンドを実行します。

   ```
   AseGacUtility /u
   C:¥Sybase¥DataAccess¥ADONET¥dll¥Sybase.AdoNet2.AseClient.dll
   ```

   たとえば、`Sybase.AdoNet4.AseClient.dll` の 64 ビット版を GAC から削除するには、次のコマンドを実行します。

   ```
   AseGacUtility /u
   C:¥Sybase¥DataAccess64¥ADONET¥dll¥Sybase.AdoNet4.AseClient.dll
   ```

.NET Framework 構成ツールを使用したアセンブリの削除

.NET Framework 構成ツールを使用してアセンブリを削除する方法を確認します。

1. [.NET Framework 構成]ツールを起動します。

   構成ツールの起動方法については、Microsoft 発行の各オペレーティングシステムのマニュアルを参照してください。
2. 左側のツリービューで、[アセンブリ キャッシュ] を選択します。
3. [アセンブリ キャッシュのアセンブリ一覧の表示]リンクをクリックします。
4. アセンブリ名のリストで、`Sybase.AdoNet2.AseClient` または `Sybase.AdoNet4.AseClient` を探します。

   システムに複数のバージョンが配備されている場合は、このアセンブリのエントリがバージョンに対応して複数表示される場合もあります。

5. 削除するアセンブリを 1 つ以上選択します。 右クリックして[削除]を選択します。 [はい]をクリックして操作を確定します。
6. 削除したバージョンに対応する発行者ポリシーファイルがないかを確認し、これらのファイルも削除します。

注意： GAC には、他のアセンブリから特定のアセンブリへの参照も格納されます。この場合、これらの参照が削除されるまで、この参照先のアセンブリは削除できません。 これらの参照は、削除コマンドの一部として強制的に削除できます。 システムによっては、ユーティリティがアセンブリの削除に失敗して、Windows インストーラで保留中の参照に関するエラーが発生することがあります。 これは、レジストリに値が残っているために発生するものです。 この問題が発生した場合は、Microsoft のサポートに連絡して解決策を確認してください。

インストールプログラムと GAC を使用したアプリケーションの配備
インストールプログラムと GAC を使用してアプリケーションを配備します。

1. エンドユーザのコンピュータのインストールプログラムを使用して Adaptive Server ADO.NET Data Provider をインストールします。
2. `exe` や `dll` などのアプリケーション固有ファイルを、システムのアプリケーション固有フォルダにコピーします。

GAC を使用したアプリケーションの配備
GAC を使用して、アプリケーションを配備します。

1. Adaptive Server ADO.NET Data Provider を構成する `dll` ファイルを、ターゲットコンピュータの `C:¥Sybase¥DataAccess¥ADONET¥dll` などのディレクトリにコピーします。
2. このディレクトリをシステムパスに追加します。
3. プロバイダアセンブリを GAC に配備します。「グローバルアセンブリキャッシュへの Adaptive Server ADO.NET Data Provider アセンブリの配備」を参照してください。
4. `exe` や `dll` などのアプリケーション固有ファイルを、システムのアプリケーション固有フォルダにコピーします。
5. アプリケーションを実行します。

GAC を使用しないアプリケーションの配備
GAC を使用しないでアプリケーションを配備する手順を確認します。

1. ターゲットシステムで、`exe` や `dll` などのアプリケーション固有ファイルに加えて、Adaptive Server ADO.NET Data Provider を構成する `dll` ファイルもアプリケーション固有フォルダにコピーします。
2. アプリケーションを実行します。

ADO.NET Data Provider の新しいバージョンへの更新

ADO.NET Data Provider が新しいバージョンに更新されます。 Adaptive Server ADO.NET Data Provider の更新は、EBF/ESD またはメンテナンスリリースから提供されます。

新しいバージョンの Data Provider への更新に関する問題、および更新に関係する Microsoft .NET のコンセプトについては、Microsoft Developer Network (http://msdn.microsoft.com) の『.NET Framework 開発ガイド』および『NET Framework Developer's Guide』を参照してください。

新しいバージョンの Adaptive Server Data Provider にアプリケーションを移行するには、次のいずれかを実行してください。

- アプリケーション設定ファイルを作成して、新しいバージョンの Adaptive Server ADO.NET Data Provider を使用するようにアプリケーションをリダイレクトします。 「アプリケーション設定ファイルの使用」を参照してください。
- 新しいバージョンの Adaptive Server ADO.NET Data Provider に対してアプリケーションを再度ビルドし、配備します。 Sybase では、この手順を選択することをおすすめします。

CLR (共通言語ランタイム (Common Language Runtime))

.NET 共通言語ランタイム (CLR) は、アプリケーションプログラムの実行時、Data Provider などのアセンブリに対する参照を見つけてバインドします。 CLR はデフォルトで、アプリケーションのビルドに使用されたバージョンのアセンブリに対する参照をバインドしようとします。

そのため、配備しただけで更新バージョンのアセンブリが自動的に使用されることはありません。新しいバージョンのアセンブリを使用してアプリケーションを再ビルドするか、新しいバージョンを使用するように設定ファイルで CLR をリダイレクトする必要があります。

一般に、Data Provider の EBF/ESD リリースのリリースレベル (メジャー/マイナー) が同じであれば、前のリリースとの間にバイナリの互換性があります。 このよう

な更新ではアプリケーションの再ビルドは必要ありません。 Data Provider の更新ごとにアプリケーションの再ビルドや再配備を行わない場合は、アプリケーション設定や発行者ポリシーファイルを使用することもできます。 Sybase では通常、ESD/EBF リリースに、適切なリダイレクトを設定した発行者ポリシーファイルを収録しています。 下位互換性の問題の詳細については、ESD/EDF のドキュメントを参照してください。

アプリケーション設定ファイルを使用した CLR のリダイレクト

アプリケーション設定ファイルを使用し、CLR をリダイレクトして、呼び出し側のアプリケーションのマニフェストに格納されているアセンブリとバージョンが異なるアセンブリをロードします。
次の例は、以前の Data Provider 1.0.x で構築されたアプリケーションで Data Provider 1.0.159 を使用するように CLR をリダイレクトする方法を示しています。

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="Sybase.AdoNet2.AseClient"
                          publicKeyToken="95d94fac46c88e1e"
                          culture="neutral" />
        <bindingRedirect oldVersion="1.0.0.0-2.155.999.65535"
                         newVersion="2.155.1000.0"/>
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

設定ファイルスキーマの詳細については、MSDN ライブラリを参照してください。

注意： アプリケーションごとに独自の設定ファイルが必要です。

発行者ポリシーファイルを使用した CLR のリダイレクト

アセンブリの発行元は、共有アセンブリの更新とともに発行者ポリシーファイルを配布できます。

発行者ポリシーファイルは、古いアセンブリバージョンのすべての参照を新しくインストールされたバージョンにリダイレクトします。 アプリケーション設定ファイルとは異なり、発行者ポリシーファイルを機能させるには、グローバルアセンブリキャッシュ (GAC) に配備する必要があります。

発行者ポリシーファイルの設定は、アプリケーションまたはアプリケーション設定ファイルのバージョン情報よりも優先されます。 ただし、「セーフモード」を強制し、特定のアプリケーションで発行者ポリシーファイルを無視するように設定することもできます。 セーフモードを使用するようにアプリケーションを設定する方法については、MSDN ライブラリ (http://msdn2.microsoft.com/en-us/default.aspx) を参照してください。

一般的に、Adaptive Server ADO.NET Data Provider の更新には、最新インストールのバージョンの Data Provider アセンブリにアプリケーションをリダイレクトする発行者ポリシーファイルが組み込まれています。 これにより、新しいプロバイダアセンブリと発行者ポリシーファイルが GAC に配備されます。

Data Provider の更新の配備

Data Provider の更新の配備、および関連する問題についての情報を確認します。

GAC への Data Provider の配備
更新された Data Provider アセンブリとポリシーアセンブリが GAC に配備されると、システム上のすべてのアプリケーションが自動的にこの Provider の使用を開始します。

更新された Data Provider を使用しないように特定のアプリケーションを除外する場合は、そのアプリケーションのアプリケーション設定ファイルを、発行者ポリシーファイルを無効にするセーフモードを強制するように設定します。

GAC にない場合の Data Provider の配備
コンピュータの GAC に Data Provider アセンブリがインストールされていない場合、Data Provider の構成要素であるファイルをアプリケーションフォルダにコピーしてください。 アプリケーションで Data Provider の更新バージョンが使用されるようにする手順は次のとおりです。

- 適切な **redirect** を使用してアプリケーション設定ファイルを作成します。
- 新しいバージョンの Data Provider に合わせてアプリケーションを再ビルドします。
- 発行者ポリシーファイルのみを GAC に配備します。 これにより、アプリケーションで特に除外されていない限り、コンピュータ上の Data Provider に対するすべての参照で、発行者ポリシーファイルの **redirect** が使用されます。

サンプルプロジェクト

Adaptive Server ADO.NET Data Provider には次の 3 つのサンプルプロジェクトが組み込まれています。

- Simple - データベースへ接続し、クエリを実行して、返された **resultsets** を読み込む方法を示すサンプルプログラム。
- TableViewer - `AseDataAdapter` オブジェクトを使用して結果を `DataGrid` コントロールにバインドする方法を示すサンプルプログラム。
- Advanced - 入力、出力、および入出力パラメータとともにストアドプロシージャを呼び出す方法を示すサンプルプログラム。ストアドプロシージャの戻り

値を読み取り、サポートされる 2 つのメカニズムを使用してパラメータを渡し、Data Provider のトレース機能を使用します。

ただし、Adaptive Server のデフォルトでは、Adaptive Server ADO.NET Data Provider のサンプルの実行に必要な `pubs2` データベースはインストールされません。`pub2` データベースをインストールする方法については、『ASE インストールガイド』を参照してください。

参照：

- サンプルアプリケーション (11 ページ)

サンプルアプリケーション

サンプルプログラムの実行には、pubs2 サンプルデータベースがインストールされた Adaptive Server へのアクセス権が必要です。さらに、Visual Studio .NET 2005 または .NET Framework 2.0、3.0、3.5、または 4.0 のいずれかがインストールされている必要もあります。

サンプルプログラムは、Adaptive Server ADO.NET Data Provider インストールディレクトリの次のディレクトリにあります。

- *Samples¥CSharp*。C# プログラミング言語で作成された 3 つのサンプルがあります。
- *Samples¥VB.NET*。Visual Basic .NET プログラミング言語で作成された 3 つのサンプルがあります。

デフォルトのインストールディレクトリは、Adaptive Server ADO.NET Data Provider の 32 ビット版では `C:¥Sybase¥DataAccess¥ADONET`、Adaptive Server ADO.NET Data Provider の 64 ビット版では `C:¥Sybase¥DataAccess64¥ADONET¥dll` です。

チュートリアル: Simple コードサンプルの使用

simple プロジェクトにはいくつかの機能があります。

- データベースへの接続
- `AseCommand` オブジェクトを使用したクエリの実行
- `AseDataReader` オブジェクトの使用
- 基本的なエラー処理

参照：

- Simple サンプルプロジェクトの理解 (13 ページ)

Visual Studio .NET での Simple コードサンプルの実行

Microsoft Visual Studio .NET で Simple コードサンプルを実行する手順を確認します。

1. Visual Studio .NET を起動します。
2. [ファイル] > [開く] > [プロジェクト] を選択します。

3. サンプルプロジェクトを指定します。

   C# の場合は、<インストールディレクトリ>¥Samples¥CSharp¥Simple に移動して、Simple.csproj を開きます。

   Visual Basic .NET の場合は、<インストールディレクトリ>¥Samples¥VB.NET¥Simple に移動して、Simple.vbproj を開きます。
4. インストールプログラムを使用して Adaptive Server ADO.NET Data Provider をインストールしている場合は、手順 7 に進みます。
5. インストールプログラムを使用しなかった場合は、プロジェクトの Adaptive Server ADO.NET Data Provider に対する参照を修正する必要があります。 これには、まず既存の参照を削除します。
   a) [ソリューション エクスプローラ] ウィンドウで、Simple プロジェクトが展開されていることを確認します。
   b) [参照設定] フォルダを展開します。
   c) Sybase.Data.AseClient を右クリックして、[削除]を選択します。
6. Adaptive Server ADO.NET Data Provider アセンブリに対する参照を追加します。
7. Simple サンプルを実行するには、[デバッグ] > [デバッグなしで開始] を選択するか、[Ctrl] キーを押しながら [F5] キーを押します。

   [AseSample] ダイアログボックスが表示されます。
8. [AseSample] ダイアログボックスで、サンプルの pubs2 データベースのある Adaptive Server への接続情報を指定して、[接続]をクリックします。

   アプリケーションがサンプルの pubs2 データベースに接続し、ダイアログボックスに各作成者の姓が表示されます。
9. ウィンドウの右上角にある[X]をクリックすると、アプリケーションが終了し、pubs2 データベースとの接続が切断されます。

これでアプリケーションを実行できました。 次の項では、アプリケーションコードについて説明します。

参照：

• Visual Studio .NET プロジェクトでの Adaptive Server ADO.NET Data Provider に対する参照の追加 (29 ページ)

Visual Studio を使用しない Simple サンプルプロジェクトの実行

Microsoft Visual Studio .を使用しないで simple サンプルプロジェクトを実行する手順を確認します。

1. DOS プロンプトを開き、<インストールディレクトリ>¥Samples の下位にある適切なサンプルディレクトリに移動します。
2. .NET Framework 2.0 バイナリのあるディレクトリをシステムパスに追加します。
3. Adaptive Server ADO.NET Data Provider インストールディレクトリにある dll ディレクトリが、システムパスと LIB 環境変数に含まれていることを確認します。

   デフォルトのインストールディレクトリは、Adaptive Server ADO.NET Data Provider の 32 ビット版では C:¥Sybase¥DataAccess¥ADONET¥dll、Adaptive Server ADO.NET Data Provider の 64 ビット版では C:¥Sybase¥DataAccess64¥ADONET¥dll です。
4. 提供されているビルドスクリプト build.bat を使用してサンプルプログラムをコンパイルします。
5. プログラムを実行するには、次のように入力します。

   ```
   simple.exe
   ```

   [AseSample] ダイアログボックスが表示されます。
6. [AseSample] ダイアログボックスで、サンプルの pubs2 データベースのある Adaptive Server への接続情報を指定して、[接続]をクリックします。

   アプリケーションがサンプルの pubs2 データベースに接続し、ダイアログボックスに各作成者の姓が表示されます。
7. ウィンドウの右上角にある[X]をクリックすると、アプリケーションが終了し、pubs2 データベースとの接続が切断されます。

Simple サンプルプロジェクトの理解

Adaptive Server サンプルデータベース pubs2 を使用する Simple コードサンプルに含まれるコードを利用して、Adaptive Server ADO.NET Data Provider の主要機能のいくつかを説明します。

pub2 データベースをインストールする方法については、『ASE インストールガイド』を参照してください。

コード全体を参照する場合は、サンプルプロジェクトを開いてください。

C# の場合:

```
<install dir>¥Samples¥CSharp¥Simple¥Simple.csproj
```

Visual Basic .NET の場合:

```
<install dir>¥Samples¥VB.NET¥Simple¥Simple.vbproj
```

インポートの宣言
プログラムの初めに、Adaptive Server ADO.NET Data Provider 情報をインポートする **import** 文を宣言します。

C# の場合:

```
using Sybase.Data.AseClient;
```

Visual Basic .NET の場合:

```
Imports Sybase.Data.AseClient
```

データベースへの接続
btnConnect_Click メソッドは、**new AseConnection** という接続オブジェクトを宣言して初期化します。

C# の場合:

```
AseConnection conn = new AseConnection(
   "Data Source='" + host +
      "';Port='" + port +
      "';UID='" + user +
      "';PWD='" + pass +
      "';Database='pubs2';" );
```

Visual Basic .NET の場合:

```
Dim conn As New AseConnection( _
     "Data Source='" + host + _
     "';Port='" + port + _
     "';UID='" + user + _
     "';PWD='" + pass + _
     "';Database='pubs2';")
```

`AseConnection` オブジェクトは、接続文字列を使用してサンプルデータベースに接続します。

C# の場合:

```
conn.Open();
```

Visual Basic .NET の場合:

```
conn.Open()
```

クエリの実行
次のコードは、Commandオブジェクト(AseCommand)によってSQL文を定義して実行します。 その後、DataReaderオブジェクト(AseDataReader)を返します。

C# の場合:

```
AseCommand cmd = new AseCommand( "select au_lname from
     authors", conn );
AseDataReader reader = cmd.ExecuteReader();
```

Visual Basic .NET の場合:

```
Dim cmd As New AseCommand( _
     "select au_lname from authors", conn)
Dim reader As AseDataReader  = cmd.ExecuteReader()
```

結果の表示
次のコードは、AseDataReaderオブジェクトに保持されているローをループしてListBoxコントロールに追加します。 DataReaderは、GetString(0)を使用して、ローの最初の値を取得します。

Read メソッドが呼び出されるたびに、DataReaderは結果セットから別のローを取得して返します。 読み込まれたそれぞれのローについて、新しい項目がListBoxに追加されます。

C# の場合:

```
listAuthors.BeginUpdate();
while( reader.Read() ) {
     listAuthors.Items.Add( reader.GetString( 0 ) );
}
listAuthors.EndUpdate();
```

Visual Basic .NET の場合:

```
listAuthors.BeginUpdate()
While reader.Read()
     listAuthors.Items.Add(reader.GetString(0))
End While
listAuthors.EndUpdate()
```

接続の終了
メソッドの最後にある次のコードで、読み込みオブジェクトと接続オブジェクトをクローズします。

C# の場合:

```
reader.Close();
conn.Close();
```

Visual Basic .NET の場合:

```
reader.Close()
conn.Close()
```

エラー処理
実行時に発生したエラーや Adaptive Server ADO.NET Data Provider オブジェクトのエラーはすべて、メッセージボックスに表示されます。 次のコードは、エラーを検出してメッセージを表示します。

C# の場合:

```
catch( AseException ex ) {
     MessageBox.Show( ex.Message );
}
```

Visual Basic .NET の場合:

```
Catch ex As AseException
     MessageBox.Show(ex.Message)
End Try
```

参照：

- AseConnection クラス (140 ページ)
- AseCommand クラス (123 ページ)
- AseDataReader クラス (168 ページ)
- AseException クラス (200 ページ)

チュートリアル: Table Viewer コードサンプルの使用

このチュートリアルは、Adaptive Server ADO.NET Data Provider に付属している Table Viewer プロジェクトに基づいています。

アプリケーション全体は、Adaptive Server ADO.NET Data Provider のインストールディレクトリにあります。

C# の場合:

```
 <install dir>¥Samples¥CSharp¥TableViewer¥TableViewer.csproj
```

Visual Basic .NET の場合:

```
<install dir>¥Samples¥VB.NET¥TableViewer¥TableViewer.vbproj
```

Table Viewer プロジェクトは、Simple プロジェクトより複雑です。 このサンプルでは、次の機能について説明します。

- データベースへの接続

- `AseDataAdapter` オブジェクトの使用
- 高度なエラー処理と結果チェック

参照：

- Table Viewer サンプルプロジェクトの理解 (19 ページ)

Visual Studio .NET での Table Viewer コードサンプルの実行

Microsoft Visual Studio .NET で Table Viewer コードサンプルプロジェクトを実行します。

1. Visual Studio .NET を起動します。
2. [ファイル] > [開く] > [プロジェクト] を選択します。
3. Adaptive Server ADO.NET Data Provider インストールディレクトリにある `Samples` ディレクトリを指定します。 `CSharp` または `VB.NET` ディレクトリに移動して、Table Viewer プロジェクトを開きます。
4. インストールプログラムを使用して Adaptive Server ADO.NET Data Provider をインストールしている場合は、手順 7 に進みます。
5. インストールプログラムを使用しなかった場合は、プロジェクトの Adaptive Server ADO.NET Data Provider に対する参照を修正する必要があります。 これには、まず既存の参照を削除します。
   a) [ソリューション エクスプローラ] ウィンドウで、Simple プロジェクトが展開されていることを確認します。
   b) [参照設定] フォルダを展開します。
   c) `Sybase.Data.AseClient` を右クリックして、[削除]を選択します。
6. Adaptive Server ADO.NET Data Provider アセンブリに対する参照を追加します。
7. TableViewer サンプルを実行するには、[デバッグ] > [デバッグなしで開始] を選択するか、[ctrl] キーを押しながら [F5] キーを押します。
8. [Table Viewer] ダイアログボックスで、`pubs2` サンプルデータベースがインストールされた Adaptive Server への接続情報を指定します。 [接続]をクリックします。

   アプリケーションが Adaptive Server `pubs2` サンプルデータベースに接続します。
9. [Table Viewer] ダイアログボックスで[実行]をクリックします。

   アプリケーションは、サンプルデータベースの authors テーブルからデータを取得して、クエリの結果を Results DataList に入力します。

このアプリケーションで、別の SQL 文を実行することもできます。 [SQL 文] ペインに SQL 文を入力して、[実行] をクリックします。

10. ウィンドウの右上角にある[X]をクリックすると、アプリケーションが終了し、サンプルデータベースとの接続が切断されます。

参照：

• Visual Studio .NET プロジェクトでの Adaptive Server ADO.NET Data Provider に対する参照の追加 (29 ページ)

Visual Studio を使用しない Table Viewer サンプルプロジェクトの実行

Microsoft Visual Studio を使用しないで Table Viewer サンプルプロジェクトを実行します。

1. DOS プロンプトを開いて、<インストールディレクトリ>¥Samples にある適切なサンプルディレクトリに移動します。
2. .NET Framework 2.0 バイナリのあるディレクトリをシステムパスに追加します。
3. Adaptive Server ADO.NET Data Provider インストールディレクトリにある dll ディレクトリが、システムパスと LIB 環境変数に含まれていることを確認します。

   デフォルトのインストールディレクトリは、Adaptive Server ADO.NET Data Provider の 32 ビット版では C:¥Sybase¥DataAccess¥ADONET¥dll、Adaptive Server ADO.NET Data Provider の 64 ビット版では C:¥Sybase¥DataAccess64¥ADONET¥dll です。
4. 提供されているビルドスクリプト build.bat を使用してサンプルプログラムをコンパイルします。
5. プログラムを実行するには、次のように入力します。

   ```
   tableviewer.exe
   ```
6. [Table Viewer] ダイアログボックスで、pubs2 サンプルデータベースがインストールされた Adaptive Server への接続情報を指定します。

   [接続]をクリックします。

   アプリケーションが Adaptive Server pubs2 サンプルデータベースに接続します。
7. [Table Viewer] ダイアログボックスで[実行]をクリックします。

   アプリケーションは、サンプルデータベースの authors テーブルからデータを取得して、クエリの結果を Results DataList に入力します。

このアプリケーションで、別の SQL 文を実行することもできます。 [SQL 文] ペインに SQL 文を入力して、[実行] をクリックします。

8. ウィンドウの右上角にある[X]をクリックすると、アプリケーションが終了し、サンプルデータベースとの接続が切断されます。

これでアプリケーションを実行できました。 次の項では、アプリケーションコードについて説明します。

Table Viewer サンプルプロジェクトの理解

Table Viewer コードサンプルを利用して、Adaptive Server ADO.NET Data Provider の主要機能の一部を説明します。

Table Viewer プロジェクトは、Adaptive Server サンプルデータベース `pubs2` を使用します。このデータベースは、Adaptive Server インストールディレクトリにあるスクリプトからインストールできます。

この項では、数行ずつコードを説明します。 コード全体を参照するには、Adaptive Server インストールディレクトリのサンプルプロジェクトを開きます。

C# の場合:

```
<install dir> ¥Samples¥CSharp¥TableViewer¥ TableViewer.csproj
```

Visual Basic .NET の場合:

```
<install dir>¥Samples¥VB.NET¥TableViewer ¥TableViewer.vbproj
```

インポートの宣言
プログラムの始めに、Adaptive Server ADO.NET Data Provider 情報をインポートする **import** 文を宣言します。

C# の場合:

```
using Sybase.Data.AseClient;
```

Visual Basic .NET の場合:

```
Imports Sybase.Data.AseClient
```

インスタンス変数の宣言
AseConnection クラスを使用して、**AseConnection** 型のインスタンス変数を宣言します。 この接続は、データベースへの初期接続と、[実行] をクリックしてデータベースから結果セットを取得するときに使用されます。

C# の場合:

```
private AseConnection
     _conn;
```

Visual Basic .NET の場合:

```
Private _conn As AseConnection
```

データベースへの接続
次のコードは、Connection String フィールドにデフォルトで表示される接続文字列のデフォルト値を設定します。

C# の場合:

```
txtConnectString.Text = "Data Source='" +
   System.Net.Dns.GetHostName() +
   "';Port='5000';UID='sa';PWD='';Database='pubs2';";
```

Visual Basic .NET の場合:

```
txtConnectString.Text = "Data Source='" + _
   System.Net.Dns.GetHostName() + _
   "';Port='5000';UID='sa';PWD='';Database='pubs2';"
```

`Connection` オブジェクトは、接続文字列を使用してサンプルデータベースに接続します。

C# の場合:

```
_conn = new AseConnection( txtConnectString.Text ); _conn.Open();
```

Visual Basic .NET の場合:

```
_conn = New AseConnection(txtConnectString.Text)
_conn.Open()
```

クエリの定義
次のコードは、SQL Statement フィールドに表示されるデフォルトのクエリを定義します。

C# の場合:

```
this.txtSQLStatement.Text = "SELECT * FROM authors";
```

Visual Basic .NET の場合:

```
Me.txtSQLStatement.Text = "SELECT * FROM authors"
```

結果の表示
アプリケーションは、`Connection` オブジェクトが初期化されているかどうかを確認してから、結果をフェッチします。 初期化されている場合は、接続ステータスがオープンであることを確認します。

C# の場合:

```
if( _conn == null || _conn.State != ConnectionState.Open )
{
```

```
     MessageBox.Show( "Connect to a database first.", "Not
connected" );
     return;
}
```

Visual Basic .NET の場合:

```
If (_conn Is Nothing) OrElse (_conn.State <> ConnectionState.Open)
Then
    MessageBox.Show("Connect to a database first.", "Not connected")
     Return
End If
```

データベースに接続されると、次のコードは DataAdapter オブジェクト (AseDataAdapter) を使用して SQL 文を実行します。 新しい DataSet オブジェクトが作成されて、DataAdapter オブジェクトの結果が入力されます。 最後に、DataSet の内容がウィンドウの DataGrid コントロールにバインドされます。

C# の場合:

```
using(AseCommand cmd = new AseCommand( txtSQLStatement.Text.Trim(),
_conn ))
{
     using(AseDataAdapter   da = new AseDataAdapter(cmd))
     {
          DataSet ds = new DataSet();
          da.Fill(ds, "Table");

          dgResults.DataSource = ds.Tables["Table"];
     }
}
```

Visual Basic .NET の場合:

```
Dim cmd As New AseCommand( _
   txtSQLStatement.Text.Trim(), _conn)
Dim da As New AseDataAdapter(cmd)
Dim ds As New DataSet
da.Fill(ds, "Table")
dgResults.DataSource = ds.Tables("Table")
```

グローバル変数を使用して接続を宣言しているため、SQL 文の実行には以前にオープンした接続が再使用されます。

エラー処理
アプリケーションがデータベースへの接続を試行しているときにエラーが発生した場合は、次のコードによってエラーが検出され、メッセージが表示されます。

C# の場合:

```
catch( AseException ex )
{
   MessageBox.Show( ex.Source + " : " + ex.Message +
```

```
      " ("+ ex.ToString() + ")",
      "Failed to connect" );
}
```

Visual Basic .NET の場合:

```
Catch ex As AseException
   MessageBox.Show(ex.Source + " : " + ex.Message + _
   "(" + ex.ToString() + ")" + _
   "Failed to connect")
End Try
```

参照：

- AseConnection クラス (140 ページ)
- AseDataAdapter クラス (158 ページ)
- AseConnection コンストラクタ (141 ページ)

チュートリアル: Advanced コードサンプルの使用

このチュートリアルは、Adaptive Server ADO.NET Data Provider に付属している Advanced プロジェクトに基づいています。

アプリケーション全体は、Adaptive Server ADO.NET Data Provider のインストールディレクトリにあります。

C# の場合:

```
<install dir>¥Samples¥CSharp¥Advanced¥Advanced.csproj
```

Visual Basic .NET の場合:

```
<install dir>¥Samples¥VB.NET¥Advanced¥Advanced.vbproj
```

Advanced プロジェクトでは、次の機能について説明します。

- データベースへの接続
- Adaptive Server ADO.NET Data Provider に対する ADO.NET の呼び出しをトレースするトレースイベント機能の使用
  トレースイベント機能を使用すると、実行した ADO.NET の呼び出しをすべてログに記録して、Sybase 製品の保守契約を結んでいるサポートセンタに送る詳細情報の収集やトラブルシューティングに利用できます。
- 名前付きパラメータ (**"@param"**) の使用
- 次のようなパラメータマーカ (**"?"**) の使用: `{? = call sp_hello(?, ?, ?)}`
- 入力パラメータ、入力/出力パラメータ、出力パラメータ、戻り値を使用したストアドプロシージャの呼び出し。 Adaptive Server では次の 2 つの方法でストアドプロシージャを呼び出すことができます。

- ストアドプロシージャ名を **CommandText** として使用して、**AseCommand.CommandType** を **CommandType.StoredProcedure** に設定する。
- ODBC および JDBC プログラムと互換性がある呼び出し構文を使用する。

Visual Studio .NET での Advanced コードサンプルの実行

Microsoft Visual Studio .NET で advanced コードサンプルプロジェクトを実行します。

1. Visual Studio .NET を起動します。
2. [ファイル] > [開く] > [プロジェクト] を選択します。
3. Adaptive Server ADO.NET Data Provider インストールディレクトリにある `Samples` ディレクトリを指定します。 `CSharp` または `VB.NET` ディレクトリに移動して、Advanced プロジェクトを開きます。
4. インストールプログラムを使用して Adaptive Server ADO.NET Data Provider をインストールしている場合は、手順 7 に進みます。
5. インストールプログラムを使用しなかった場合は、プロジェクトの Adaptive Server ADO.NET Data Provider に対する参照を修正する必要があります。 これには、まず既存の参照を削除します。
   a) [ソリューション エクスプローラ] ウィンドウで、Simple プロジェクトが展開されていることを確認します。
   b) [参照設定] フォルダを展開します。
   c) `Sybase.Data.AseClient` を右クリックして、[削除]を選択します。
6. Adaptive Server ADO.NET Data Provider アセンブリに対する参照を追加します。
7. [デバッグ] > [デバッグなしで開始]を選択して、Advanced プロジェクトを実行します。

   [Form1] ダイアログボックスが表示されます。
8. [Form1] ダイアログボックスで[接続]をクリックします。

   アプリケーションが Adaptive Server サンプルデータベースに接続します。
9. [Form1] ダイアログボックスで[実行]をクリックします。

   アプリケーションはストアドプロシージャを実行し、入力/出力パラメータ、出力パラメータ、戻り値を返します。
10. ウィンドウの右上角にある[X]をクリックすると、アプリケーションが終了し、サンプルデータベースとの接続が切断されます。

これでアプリケーションを実行できました。 次の項では、アプリケーションについて説明します。

Visual Studio を使用しない Advanced サンプルプロジェクトの実行

Microsoft Visual Studio を使用しないで advanced サンプルプロジェクトを実行します。

1. DOS プロンプトを開いて、<インストールディレクトリ>¥Samples ディレクトリにある適切なサンプルディレクトリに移動します。
2. .NET Framework 2.0 バイナリのあるディレクトリをシステムパスに追加します。
3. Adaptive Server ADO.NET Data Provider インストールディレクトリにある dll ディレクトリが、システムパスと LIB 環境変数に含まれていることを確認します。

   デフォルトのインストールディレクトリは、Adaptive Server ADO.NET Data Provider の 32 ビット版では C:¥Sybase¥DataAccess¥ADONET¥dll、Adaptive Server ADO.NET Data Provider の 64 ビット版では C:¥Sybase¥DataAccess64¥ADONET¥dll です。
4. 提供されているビルドスクリプト build.bat を使用してサンプルプログラムをコンパイルします。
5. 次のように入力して、プログラムを実行します。

   ```
   advanced.exe
   ```
6. [Form1] ダイアログボックスが表示されます。 [接続]をクリックします。

   アプリケーションが Adaptive Server サンプルデータベースに接続します。
7. [Form1] ダイアログボックスで[実行]をクリックします。

   アプリケーションはストアドプロシージャを実行し、入力/出力パラメータ、出力パラメータ、戻り値を返します。
8. ウィンドウの右上角にある[X]をクリックすると、アプリケーションが終了し、サンプルデータベースとの接続が切断されます。

これでアプリケーションを実行できました。 次の項では、アプリケーションコードについて説明します。

Advanced サンプルプロジェクトの理解

Advanced コードサンプルを利用して、Adaptive Server ADO.NET Data Provider の主要機能の一部を説明します。 Advanced プロジェクトは、Adaptive Server サンプルデータベース pubs2 を使用します。このデータベースは、Adaptive Server の CD からインストールできます。

コードは数行ずつ説明します。 コード全体を参照する場合は、サンプルプロジェクトを開いてください。

C# の場合:

```
<install dir>¥Samples¥CSharp¥Advanced¥Advanced.csproj
```

Visual Basic .NET の場合:

```
<install dir>¥Samples¥VB.NET¥Advanced¥Advanced.vbproj
```

トレースイベントハンドラの付加
次のコード行は、トレースイベントハンドラを `AseConnection` に付加します。

C# の場合:

```
_conn.TraceEnter += new
     TraceEnterEventHandler(TraceEnter);
_conn.TraceExit += new
     TraceExitEventHandler(TraceExit);
```

Visual Basic .NET の場合:

```
AddHandler _conn.TraceEnter, AddressOf TraceEnter
AddHandler _conn.TraceExit, AddressOf TraceExit
```

名前付きパラメータを使用したストアドプロシージャの呼び出し
メソッド **ExecuteCommandUsingNamedParams()** は、名前付きパラメータを使用して、名前でストアドプロシージャを呼び出します。

C# の場合:

```
using(AseCommand cmd = new AseCommand("sp_hello", _conn))
{
   cmd.CommandType = CommandType.StoredProcedure;

   AseParameter inParam = new AseParameter("@inParam",
AseDbType.VarChar, 32);
   inParam.Direction = ParameterDirection.Input;
   inParam.Value = textBoxInput.Text;
   cmd.Parameters.Add(inParam);

   AseParameter inoutParam = new AseParameter("@inoutParam",
   AseDbType.VarChar, 64);
   inoutParam.Direction = ParameterDirection.InputOutput;
   inoutParam.Value = textBoxInOut.Text;
   cmd.Parameters.Add(inoutParam);

   AseParameter outParam = new AseParameter("@outParam",
   AseDbType.VarChar, 64);
   outParam.Direction = ParameterDirection.Output;
   cmd.Parameters.Add(outParam);

   AseParameter retValue = new AseParameter("@retValue",
AseDbType.Integer);
   retValue.Direction = ParameterDirection.ReturnValue;
   cmd.Parameters.Add(retValue);
```

```
   try
   {
      cmd.ExecuteNonQuery();
   }
   catch (AseException ex)
   {
      MessageBox.Show(ex.Source + " : " + ex.Message + " (" +
ex.ToString() +
      ")", "Execute Stored Precedure failed.");
   }
}
```

Visual Basic .NET の場合:

```
Dim cmd As New AseCommand("sp_hello", _conn)
' set command type to stored procedure
cmd.CommandType = CommandType.StoredProcedure

' create the input parameter object and bind it to the command
Dim inParam As New AseParameter("@inParam", AseDbType.VarChar, 32)
inParam.Direction = ParameterDirection.Input
inParam.Value = textBoxInput.Text
cmd.Parameters.Add(inParam)

' create the inout parameter object and bind it to the command
Dim inoutParam As New AseParameter("@inoutParam", AseDbType.VarChar,
64)
inoutParam.Direction = ParameterDirection.InputOutput
inoutParam.Value = textBoxInOut.Text
cmd.Parameters.Add(inoutParam)

' create the output parameter object and bind it to the command
Dim outParam As New AseParameter("@outParam", AseDbType.VarChar, 64)
outParam.Direction = ParameterDirection.Output
cmd.Parameters.Add(outParam)

' create the return value object and bind it to the command
Dim retValue As New AseParameter("@retValue", AseDbType.Integer)
retValue.Direction = ParameterDirection.ReturnValue
cmd.Parameters.Add(retValue)

' execute the stored procedure
Try
   cmd.ExecuteNonQuery()

Catch ex As AseException
   MessageBox.Show(ex.Source + " : " + ex.Message + " (" +
ex.ToString() + ")",
      "Execute Query failed.")
Finally
   ' dispose the command object
   cmd.Dispose()
End Try
```

呼び出し構文とパラメータマーカを使用したストアドプロシージャの呼び出し
メソッド **ExecuteCommandUsingParameterMarkers()** は、呼び出し構文とパラメータマーカを使用してストアドプロシージャを呼び出します。

C# の場合:

```
using(AseCommand cmd = new AseCommand("{ ? = call
sp_hello(?, ?, ?)}", _conn))
{
   cmd.NamedParameters = false;
   AseParameter retValue = new AseParameter(0, AseDbType.Integer);
   retValue.Direction = ParameterDirection.ReturnValue;
   cmd.Parameters.Add(retValue);

  AseParameter inParam = new AseParameter(1, AseDbType.VarChar, 32);
   inParam.Direction = ParameterDirection.Input;
   inParam.Value = textBoxInput.Text;
   cmd.Parameters.Add(inParam);

   AseParameter inoutParam = new AseParameter(2, AseDbType.VarChar,
64);
   inoutParam.Direction = ParameterDirection.InputOutput;
   inoutParam.Value = textBoxInOut.Text;
   cmd.Parameters.Add(inoutParam);

   AseParameter outParam = new AseParameter(3, AseDbType.VarChar,
64);
   outParam.Direction = ParameterDirection.Output;
   cmd.Parameters.Add(outParam);
   try
   {
      cmd.ExecuteNonQuery();
   }
   catch (AseException ex)
   {
      MessageBox.Show(ex.Source + " : " + ex.Message + " (" +
ex.ToString() +
      ")", "Execute Stored Precedure failed.");
   }
}
```

Visual Basic .NET の場合:

```
Dim cmd As New AseCommand("{ ? = call sp_hello(?, ?, ?)}", _conn)
' need to notify Named Parameters are not being used (which is the
default)
cmd.NamedParameters = False

' create the return value object and bind it to the command
Dim retValue As New AseParameter(0, AseDbType.Integer)
retValue.Direction = ParameterDirection.ReturnValue
cmd.Parameters.Add(retValue)

' create the input parameter object and bind it to the command
```

```
Dim inParam As New AseParameter(1, AseDbType.VarChar, 32)
inParam.Direction = ParameterDirection.Input
inParam.Value = textBoxInput.Text
cmd.Parameters.Add(inParam)

' create the inout parameter object and bind it to the command
Dim inoutParam As New AseParameter(2, AseDbType.VarChar, 64)
inoutParam.Direction = ParameterDirection.InputOutput
inoutParam.Value = textBoxInOut.Text
cmd.Parameters.Add(inoutParam)

' create the output parameter object and bind it to the command
Dim outParam As New AseParameter(3, AseDbType.VarChar, 64)
outParam.Direction = ParameterDirection.Output
cmd.Parameters.Add(outParam)

' execute the stored procedure
Try
   cmd.ExecuteNonQuery()

   ' get the output, inout and return values and display them
   textBoxReturn.Text = cmd.Parameters(0).Value
   textBoxReturn.ForeColor = Color.Blue

   textBoxInOut.Text = cmd.Parameters(2).V

   textBoxOutput.Text = cmd.Parameters(3).Value
   textBoxOutput.ForeColor = Color.Blue
Catch ex As AseException
   MessageBox.Show(ex.Source + " : " + ex.Message + " (" +
ex.ToString() + ")",_
   "Execute Query Failed")
Finally
   ' dispose the command object
   cmd.Dispose()
End Try
```

アプリケーションの開発

Adaptive Server ADO.NET Data Provider を使用したアプリケーションの開発と配備の方法について説明します。

Visual Studio .NET プロジェクトでの Data Provider の使用

Adaptive Server ADO .NET Data Provider のインストール後に必要となる Visual Studio .NET プロジェクトの変更を確認します。

1. Adaptive Server ADO.NET Data Provider アセンブリに対する参照を追加します。
2. ソースコードに Adaptive Server ADO.NET Data Provider クラスを参照する行を追加します。

参照：

- Adaptive Server ADO.NET Data Provider 配備の要件 (1 ページ)

Visual Studio .NET プロジェクトでの Adaptive Server ADO.NET Data Provider に対する参照の追加

参照を追加して、Adaptive Server ADO.NET Data Provider のコードを検索するために含めるアセンブリを Visual Studio.NET に指示します。

1. Visual Studio .NET を起動して、プロジェクトを開きます。
2. [ソリューション エクスプローラ] ウィンドウで、[参照設定] フォルダを右クリックし、ポップアップメニューから、[参照の追加]を選択します。

   [参照の追加] ダイアログボックスが表示されます。
3. [.NET] タブで、`Sybase.AdoNet2.AseClient` または `Sybase.AdoNet4.AseClient` コンポーネントが見つかるまでコンポーネントの一覧をスクロールします。 このコンポーネントを指定して、[選択]をクリックします。
4. [OK] をクリックします。

   コンポーネントの一覧に Adaptive Server ADO.NET Data Provider アセンブリがない場合は、[参照] から <インストールディレクトリ>¥`dll` ディレクトリにある `Sybase.AdoNet2.AseClient.dll` または `Sybase.AdoNet4.AseClient.dll` を探します。 この DLL を選択して、[開く]、[OK] の順にクリックします。

注意： デフォルトのロケーションは、Adaptive Server ADO.NET Data Provider の 32 ビット版では `C:¥Sybase¥DataAccess¥ADONET¥dll`、Adaptive Server ADO.NET Data Provider の 64 ビット版では `C:¥Sybase ¥DataAccess64¥ADONET¥dll` です。

プロジェクトの [ソリューション エクスプローラ] ウィンドウの [参照設定] フォルダにアセンブリが追加されます。

コードでの Adaptive Server ADO.NET Data Provider クラスの参照

ソースコードに Adaptive Server ADO.NET Data Provider を参照する行を追加して、使用できるようにします。 C# と Visual Basic .NET では追加する行が異なります。

1. Visual Studio .NET を起動します。
2. プロジェクトを開きます。
   - C# の場合は、プロジェクトの先頭にある `using` ディレクティブの一覧に次の行を追加します。
     ```
     using Sybase.Data.AseClient;
     ```
   - Visual Basic .NET の場合は、プロジェクトの先頭にある行 `Public Class Form1` の前に次の行を追加します。
     ```
     Imports Sybase.Data.AseClient
     ```
     この行は絶対に必須というわけではありません。 ただし、これによって Adaptive Server クラスの省略形を使用できるようになります。 この行がない場合も、コード内で次のコードを使用することができます。
     ```
     Sybase.Data.AseClient.AseConnection conn = new
     Sybase.Data.AseClient.AseConnection();
     ```
     この行を次の代わりに
     ```
     AseConnection conn = new AseConnection();
     ```
     コード内で使用します。

Adaptive Server データベースへの接続

Adaptive Server データベースに接続するためのコードを作成します。 操作を実行するには、アプリケーションをデータベースに接続する必要があります。

1. `AseConnection` オブジェクトを割り付けます。

   次のコードは、「conn」という名前の `AseConnection` オブジェクトを作成します。

   C# の場合:

```
AseConnection conn = new AseConnection();
```

Visual Basic .NET の場合:

```
Dim conn As New AseConnection()
```

アプリケーションからデータベースへ複数の接続を設定できます。 アプリケーションによっては、Adaptive Server データベースに対する接続を 1 つだけ使用して、常時この接続をオープンにします。 これを実行するには、接続にグローバル変数を宣言します。

C# の場合:

```
private AseConnection_conn;
```

Visual Basic .NET の場合:

```
Private _conn As AseConnection
```

詳細については、<インストールディレクトリ>¥Samples にある Table Viewer サンプル、および「Table Viewer サンプルプロジェクトの理解」を参照してください。

2. データベースへの接続に使用する接続文字列を指定します。

   C# の場合:

   ```
   AseConnection conn = new AseConnection(
      "Data Source='mango';Port=5000;" +
      "UID='sa';PWD='';" +
      "Database='pubs2';" );
   ```

   「mango」には、データベースサーバが実行されているホスト名を指定します。

   Visual Basic .NET の場合:

   ```
   Dim conn As New AseConnection(_
        "Data Source='mango',Port=5000," +_
        "UID='sa';PWD='''';" + _
        "Database='pubs2';")
   ```

3. 次のコードを使用して、データベースへの接続をオープンします。

   C# の場合:

   ```
   conn.Open();
   ```

   Visual Basic .NET の場合:

   ```
   conn.Open()
   ```

4. 接続エラーを検出します。

   データベースへの接続試行時に発生したすべてのエラーが検出されるようにアプリケーションを設計してください。 次のコードは、エラーを検出し、そのメッセージを表示する方法を示しています。

C# の場合:

```
try {
   _conn = new AseConnection(
      txtConnectString.Text );
   _conn.Open();
}
catch( AseException ex ) {
   MessageBox.Show(
      ex.Message,
      "Failed to connect");
}
```

Visual Basic .NET の場合:

```
Try
   _conn = New AseConnection(_
      txtConnectionString.Text)
   _conn.Open()
Catch ex As AseException
   MessageBox.Show(_
      ex.Message,_
      "Failed to connect")
End Try
```

`AseConnection` オブジェクトの作成時に接続文字列を渡すのではなく、`ConnectionString` プロパティを使用して接続文字列を設定することもできます。

C# の場合:

```
AseConnection conn = new AseConnection();
conn.ConnectionString = "Data Source='mango';" +
   "Port=5000;" +
   "UID='sa';" +
   "PWD='';" +
   "Database='pubs2';" ;
```

Visual Basic .NET の場合:

```
Dim conn As New AseConnection()
conn.ConnectionString = "Data Source='mango';" + _
   "Port=5000;" + _
   "UID='sa';" + _
   "PWD='';" + _
   "Database='pubs2';"
```

「mango」には、データベースサーバ名を指定します。

5. データベースへの接続をクローズします。`conn.Close()` メソッドを使用して明示的にクローズするまで、データベースへの接続はオープンしたままになります。

参照：

- Table Viewer サンプルプロジェクトの理解 (19 ページ)

- AseConnection クラス (140 ページ)
- ConnectionString プロパティ (151 ページ)
- AseConnection コンストラクタ (141 ページ)

接続プール

Adaptive Server Enterprise ADO.NET プロバイダは、アプリケーションがプールから既存の接続を再使用できるようにする接続プールをサポートします。

これにより、データベースへの新しい接続を繰り返し作成する代わりに、接続ハンドルをプールに保存し、接続を再使用できるようになります。 接続プールは、デフォルトでオンに設定されています。

プールサイズの最小値と最大値を指定することもできます。 次に例を示します。

```
"Data Source='mango';" +
   "Port=5000;" +
   "UID='sa';" +
   "PWD='';" +
   "Database='pubs2';" +
   "Max Pool Size=50;" +
   "Min Pool Size=5";
```

データベースに初めて接続しようとするときにアプリケーションは、指定した接続パラメータと同じパラメータを使用している既存の接続がプールにあるかどうかをチェックします。 一致する接続があった場合は、その接続を使用します。 一致する接続がなかった場合は、新しい接続を使用します。 接続が切断されると、その接続はプールに戻されて再利用できるようになります。

注意： Max Pool Size が指定されている場合は、オープンできる接続の最大数がこの値に制限されます。 制限値に達すると、**AseConnection.Open()** の呼び出しが失敗して **AseException** が発生します。

接続プールの無効化
接続プールを無効にするには、接続文字列で **Pooling=False** を指定します。

接続ステータスの確認

データベースへの接続が確立した後は、接続ステータスをチェックして接続がオープンになっていることを確認してから、データをフェッチして更新できます。

接続が失われたり、ビジー状態だったり、別のコマンドを処理中の場合は、それに相当するメッセージを返すことができます。

`AseConnection` クラスには、接続ステータスをチェックする「State プロパティ」があります。 使用可能なステータス値は Open と Closed です。

次のコードは、Connection オブジェクトが初期化されているかどうかをチェックし、初期化されている場合は、接続がオープンになっていることを確認します。

C# の場合:

```
if( _conn == null || _conn.State !=
   ConnectionState.Open )
   {
      MessageBox.Show( "Connect to a database first",
      "Not connected" );
      return;
   }
```

Visual Basic .NET の場合:

```
If (_conn Is Nothing) OrElse (_conn.State <> ConnectionState.Open)
Then
     MessageBox.Show("Connection to a database first",
     "Error")
     Return
End If
```

接続がオープンになっていない場合は、メッセージが返されます。

参照：

- State プロパティ (155 ページ)

文字セット

Adaptive Server ADO.NET Data Provider は、ネゴシエートされた文字セットを使用して Adaptive Server と通信します。この文字セットは、Adaptive Server ADO.NET Data Provider のユーザインタフェースで ClientCharset または Server Default に設定できます。 Server Default がデフォルト設定です。

使用法

- ClientCharset を文字セットとして選択する場合は、CodePageType プロパティの値も指定します。 有効な値は、ANSI (デフォルト値) および OEM です。
- Other を選択する場合、charset プロパティの値には文字セットの名前を指定します。 たとえば、ネゴシエートされた文字セットを utf8 に設定するには、次のように指定します。

  ```
  charset=utf8
  ```
- charset プロパティは、できるだけ使用しないでください。 .NET 文字列は unicode であり、マルチバイトに変換してから Adaptive Server に char または varchar パラメータとして送信する必要があります。 このため、サーバのデフォルト以外の文字セットを使用した場合、パフォーマンスに影響します。
- サポートされていない文字セットを使用すると、次のようなエラーが発生します。

```
[Sybase][driver] Could not load code page for requested charset
```

このエラーを回避するには、次のいずれかの手順を実行します。

- ドライバのユーザインタフェースで、[詳細設定] タブを開き、ClientCharset を文字セットとして選択します。 Other を選択した場合、指定した文字セットがサポートされることを確認してください。
- 接続文字列では、charset プロパティで、サポートされている文字セットが指定されていることを確認してください。

DDEX Provider for Adaptive Server

Data Designer Extensibility (DDEX) Provider for Adaptive Server を使用すると、サーバエクスプローラなどの Visual Studio コンポーネントが、Adaptive Server ADO.NET Data Provider を介して Adaptive Server およびそのオブジェクトと対話できるようになります。

DDEX Provider for Adaptive Server により、次の操作が可能になります。

- Visual Studio から Adaptive Server に接続してログインする
- Visual Studio のサーバエクスプローラに Adaptive Server オブジェクトを階層ノードとして表示する
- Adaptive Server のテーブルとビューをサーバエクスプローラからデータデザイナにドラッグアンドドロップする

DDEX Provider for Adaptive Server は、Visual Studio 2005、2008、および 2010 と互換性があります。

Adaptive Server との接続

Visual Studio のサーバエクスプローラビューを使用して、Adaptive Server へのデータベース接続を追加します。

前提条件

- SDK をまだインストールしていない場合は、グローバルアセンブリキャッシュ (GAC) にドライバを追加します。
- 使用しているアプリケーションで以前のバージョンの Adaptive Server ADO.NET Data Provider を参照している場合は、次のコマンドを実行します。
  - Visual Studio 2005 および 2008 の場合:

```
AseGacUtility -i
   policy.2.157.Sybase.AdoNet2.AseClient.dll
```

```
AseGacUtility -i
   policy.2.157.Sybase.AdoNet4.AseClient.dll
```

- Visual Studio 2010 の場合:

```
AseGacUtility4 -i
   policy.2.157.Sybase.AdoNet2.AseClient.dll
```

```
AseGacUtility4 -i
   policy.4.157.Sybase.AdoNet4.AseClient.dll
```

- Visual Studio 2005 および 2008 の場合は **AdoNetRegistrar** を、Visual Studio 2010 の場合は **AdoNetRegistrar4** を実行します。

手順

1. サーバエクスプローラビューで、[データ接続]を右クリックします。
2. [接続の追加]を選択します。
3. データソースとして [Sybase ASE Database] を選択し、データプロバイダとして [.NET Framework Data Provider for Sybase ASE] を選択します。
4. Adaptive Server のその他の接続プロパティを入力します。
   - 接続プロパティは、セミコロン (;) で区切ったリストとして指定することができます。
   - 最後の接続プロパティの末尾にセミコロン (;) を付ける必要はありません。
   - 値のないプロパティは無視されます。

   現時点では、誤った接続の指定にフラグを付ける警告メッセージまたはエラーメッセージはありません。

Adaptive Server オブジェクトの表示

Visual Studio のサーバエクスプローラで、Adaptive Server とその関連情報を表示します。

前提条件

このタスクの実行には、アクティブな Adaptive Server との有効な接続が必要です。

手順

1. Adaptive Server に接続します。
2. Adaptive Server オブジェクトを展開します。
3. 任意の Adaptive Server オブジェクトをクリックして、プロパティ情報を表示します。

サポートされている Adaptive Server オブジェクト

DDEX Provider for Adaptive Server は、Adaptive Server オブジェクトを公開します。公開された Adaptive Server オブジェクトは Visual Studio のサーバエクスプローラに表示されるため、エクスプローラでオブジェクトにアクセスすることができます。

データベースオブジェクト	プロパティ
検査制約	Name、Catalog、Schema、Table、Constraint Column、Constraint Type
テーブルとしての Web サービスのカラム	Name、Catalog、Schema、Table、Default Value、Is Nullable、Ordinal、Length、Precision、Scale、System Type
データベース	Name、Create Date、Dump Date
データ型	Name
デフォルト	Name、Catalog、Schema
外部キー	Name、Catalog、Schema、Table、Referenced Table、Referenced Table Catalog、Referenced Table Column、Referenced Table Schema
インデックス	Name、Catalog、Schema、Table、Referred Column
instead-of トリガ	Name、Catalog、Schema、View
プライマリキー	Name、Catalog、Schema、Table、Referred Column
プロキシテーブル	Name、Catalog、Schema
プロキシテーブルカラム	Name、Catalog、Schema、Table、Default Value、Is Nullable、Ordinal、Length、Precision、Scale、System Type
ルール	Name、Catalog、Schema
スキーマ	Name
ストアドプロシージャ	Name、Catalog、Schema
ストアドプロシージャパラメータ	Name、Catalog、Schema、Stored Procedure、Is Output、Ordinal、Length、Precision、Scale、System Type
テーブル	Name、Catalog、Schema、Type

データベースオブジェクト	プロパティ
テーブルカラム	Name、Catalog、Schema、Table、Default Value、Is Nullable、Ordinal、Length、Precision、Scale、System Type
トリガ	Name、Catalog、Schema、Table
一意性制約	Name、Catalog、Schema、Table、Referred Column
ユーザ定義関数	Name、Catalog、Schema
ユーザ定義関数パラメータ	Name、Catalog、Schema、User-defined Function、Is Output、Ordinal、Length、Precision、Scale、System Type
ユーザ定義データ型	Name、Catalog、Schema
ビュー	Name、Catalog、Schema
ビューカラム	Name、Catalog、Schema、View、Default Value、Is Nullable、Ordinal、Length、Precision、Scale、System Type
テーブルとしての Web サービス	Name、Catalog、Schema、Method、Timeout、WSDL URI

拡張 DDEX Provider for Adaptive Server

Entity Framework を使用して、Adaptive Server データベースでデータモデルを作成することができます。 この拡張は Visual Studio 2008 SP1 および Visual Studio 2010 でサポートされています。

Entity Framework を使用した Adaptive Server エンティティデータモデルの作成

Microsoft Entity Framework を使用して、データアクセスクラスを作成します。

前提条件

このタスクの実行に必要な、アクティブな Adaptive Server との有効な接続が存在することを確認します。

手順

1. 新しいアプリケーションプロジェクトを作成します。
2. [ソリューション エクスプローラ] ウィンドウで、プロジェクトを右クリックして、[追加] > [新しいアイテム] を選択します。

3. カテゴリとして[データ] を選択し、テンプレートとして [ADO.NET エンティティ データ モデル] を選択します。
4. エンティティデータモデルの名前を入力して[追加] をクリックします。

   Data Model ウィザードが起動します。
5. [データベースから生成] を選択します。 [次へ] をクリックします。
6. 既存の Adaptive Server 接続を選択するか、または[新しい接続] をクリックします。 [新しい接続] を選択した場合は、接続設定の名前を入力します。 [次へ] をクリックします。
7. Adaptive Server オブジェクトを選択します。 [完了] をクリックします。

エンティティデザイナを使用して、モデルを変更します。 モデルを保存すると、Entity Framework によってデータアクセスクラスが自動的に生成されます。

Adaptive Server ADO.NET Data Provider での SSIS のサポート

Adaptive Server ADO.NET Data Provider を SQL Server Integration Services (SSIS) に統合することができます。それにより、ADO.NET Data Provider 関数へのネイティブアクセスが可能になります。

この統合では、Adaptive Server を次の役割で使用することができます。

- ADO.NET 接続マネージャ
- ADO.NET 変換元データフローコンポーネント
- ADO.NET 変換先データフローコンポーネント

拡張 Adaptive Server ADO.NET Data Provider は、SSIS 2008 および SSIS 2012 をサポートします。 SSIS のサポートは、DDEX Provider for Adaptive Server に依存しています。SSIS を使用するには、DDEX Provider for Adaptive Server がインストールされている必要があります。

Adaptive Server 接続の設定

Adaptive Server 接続を設定します。

前提条件

- Sybase ADO.NET Data Provider が含まれる SDK をインストールします。
- SDK をまだインストールしていない場合は、次のようにグローバルアセンブリキャッシュ (GAC) にドライバを追加します。

```
AseGacUtility -i Sybase.AdoNet2.AseClient.dll
```

```
AseGacUtility4 -i Sybase.AdoNet4.AseClient.dll
```

その後、**AdoNetRegistrar** を実行します。

手順

1. [データ フロー] タブで、設定する ADO NET 変換元/変換先コンポーネントを右クリックして[編集] を選択します。
2. 接続マネージャの横にある[新規作成] ボタンをクリックします。
3. [ADO.NET の接続マネージャの構成] ウィンドウで[新規作成] をクリックします。
4. 表示されたプロバイダのリストから、[Sybase Adaptive Server Enterprise Data Provider] を選択します。
5. 適切な接続プロパティを入力します。

   Adaptive Server ADO.NET を SSIS とともに使用するには、[QuotedIdentifier] を 1 に設定します。
6. [OK] をクリックします。

注意： デフォルトでは、ADO.NET 変換先コンポーネントは、実行する insert コマンドをバッチ処理します。 現在のところ、SSIS を使用して Adaptive Server へのバッチアップロードを実行するよりも、単純な insert コマンドを実行するほうが高速です。 変換先コンポーネントで単純な insert コマンドを実行するように設定するには、BatchSize プロパティを 1 に設定します。

SQL Server 2008 で Adaptive Server へのデータ転送を高速化する SSIS Custom Data Flow Destination コンポーネント

Adaptive Server ADO.NET Data Provider のディストリビューションには SQL Server Integration Services (SSIS) Custom Data Flow Destination コンポーネントが含まれています。これは、Adaptive Server の変換先へのデータ転送を高速化します。

この高速データ転送は、**AseBulkCopy** クラスでサポートされる Adaptive Server のバルク挿入プロトコルを使用します。 この Sybase.AdoNet2.AseDestination.dll という名前のコンポーネントは、Adaptive Server ADO.NET Data Provider およびアセンブリファイルとともに、32 ビットシステムでは `%SYBASE%¥DataAccess¥ADONET¥` に、64 ビットシステムでは `%SYBASE%¥DataAccess64¥ADONET¥` にインストールされます。

SQL Server 2008 の Adaptive Server ADO.NET Destination SSIS コンポーネントの設定

Adaptive Server ADO.NET Destination SSIS コンポーネントを設定します。

1. `Sybase.AdoNet2.AseDestination.dll` を `C:¥Program Files ¥Microsoft SQL Server¥100¥DTS¥PipelineComponents` および `C:`

¥Program Files (x86)¥Microsoft SQL Server¥100¥DTS ¥PipelineComponents にコピーします。

2. ローカルドライブの Microsoft SQL Server ディレクトリのいずれかから、SDK インストールで提供される AseGacUtility を使用して Sybase.AdoNet2.AseDestination.dll アセンブリを登録します。
3. SQL Server Business Intelligence Studio を起動します。
4. [ツールボックス] タブで、[データフローの変換先] タブを右クリックし、[アイテムの選択] を選択します。
   [ツールボックスアイテムの選択] ウィンドウが表示されます。
5. [SSIS データフロー項目] タブを選択します。 [Sybase Adaptive Server Enterprise ADO NET Destination]、[OK] の順にクリックします。 [ツールボックス] > [データフローの変換先] を選択して、Sybase Adaptive Server ADO NET Destination コンポーネントを確認します。
6. SSIS プロジェクトを作成するには、メニューから [ファイル] > [新規作成] > [プロジェクト] > [Integration Services プロジェクト] を選択します。 制御フローオブジェクトを作成するか、[制御フロー項目] ツールボックスからドラッグアンドドロップします。
7. [データフローの変換先] および [データフローの変換元ツールボックス] タブから、[Sybase Adaptive Server ADO NET Destination Component] と [ADO NET Source Component] を [データフロー] タブにドラッグアンドドロップします。
8. [接続マネージャー] ウィンドウで使用可能な変換元または変換先がない場合は、[接続マネージャー] ウィンドウを右クリックして、[新しい ADO.NET 接続] を選択します。 既存のデータ接続を選択するか、[新規] をクリックします。
9. 変換先 Adaptive Server への接続を新たに作成するには、[ADO.NET の接続マネージャーの構成] ウィンドウで、[新規] ボタンをクリックして、[Sybase Adaptive Server Enterprise Data Provider] を選択します。
10. [接続マネージャー] ウィンドウに、接続のプロパティを入力します。
11. バルク挿入を有効にするには、[追加の接続プロパティ] テキストボックスに次のように入力します。 enablebulkload=1

    注意： バルク挿入機能使用の詳細は、『Adaptive Server Enterprise ADO.NET Data Provider ユーザーズガイド』の「**AseBulkCopy**」を参照してください。
12. [OK] をクリックします。
13. データフローの ADO.NET 変換元に、接続とデータアクセスモードを設定します。 ADO.NET 変換元からデータフローパスに接続したら、Sybase Adaptive Server ADO NET Destination Component を右クリックして、[高度な編集の表示] を選択します。

14. [接続マネージャー] タブの [接続マネージャー] フィールドから ASE の接続を選択します。 [コンポーネントのプロパティ] タブで、TableName プロパティを変換先のテーブル名に設定します。
15. [入力列] タブを選択し、[名前] チェックボックスをオンにします。 これで、変換元テーブルで指定されたすべてのカラムが選択されます。
16. [OK] をクリックします。

注意： SQL Server 2008 からのデータ転送用 SSIS 変換先コンポーネントは、`Sybase.AdaptiveServerAdoNetDestination.dll` から `Sybase.AdoNet2.AseDestination.dll` に名前が変更されています。

接続が確立されます。 データ転送の詳細については、 Microsoft SSIS の文書を参照してください。

SQL Server 2012 で Adaptive Server へのデータ転送を高速化する SSIS Custom Data Flow Destination コンポーネント

Adaptive Server ADO.NET Data Provider のディストリビューションには SQL Server Integration Services (SSIS) Custom Data Flow Destination コンポーネントが含まれています。これは、SQLServer 2012 と互換性があり、バルク挿入プロトコルを使用して Adaptive Server Enterprise へのデータ転送を高速化します。

このカスタムデータフロー変換先コンポーネントは、`AseBulkCopy` クラスでサポートされる Adaptive Server のバルク挿入プロトコルを使用します。 この `Sybase.AdoNet4.AseDestination.dll` という名前のコンポーネントは、Adaptive Server ADO.NET Data Provider とともにインストールされ、32 ビットシステムでは `%SYBASE%¥DataAccess¥ADONET¥dll` に、64 ビットシステムでは `%SYBASE%¥DataAccess64¥ADONET¥dll` にインストールされます。

Adaptive Server ADO.NET Destination SSIS コンポーネントの設定

Adaptive Server ADO.NET Destination SSIS コンポーネントは、Adaptive Server の変換先へのデータ転送を迅速に行います。

1. `Sybase.AdoNet4.AseDestination.dll` を `C:¥Program Files ¥Microsoft SQL Server¥110¥DTS¥PipelineComponents` および `C:¥Program Files (x86)¥Microsoft SQL Server¥110¥DTS ¥PipelineComponents` にコピーします。
2. ステップ 1 で使用したローカルドライブの Microsoft SQL Server ディレクトリのいずれかから、SDK インストールで提供される `AseGacUtility4.exe` を使用して `Sybase.AdoNet4.AseDestination.dll` を登録します。

3. Windows で SQLServer 2012 Data Tools または SQL Server 2012 Data を起動するには、[スタート] > [すべてのプログラム] > [Microsoft SQL Server 2012] > [SQL Server Data Tools] を選択します。
4. [ファイル] > [新規作成] > [プロジェクト] > [Integration Services プロジェクト] を選択します。
   [SSIS] ツールボックスに Sybase 変換先コンポーネントが自動的に表示されます。
5. [制御フロー項目] ツールボックスから、制御フローオブジェクトをドラッグアンドドロップします。
6. [データフローの変換先] タブ、[データフローの変換元ツールボックス] タブの順に選択し、[Sybase AdoNet4 ASE Destination] と [ADO NET Source Component] を [データフロー] タブにドラッグアンドドロップします。
7. [接続マネージャー] ウィンドウに使用可能な変換元または変換先が表示されない場合は、[接続マネージャー] ウィンドウを右クリックして、[新しい ADO.NET 接続] を選択します。 既存の接続がある場合はそれを選択し、そうでない場合は、[新規] をクリックします。
8. 変換先 Adaptive Server への接続を新たに作成するには、[ADO.NET の接続マネージャーの構成] ウィンドウで、[新規] をクリックして、[Sybase Adaptive Server Enterprise Data Provider] を選択します。
9. [接続マネージャー] ウィンドウに、接続のプロパティを入力します。
10. バルク挿入を有効にするには、[追加の接続プロパティ] テキストボックスに次のように入力します。
    ```
    enablebulkload=1
    ```

    注意： バルク挿入使用の詳細については、『Adaptive Server Enterprise ADO.NET Data Provider ユーザーズガイド』の「AseBulkCopy」を参照してください。
11. [OK] をクリックします。
12. データフローの ADO.NET 変換元に、接続とデータアクセスモードを設定します。 ADO.NET 変換元からデータフローパスに接続したら、[Sybase AdoNet4 ASE Destination] を右クリックして、[高度な編集の表示] を選択します。
13. [接続マネージャー] タブの [接続マネージャー] フィールドから ASE の接続を選択します。 [コンポーネントのプロパティ] タブで、TableName プロパティを変換先のテーブル名に設定します。
14. [入力列] タブを選択してから、[名前] を選択します。 これにより、変換元テーブルで指定されたすべてのカラムが選択されます。
15. [OK] をクリックして、接続を確立します。

SQL Server Integration Service を使用したデータ転送の詳細については、Microsoft SSIS のマニュアルを参照してください。

Adaptive Server ADO.NET Data Provider での SSRS のサポート

Adaptive Server ADO.NET Data Provider のディストリビューションには、Microsoft SQL Server Reporting Services (SSRS) のカスタムデータ拡張機能コンポーネントが含まれているため、ユーザは認証情報をレポートサーバに格納することができます。

Adaptive Server SSRS コンポーネントでは、以下がサポートされます。

- Microsoft SQL Server 2008
- Microsoft SQL Server 2008 R2

この Sybase.AdoNet2.AseReportingServices という名前のコンポーネントは Adaptive Server ADO.NET Data Provider とともに次のディレクトリにインストールされます。32 ビットシステムの場合は、`%SYBASE%¥DataAccess¥ADONET¥dll` で、64 ビットシステムの場合は、`%SYBASE%¥DataAccess64¥ADONET¥dll` です。

Adaptive Server ADO.NET SSRS コンポーネントの設定

Adaptive Server ADO.NET SSRS コンポーネントを設定します。

1. `Sybase.AdoNet2.AseReportingServices.dll` を `C:¥Program Files (x86)¥Microsoft Visual Studio 9.0¥Common7¥IDE ¥PrivateAssemblies` にコピーします。
2. テキストエディタを使用して `C:¥Program Files (x86)¥Microsoft Visual Studio 9.0¥Common7¥IDE¥PrivateAssemblies` から `RSReportDesigner.config` を開きます。
   - Data セクションの下に以下を入力します。

     ```
     <Extension Name="Sybase"
     Type="Sybase.AdoNet2.AseReportingServices.SybaseClientConnecti
     onWrapper,Sybase.AdoNet2.AseReportingServices"/>
     ```

   - Designer セクションの下に以下を入力します。

     ```
     <Extension Name="Sybase"
     Type="Microsoft.ReportingServices.QueryDesigners.GenericQueryD
     esigner,Microsoft.ReportingServices.QueryDesigners"/>
     ```

3. `RSReportDesigner.config` ファイルを保存します。

データへのアクセスおよび操作方法

Adaptive Server ADO.NET Data Provider では、AseCommand オブジェクトを使用する方法と、AseDataAdapter オブジェクトを使用する方法の 2 とおりの方法でデータにアクセスできます。

- **AseCommandObject** オブジェクト: AseCommand オブジェクトを使用する方法は、プログラマが接続をより制御しやすくなるため、.NET でデータアクセスとデータ操作を行う場合におすすめします。 ただし、AseDataAdapter を使用すればオフラインでの作業が可能になります。
  AseCommand オブジェクトを使用すると、SQL 文を実行してデータベースのデータを直接取得したり修正したりできます。 また AseCommand オブジェクトを使用すると、データベースに対して直接 SQL 文を発行してストアドプロシージャを呼び出すことができます。
  AseCommand オブジェクト内では、**AseDataReader** クラスを使用して、クエリまたはストアドプロシージャから読み込み専用の結果セットを返すことができます。
- **AseDataAdapter** オブジェクト: AseDataAdapter オブジェクトは、結果セット全体を DataSet に取得します。 DataSet は接続が切断された記憶領域で、データベースから取得されたデータが格納されます。 DataSet に格納されたデータは編集できます。編集操作が終了すると、AseDataAdapter オブジェクトは、DataSet に対して行われた変更でデータベースを更新します。
  AseDataAdapter を使用する場合、DataSet に取得したローを他のユーザが変更しないように防ぐ方法はありません。そのため、発生する可能性のある競合すべてを解決するロジックをアプリケーションに組み込む必要があります。

参照：

- AseCommand クラス (123 ページ)
- AseDataReader クラス (168 ページ)
- AseDataAdapter 使用時の競合 (60 ページ)
- AseDataAdapter クラス (158 ページ)

AseCommand を使用したデータの取得と操作

AseDataReader を使用したローの挿入、更新、削除、およびデータの取得についての情報を確認します。

AseCommand オブジェクトを使用したデータの取得

`AseCommand` オブジェクトを使用すると、Adaptive Server データベースに対して SQL 文を発行したり、ストアドプロシージャを呼び出したりできます。

データベースからデータを取得するには、次の種類のコマンドを発行します。

- **ExecuteReader** - 単一の結果セットを返すコマンドを発行します。 デフォルトではカーソルは使用されません。 結果セット全体がクライアント側でフェッチされ、ユーザは 1 度に 1 つのローを前方向へのみフェッチできます。 カーソルの使用をオンにするには、次の行を `ConnectString` に追加します。

  ```
  "Use Cursor=true;"
  ```

  これにより、データベースサーバから結果セット全体をフェッチする代わりに、前方向への読み込み専用カーソルが使用されるようになります。
  カーソルを使用すると、クエリによって大規模な `resultset` が返されることが予測される場合にパフォーマンスを改善できますが、クライアントは必然的に `resultset` 全体を使用できなくなります。

  いずれの場合も、ユーザは一方向にのみ結果セットのローをすばやくループできます。
- **ExecuteScalar** - 単一の値を返すコマンドを発行します。 結果セットの最初のローの最初のカラムや、COUNT や AVG など、集計値を返す SQL 文が返される場合もあります。
- **ExecuteXmlReader** - 単一の結果セットを XML フォーマットで返すコマンドを発行します。 通常、このメソッドは `FOR XML` 句のある **select** 文で使用されます。

Adaptive Server ADO.NET Data Provider に付属の Simple コードサンプルを使用して説明します。

参照：

- Simple サンプルプロジェクトの理解 (13 ページ)
- ExecuteReader メソッド (128 ページ)
- ExecuteScalar メソッド (129 ページ)
- ExecuteXmlReader メソッド (129 ページ)

結果セット全体を返すコマンドの発行
結果セット全体を返すコマンドを発行します。

1. `Connection` オブジェクトを宣言して初期化します。

   C# の場合:

   ```
   AseConnection conn = new AseConnection(connStr);
   ```

   Visual Basic .NET の場合:

   ```
   Dim conn As New AseConnection(connStr)
   ```

   C# の場合:

   ```
   try {
        conn.Open();
   }
   catch (AseException ex)
   {
        <error handling>
   }
   ```

   Visual Basic .NET の場合:

   ```
   Try
        conn.Open()
   Catch ex As AseExecption
        <error handling>
   End Try
   ```

2. `Command` オブジェクトを追加して、SQL 文を定義し、実行します。

   C# の場合:

   ```
   AseCommand cmd = new AseCommand(_
             "select au_lname from authors", conn );
   ```

   Visual Basic .NET の場合:

   ```
   Dim cmd As New AseCommand("select au_lname from authors", conn)
   ```

   注意： ストアドプロシージャを使用してデータベースからデータを取得する場合、ストアドプロシージャから出力パラメータ値と結果セットの両方が返されると、結果セットはリセットされ、出力パラメータ値が参照されると同時に結果セットのローは参照できなくなります。 Sybase では、このような場合、結果セットのローすべてを参照して使い終わるまで、参照側の出力パラメータ値を最後までそのままにしておくことをおすすめします。

   詳細については、「ストアドプロシージャの使用」および「AseParameter クラス」を参照してください。

3. **ExecuteReader** メソッドを呼び出して、`DataReader` オブジェクトを返します。

C# の場合:

```
AseDataReader reader = cmd.ExecuteReader();
```

Visual Basic .NET の場合:

```
Dim reader as AseDataReader = cmd.ExecuteReader()
```

4. 結果を表示します。

C# の場合:

```
listAuthors.BeginUpdate();

while( reader.Read() ) {

          listAuthors.Items.Add( reader.GetString( 0 ) );
}
listAuthors.EndUpdate();
```

Visual Basic .NET の場合:

```
 listAuthors.BeginUpdate()
While reader.Read()
   listAuthors.Items.Add(reader.GetString(0))
End While
listAuthors.EndUpdate()
```

5. DataReader オブジェクトと Connection オブジェクトをクローズします。

C# の場合:

```
reader.Close();

conn.Close();
```

Visual Basic .NET の場合:

```
reader.close()
conn.close()
```

参照：

- ストアドプロシージャ (83 ページ)
- AseParameter クラス (203 ページ)

単一の値のみを返すコマンドの発行

単一の結果セット全体を返すコマンドを発行します。

1. AseConnection オブジェクトを宣言して初期化します。

C# の場合:

```
AseConnection conn = new AseConnection(
   "Data Source='mango';" +
   "Port=5000;" +
   "UID='sa';" +
```

```
   "PWD='';" +
   "Database='pubs2';" );
```

Visual Basic .NET の場合:

```
Dim conn As New AseConnection( _
   "Data Source='mango';" + _
   "Port=5000;" + _
   "UID='sa';" + _
   "PWD='';" + _
   "Database='pubs2';")
```

「mango」には、データベースサーバ名を指定します。

2. 接続をオープンします。

   C# の場合:

   ```
   conn.Open();
   ```

   Visual Basic .NET の場合:

   ```
   conn.Open()
   ```

3. SQL 文を定義して実行する `AseCommand` オブジェクトを追加します。

   C# の場合:

   ```
   AseCommand cmd = new AseCommand(
     "select count(*) from authors where state = 'CA'",
     conn );
   ```

   Visual Basic .NET の場合:

   ```
   Dim cmd As New AseCommand(
    "select count(*) from authors where state = 'CA'"_,
    conn );
   ```

4. **ExecuteScalar** メソッドを呼び出して、値を含むオブジェクトを返します。

   C# の場合:

   ```
   int count = (int) cmd.ExecuteScalar();
   ```

   Visual Basic .NET の場合:

   ```
   Dim count As Integer = cmd.ExecuteScalar()
   ```

5. `AseConnection` オブジェクトをクローズします。

   C# の場合:

   ```
   conn.Close();
   ```

   Visual Basic .NET の場合:

   ```
   conn.Close()
   ```

AseDataReader には、任意のデータ型で結果を返すことのできる **Get** メソッドがいくつかあります。

参照：
- AseDataReader クラス (168 ページ)

XmlReader オブジェクトを返すコマンドの発行
XmlReader オブジェクトを返すコマンドを発行します。

1. Connection オブジェクトを宣言して初期化します。

   C# の場合:

   ```
   AseConnection conn = new AseConnection(connStr);
   ```

   Visual Basic .NET の場合:

   ```
   Dim conn As New AseConnection(connStr)
   ```

2. 接続をオープンします。

   C# の場合:

   ```
   try {
      conn.Open();
   }
   catch (AseException ex)
   {
      <error handling>
   }
   ```

   Visual Basic .NET の場合:

   ```
   Try
      conn.Open()
   Catch ex As AseExecption
      <error handling>
   End Try
   ```

3. Command オブジェクトを追加して、SQL 文を定義し、実行します。

   C# の場合:

   ```
   AseCommand cmd = new AseCommand(
      "select * from authors for xml",
      conn );
   ```

   Visual Basic .NET の場合:

   ```
   Dim cmd As New AseCommand( _
      "select au_lname from authors for xml", _
      conn
   ```

4. `ExecuteReader` メソッドを呼び出して、DataReader オブジェクトを返します。

C# の場合:

```
XmlReader reader = cmd.ExecuteXmlReader();
```

Visual Basic .NET の場合:

```
Dim reader as XmlReader = cmd.ExecuteXmlReader()
```

5. XML の結果を使用します。

   C# の場合:

   ```
   reader.read();
   <process xml>
   ```

   Visual Basic .NET の場合:

   ```
   reader.read()
   <process xml>
   ```

6. DataReader オブジェクトと Connection オブジェクトをクローズします。

   C# の場合:

   ```
   reader.Close();
   conn.Close();
   ```

   Visual Basic .NET の場合:

   ```
   reader.close()
   conn.close()
   ```

AseCommand オブジェクトを使用したローの挿入、更新、削除

`ExecuteNonQuery` 関数を使用して、AseCommand オブジェクトで挿入、更新、削除の操作を実行します。

The **ExecuteNonQuery** 関数は、結果セットを返さないコマンド (SQL 文またはストアドプロシージャ) を発行します。

コマンドの独立性レベルを設定する場合は、`AseCommand` オブジェクトを `AseTransaction` オブジェクトの一部として使用してください。`AseTransaction` オブジェクトを使用せずにデータを変更すると、Adaptive Server ADO.NET Data Provider は **autocommit** モードで動作し、加えられたすべての変更がただちに適用されます。

参照：

- トランザクション処理 (86 ページ)
- ExecuteNonQuery メソッド (127 ページ)
- プライマリキー値の取得 (74 ページ)

ローを挿入するコマンドの発行

ローを挿入するコマンドを発行します。

1. `AseConnection` オブジェクトを宣言して初期化します。

   C# の場合:

   ```
   AseConnection conn = new AseConnection( c_connStr );
   ```

   Visual Basic .NET の場合:

   ```
   Dim conn As New AseConnection(c_connStr)
   ```

2. 接続をオープンします。

   C# の場合:

   ```
   conn.Open();
   ```

   Visual Basic .NET の場合:

   ```
   conn.Open()
   ```

3. **Insert** 文を定義して実行する `AseCommand` オブジェクトを追加します。

   C# の場合:

   ```
   AseCommand insertCmd = new AseCommand(
      "INSERT INTO publishers " +
      " ( pub_id, pub_name, city, state) " +
      " VALUES( @pub_id, @pub_name, @city, @state )",
      conn);
   ```

   Visual Basic .NET の場合:

   ```
   Dim insertCmd As new AseCommand( _
      "INSERT INTO publishers " + _
      " ( pub_id, pub_name, city, state) " + _
      " VALUES (@pub_id, @pub_name, @city, @state )",  _
      conn )
   ```

4. `AseCommand` オブジェクトのパラメータを設定します。

   次のコードは、それぞれ `dept_id` カラムと `dept_name` カラムのパラメータを定義します。

   C# の場合:

   ```
   AseParameter parm = new AseParameter("@pub_id", AseDbType.Char,
   4);
   insertCmd.Parameters.Add( parm );
   parm = new AseParameter("@pub_name", AseDbType.VarChar, 40);
   insertCmd.Parameters.Add( parm );
   parm = new AseParameter("@city", AseDbType.VarChar, 20);
   insertCmd.Parameters.Add( parm );
   parm = new AseParameter("@state", AseDbType.Char, 2);
   insertCmd.Parameters.Add( parm );
   ```

Visual Basic .NET の場合:

```
Dim parm As New AseParameter("@pub_id", AseDbType.Char, 4)
insertCmd.Parameters.Add(parm)
parm = New AseParameter("@pub_name", AseDbType.VarChar, 40)
insertCmd.Parameters.Add(parm)
parm = New AseParameter("@city", AseDbType.VarChar, 20)
insertCmd.Parameters.Add(parm)
parm = New AseParameter("@state", AseDbType.Char, 2)
insertCmd.Parameters.Add(parm)
```

5. 新しい値を挿入して `ExecuteNonQuery` メソッドを呼び出し、データベースに変更を適用します。

   C# の場合:

   ```
   int recordsAffected = 0;
   insertCmd.Parameters[0].Value = "9901";
   insertCmd.Parameters[1].Value = "New Publisher";
   insertCmd.Parameters[2].Value = "Concord";
   insertCmd.Parameters[3].Value = "MA";
   recordsAffected = insertCmd.ExecuteNonQuery();
   insertCmd.Parameters[0].Value = "9902";
   insertCmd.Parameters[1].Value = "My Publisher";
   insertCmd.Parameters[2].Value = "Dublin";
   insertCmd.Parameters[3].Value = "CA";
   recordsAffected = insertCmd.ExecuteNonQuery();
   ```

   Visual Basic .NET の場合:

   ```
   Dim recordsAffected As Integer
   insertCmd.Parameters(0).Value = "9901"
   insertCmd.Parameters(1).Value = "New Publisher"
   insertCmd.Parameters(2).Value = "Concord"
   insertCmd.Parameters(3).Value = "MA"
   recordsAffected = insertCmd.ExecuteNonQuery()
   insertCmd.Parameters(0).Value = "9902"
   insertCmd.Parameters(1).Value = "My Publisher"
   insertCmd.Parameters(2).Value = "Dublin"
   insertCmd.Parameters(3).Value = "CA"
   recordsAffected = insertCmd.ExecuteNonQuery()
   ```

   注意：ExecuteNonQuery メソッドでは、**Insert**、**Update**、または `Delete` 文を使用できます。

6. 結果を表示して、ウィンドウのグリッドにバインドします。

   C# の場合:

   ```
   AseCommand selectCmd = new AseCommand("SELECT * FROM publishers",
   conn );
   AseDataReader dr = selectCmd.ExecuteReader();
   dataGrid.DataSource = dr;
   ```

   Visual Basic .NET の場合:

```
Dim selectCmd As New AseCommand("SELECT * FROM publishers", conn)
Dim dr As AseDataReader = selectCmd.ExecuteReader()
DataGrid.DataSource = dr
```

7. AseDataReader オブジェクトと `AseConnection` オブジェクトをクローズします。

   C# の場合:

   ```
   dr.Close();
   conn.Close();
   ```

   Visual Basic .NET の場合:

   ```
   dr.Close()
   conn.Close()
   ```

ローを更新するコマンドの発行

ローを更新するコマンドを発行します。

1. `AseConnection` オブジェクトを宣言して初期化します。

   C# の場合:

   ```
   AseConnection conn = new AseConnection(
      c_connStr );
   ```

   Visual Basic .NET の場合:

   ```
   Dim conn As New AseConnection(c_connStr)
   ```

2. 接続をオープンします。

   C# の場合:

   ```
   conn.Open();
   ```

   Visual Basic .NET の場合:

   ```
   conn.Open()
   ```

3. **Update** 文を定義して実行する `AseCommand` オブジェクトを追加します。

   C# の場合:

   ```
   AseCommand updateCmd = new AseCommand(
      "UPDATE publishers " +
      "SET pub_name = 'My Publisher' " +
      "WHERE pub_id='9901'",
      conn );
   ```

   Visual Basic .NET の場合:

   ```
   Dim updateCmd As New AseCommand( _
      "UPDATE publishers " + _
      "SET pub_name = 'My Publisher' " + _
      "WHERE pub_id='9901'", _
      conn )
   ```

詳細については、「ストアドプロシージャの使用」および「AseParameter クラス」を参照してください。

4. **ExecuteNonQuery** メソッドを呼び出して、データベースに変更を適用します。

   C# の場合:

   ```
   int recordsAffected = updateCmd.ExecuteNonQuery();
   ```

   Visual Basic .NET の場合:

   ```
   Dim recordsAffected As Integer =
      updateCmd.ExecuteNonQuery()
   ```

5. 結果を表示して、ウィンドウのグリッドにバインドします。

   C# の場合:

   ```
   AseCommand selectCmd = new AseCommand(
      "SELECT * FROM publishers", conn );
   AseDataReader dr = selectCmd.ExecuteReader();
   dataGrid.DataSource = dr;
   ```

   Visual Basic .NET の場合:

   ```
   Dim selectCmd As New AseCommand(_
      "SELECT * FROM publishers", conn)
   Dim dr As AseDataReader = selectCmd.ExecuteReader()
   DataGrid.DataSource = dr
   ```

6. `AseDataReader` オブジェクトと `AseConnection` オブジェクトをクローズします。

   C# の場合:

   ```
   dr.Close();
   conn.Close();
   ```

   Visual Basic .NET の場合:

   ```
   dr.Close()
   conn.Close()
   ```

参照：

- ストアドプロシージャ (83 ページ)
- AseParameter クラス (203 ページ)

ローを削除するコマンドの発行

ローを削除するコマンドを発行します。

1. `AseConnection` オブジェクトを宣言して初期化します。

   C# の場合:

   ```
   AseConnection conn = new AseConnection(c_connStr);
   ```

Visual Basic .NET の場合:

```
Dim conn As New AseConnection(c_connStr)
```

2. 接続をオープンします。

   C# の場合:

   ```
   conn.Open();
   ```

   Visual Basic .NET の場合:

   ```
   conn.Open()
   ```

3. **Delete** 文を定義して実行する `AseCommand` オブジェクトを作成します。

   C# の場合:

   ```
   AseCommand updateCmd = new AseCommand
      "DELETE FROM publishers " +
      " WHERE (pub_id > '9900')",
      conn );
   ```

   Visual Basic .NET の場合:

   ```
   Dim updateCmd As New AseCommand(_
   "DELETE FROM publishers " + _
   "WHERE (pub_id > '9900')", _
   conn )
   ```

4. **ExecuteNonQuery** メソッドを呼び出して、データベースに変更を適用します。

   C# の場合:

   ```
   int recordsAffected = deleteCmd.ExecuteNonQuery();
   ```

   Visual Basic .NET の場合:

   ```
   Dim recordsAffected As Integer =
      updateCmd.ExecuteNonQuery()
   ```

5. `AseConnection` オブジェクトをクローズします。

   C# の場合:

   ```
   conn.Close();
   ```

   Visual Basic .NET の場合:

   ```
   dr.Close()
   conn.Close()
   ```

DataReader スキーマ情報の取得

結果セット内のカラムに関するスキーマ情報を取得できます。

AseDataReader を使用している場合、**GetSchemaTable** メソッドを使用して結果セットの情報を取得できます。 **GetSchemaTable** メソッドは、標準の .NET

DataTableオブジェクトを返します。このオブジェクトは、カラムプロパティも含め、結果セット内のすべてのカラムに関する情報を提供します。

参照：

- GetSchemaTable メソッド (179 ページ)

GetSchemaTable メソッドを使用した結果セット情報の取得

GetSchemaTable メソッドを使用して結果セット情報を取得します。

1. Connection オブジェクトを宣言して初期化します。

   C# の場合:

   ```
   AseConnection conn = new AseConnection(
      c_connStr );
   ```

   Visual Basic .NET の場合:

   ```
   Dim conn As New AseConnection( _
      c_connStr )
   ```

2. 接続をオープンします。

   C# の場合:

   ```
   conn.Open();
   ```

   Visual Basic .NET の場合:

   ```
   conn.Open()
   ```

3. 目的の **Select** 文を使用して AseCommand オブジェクトを作成します。 このクエリの結果セットのスキーマが返されます。

   C# の場合:

   ```
   AseCommand cmd = new AseCommand(
      "SELECT * FROM authors", conn );
   ```

   Visual Basic .NET の場合:

   ```
   Dim cmd As New AseCommand( _
      "SELECT * FROM authors", conn )
   ```

4. AseDataReader オブジェクトを作成して、作成した Command オブジェクトを実行します。

   C# の場合:

   ```
   AseDataReader dr = cmd.ExecuteReader();
   ```

   Visual Basic .NET の場合:

   ```
   Dim dr As AseDataReader = cmd.ExecuteReader()
   ```

5. データソースからのスキーマを DataTable に書き込みます。

   C# の場合:

   ```
   DataTable
   schema = dr.GetSchemaTable();
   ```

   Visual Basic .NET の場合:

   ```
   Dim schema As DataTable = _
      dr.GetSchemaTable()
   ```

6. AseDataReader オブジェクトと AseConnection オブジェクトをクローズします。

   C# の場合:

   ```
   dr.Close();
   conn.Close();
   ```

   Visual Basic .NET の場合:

   ```
   dr.Close()
   conn.Close()
   ```

7. DataTable をウィンドウのグリッドにバインドします。

   C# の場合:

   ```
   dataGrid.DataSource = schema;
   ```

   Visual Basic .NET の場合:

   ```
   dataGrid.DataSource = schema
   ```

AseDataAdapter を使用したデータへのアクセスと操作

AseDataAdapter を使用して、データの取得、ローの挿入、更新、または削除を実行できます。

AseDataAdapter オブジェクトを使用したデータの取得

AseDataAdapter は、**Fill** メソッドを使用してクエリの結果を DataSet に書き込み、その DataSet を表示グリッドにバインドすることによって結果セット全体を表示します。

AseDataAdapter を使用すると、単一の結果セットを返す任意の文字列 (SQL 文またはストアドプロシージャ) を渡すことができます。 AseDataAdapter では、デフォルトで、すべてのローが 1 度の操作でフェッチされます。 カーソルを使用するには、接続文字列でプロパティを **'use cursor = true'** に設定します。 この場合、前方向への読み込み専用カーソルが使用され、すべてのローが読み込まれると、カーソルは自動的にクローズします。 AseDataAdapter では、DataSet に変更

を加えることができます。 変更操作が完了したら、データベースに再接続して変更を適用します。

`AseDataAdapter` オブジェクトを使用すると、特定のジョインにもとづく結果セットを取得できます。

参照：

- AseDataAdapter クラス (158 ページ)

AseDataAdapter オブジェクトを使用したデータの取得

この例では、AseDataAdapter を使用して DataSet に書き込む方法を示します。

1. データベースに接続します。
2. 新しい DataSet を作成します。 ここでは、DataSet の名前を「Results」とします。

   C# の場合:

   ```
   DataSet ds =new DataSet ();
   ```

   Visual Basic .NET の場合:

   ```
   Dim ds As New DataSet()
   ```
3. SQL 文を実行して「Results」という DataSet に結果を書き込む、新しい `AseDataAdapter` オブジェクトを作成します。

   C# の場合:

   ```
   AseDataAdapter da=new
      AseDataAdapter(txtSQLStatement.Text, _conn);
   da.Fill(ds, "Results"),
   ```

   Visual Basic .NET の場合:

   ```
   Dim da As New
      AseDataAdapter(txtSQLStatement.Text, conn)
   da.Fill(ds, "Results")
   ```
4. DataSet をウィンドウのグリッドにバインドします。

   C# の場合:

   ```
   dgResults.DataSource = ds.Tables["Results"],
   ```

   Visual Basic .NET の場合:

   ```
   dgResults.DataSource = ds.Tables("Results")
   ```

AseDataAdapter オブジェクトを使用したローの挿入、更新、削除

`AseDataAdapter` オブジェクトは、DataSet に結果セットを取得します。この結果セットは、テーブルのコレクション、およびそれらのテーブル間の関係と制約を保持します。

DataSet は .NET Framework 内に構築され、データベースへの接続に使用される Adaptive Server ADO.NET Data Provider から独立しています。

`AseDataAdapter` を使用するときに、まだデータベースに接続していない場合は接続がオープンされ、`DataSet` への書き込みが行われた後、接続が明示的にオープンされたものでない場合は接続がクローズされます。 ただし、`DataSet` に書き込まれたデータは、データベースとの接続が切断している間も変更できます。

変更をデータベースへただちに適用する必要がない場合は、データやスキーマを含め、DataSet を **WriteXml** メソッドを使用して XML ファイルに書き込むことができます。 この場合、後から **ReadXml** メソッドを使用して DataSet をロードすれば、変更を適用できます。

詳細については、.Net Framework のドキュメントで **WriteXml** と **ReadXml** の説明を参照してください。

Update メソッドを呼び出して DataSet からデータベースへ変更を適用するとき、`AseDataAdapter` は加えられた変更を分析し、必要に応じて **Insert**、**Update**、**Delete** のいずれかのコマンドを呼び出します。

DataSet を使用して変更 (挿入、更新、削除) できるのは、単一のテーブルのデータのみです。 ジョインにもとづく結果セットは更新できません。

注意： DataSet に対する変更はすべて、接続が切断している間に行われます。 これは、データベース内で、これらのローがアプリケーションによってロックされていないことを意味します。 変更の適用対象データを他のユーザがすでに変更している場合もあるため、DataSet に加えた変更をデータベースに適用するときに発生する可能性のある競合すべてを解決するように、アプリケーションを設計する必要があります。

AseDataAdapter 使用時の競合

AseDataAdapter の使用時は、アプリケーションロジックで競合を解決する必要があります。

- *ユニークなプライマリキー* - 2 人のユーザが同じテーブルに新しいローを挿入するときに、各ローにユニークなプライマリキーを設定する必要があります。自動インクリメントプライマリキーのあるテーブルでは、DataSet の値とデータソースの値とが同期しなくなることがあります。

- *1 つの値に対する複数の更新* - 2 人のユーザが同じ値を変更するときに、どちらの値が正しいかを判定する論理をアプリケーションに組み込む必要があります。
- *スキーマの変更* - DataSet で更新したテーブルのスキーマを他のユーザが変更した場合、その DataSet に加えた変更をデータベースに適用すると、更新が失敗します。
- *データの同時実行性* - 同時実行性のある複数のアプリケーションが一貫したデータを参照できる場合、 `AseDataAdapter` はフェッチしたローをロックしないため、DataSet を取得してオフラインで作業しているときに、2 人目のユーザがデータベース内の値を更新できます。

これらの潜在的な問題の多くは、`AseCommand`、`AseDataReader`、`AseTransaction` のオブジェクトを使用してデータベースに変更を適用することで回避できます。 トランザクションの独立性レベルを設定し、別のユーザが変更できないようにローをロックするため、Sybase では `AseTransaction` オブジェクトを使用することをおすすめします。

競合解決処理を単純化するため、**insert** 文、**update** 文、または **delete** 文がストアドプロシージャコールに組み込まれるように設計することができます。 **Insert**、**Update**、**Delete** 文をストアドプロシージャに組み込むと、操作が失敗した場合にエラーを検出できます。 これらの文に加えて、ストアドプロシージャにエラー処理論理を追加することで、操作が失敗した場合にログファイルへのエラーの記録、操作の再試行などの適切なアクションを実行できます。

参照：
- プライマリキー値の取得 (74 ページ)
- AseCommand オブジェクトを使用したローの挿入、更新、削除 (51 ページ)

AseDataAdapter 使用によるテーブルへのローの挿入
AseDataAdapter オブジェクトを使用してローを挿入します。

1. `AseConnection` オブジェクトを宣言して初期化します。

   C# の場合:

   ```
   AseConnection conn = new AseConnection(c_connStr);
   ```

   Visual Basic .NET の場合:

   ```
   Dim conn As New AseConnection( _
      c_connStr )
   ```

2. 接続をオープンします。

   C# の場合:

   ```
   conn.Open();
   ```

Visual Basic .NET の場合:

```
conn.Open()
```

3. 新しい `AseDataAdapter` オブジェクトを作成します。

   C# の場合:

```
AseDataAdapter adapter = new AseDataAdapter();
adapter.MissingMappingAction =
   MissingMappingAction.Passthrough;
adapter.MissingSchemaAction =        MissingSchemaAction.Add;
```

   Visual Basic .NET の場合:

```
Dim adapter As New AseDataAdapter()
adapter.MissingMappingAction = _
   MissingMappingAction.Passthrough
adapter.MissingSchemaAction = _
   MissingSchemaAction.Add
```

4. 必要な `AseCommand` オブジェクトを作成して、必要なパラメータをすべて定義します。

   次のコードは、**Select** コマンドと **Insert** コマンドを作成して、**Insert** コマンドのパラメータを定義します。

   C# の場合:

```
adapter.SelectCommand = new AseCommand(
   "SELECT * FROM publishers", conn );
adapter.InsertCommand = new AseCommand(
   "INSERT INTO publishers( pub_id, pub_name, city, state) " +
   "VALUES( @pub_id, @pub_name, @city, @state  )", conn);
adapter.InsertCommand.UpdatedRowSource = UpdateRowSource.None;
AseParameter parm = new AseParameter("@pub_id", AseDbType.Char,
4);
parm.SourceColumn = "pub_id";
parm.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parm );
parm = new AseParameter("@pub_name", AseDbType.VarChar, 40);
parm.SourceColumn = "pub_name";
parm.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parm );
parm = new AseParameter("@city", AseDbType.VarChar, 20);
parm.SourceColumn = "city";
parm.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parm );
parm = new AseParameter("@state", AseDbType.Char, 2);
parm.SourceColumn = "state";
parm.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parm );
```

   Visual Basic .NET の場合:

```
adapter.SelectCommand = New AseCommand( _
   "SELECT * FROM publishers", conn )
```

```
adapter.InsertCommand = New AseCommand( _
   "INSERT INTO publishers( pub_id, pub_name,  city, state) " + _
   " VALUES( @pub_id, @pub_name, @city, @state  )", conn)
adapter.InsertCommand.UpdatedRowSource = _
   UpdateRowSource.None
Dim parm As New AseParameter("@pub_id", AseDbType.Char, 4)
parm.SourceColumn = "pub_id"
parm.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parm )
parm = New AseParameter("@pub_name", AseDbType.VarChar, 40)
parm.SourceColumn = "pub_name"
parm.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parm )
parm = New AseParameter("@city", AseDbType.VarChar, 20)
parm.SourceColumn = "city"
parm.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parm )
parm = New AseParameter("@state", AseDbType.Char, 2)
parm.SourceColumn = "state"
parm.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parm )
```

5. `DataTable` に **Select** 文の結果を書き込みます。

   C# の場合:

   ```
   DataTable dataTable = new DataTable( "publishers" );
   int rowCount = adapter.Fill( dataTable );
   ```

   Visual Basic .NET の場合:

   ```
   Dim dataTable As New DataTable( "publishers" )
   Dim rowCount As Integer = adapter.Fill( dataTable )
   ```

6. `DataTable` に新しいローを挿入して、変更をデータベースに適用します。

   C# の場合:

   ```
   DataRow row1 = dataTable.NewRow();
   row1[0] = "9901";
   row1[1] = "New Publisher";
   row1[2] = "Concord";
   row1[3] = "MA";
   dataTable.Rows.Add( row1 );
   DataRow row2 = dataTable.NewRow();
   row2[0] = "9902";
   row2[1] = "My Publisher";
   row2[2] = "Dublin";
   row2[3] = "CA";
   dataTable.Rows.Add( row2 );
   int recordsAffected = adapter.Update( dataTable );
   ```

   Visual Basic .NET の場合:

   ```
   Dim row1 As DataRow = dataTable.NewRow()
   row1(0) = "9901"
   row1(1) = "New Publisher"
   ```

```
row1(2) = "Concord"
row1(3) = "MA"
dataTable.Rows.Add( row1 )
Dim row2 As DataRow = dataTable.NewRow()
row2(0) = "9902"
row2(1) = "My Publisher"
row2(2) = "Dublin"
row2(3) = "CA"
dataTable.Rows.Add( row2 )
Dim recordsAffected As Integer =_
   adapter.Update( dataTable )
```

7. 更新の結果を表示します。

   C# の場合:

   ```
   dataTable.Clear();
   rowCount = adapter.Fill( dataTable );
   dataGrid.DataSource = dataTable;
   ```

   Visual Basic .NET の場合:

   ```
   dataTable.Clear()
   rowCount = adapter.Fill( dataTable )
   dataGrid.DataSource = dataTable
   ```

8. 接続をクローズします。

   C# の場合:

   ```
   conn.Close();
   ```

   Visual Basic .NET の場合:

   ```
   conn.Close()
   ```

AseDataAdapter オブジェクトの使用によるローの更新

AseDataAdapter オブジェクトを使用してローを更新します。

1. `AseConnection` オブジェクトを宣言して初期化します。

   C# の場合:

   ```
   AseConnection conn = new AseConnection( c_connStr );
   ```

   Visual Basic .NET の場合:

   ```
   Dim conn As New AseConnection( _
      c_connStr )
   ```

2. 接続をオープンします。

   C# の場合:

   ```
   conn.Open();
   ```

   Visual Basic .NET の場合:

```
conn.Open()
```

3. 新しい AseDataAdapter オブジェクトを作成します。

   C# の場合:

```
AseDataAdapter adapter = new AseDataAdapter();
adapter.MissingMappingAction =
   MissingMappingAction.Passthrough;
adapter.MissingSchemaAction =        MissingSchemaAction.Add;
```

   Visual Basic .NET の場合:

```
Dim adapter As New AseDataAdapter()
adapter.MissingMappingAction = _
   MissingMappingAction.Passthrough
adapter.MissingSchemaAction = _
   MissingSchemaAction.Add
```

4. AseCommand オブジェクトを作成して、そのパラメータを定義します。

   次のコードは、**Select** コマンドと **Update** コマンドを作成して、**Update** コマンドのパラメータを定義します。

   C# の場合:

```
adapter.SelectCommand = new AseCommand(
   "SELECT * FROM publishers WHERE pub_id > '9900'",    conn );
adapter.UpdateCommand = new AseCommand(
   "UPDATE publishers SET pub_name = @pub_name, " +
   "city = @city, state = @state " +
   "WHERE pub_id = @pub_id", conn );
adapter.UpdateCommand.UpdatedRowSource =
   UpdateRowSource.None;
AseParameter parm = new AseParameter("@pub_id",
   AseDbType.Char, 4);
parm.SourceColumn = "pub_id";
parm.SourceVersion = DataRowVersion.Current;
adapter.UpdateCommand.Parameters.Add( parm );
parm = new AseParameter("@pub_name",
   AseDbType.VarChar, 40);
parm.SourceColumn = "pub_name";
parm.SourceVersion = DataRowVersion.Current;
adapter.UpdateCommand.Parameters.Add( parm );
parm = new AseParameter("@city",
   AseDbType.VarChar, 20);
parm.SourceColumn = "city";
parm.SourceVersion = DataRowVersion.Current;
adapter.UpdateCommand.Parameters.Add( parm );
parm = new AseParameter("@state",
   AseDbType.Char, 2);
parm.SourceColumn = "state";
parm.SourceVersion = DataRowVersion.Current;
adapter.UpdateCommand.Parameters.Add( parm );
```

   Visual Basic .NET の場合:

```
adapter.SelectCommand = New AseCommand( _
  "SELECT * FROM publishers WHERE pub_id > '9900'",_
  conn )
adapter.UpdateCommand = New AseCommand( _
  "UPDATE publishers SET pub_name = @pub_name, " + _
  "city = @city, state = @state " + _
  "WHERE pub_id = @pub_id", conn )
adapter.UpdateCommand.UpdatedRowSource = _
  UpdateRowSource.None
Dim parm As New AseParameter("@pub_id", _
  AseDbType.Char, 4)
parm.SourceColumn = "pub_id"
parm.SourceVersion = DataRowVersion.Current
adapter.UpdateCommand.Parameters.Add( parm )
parm = New AseParameter("@pub_name", _
  AseDbType.VarChar, 40)
parm.SourceColumn = "pub_name"
parm.SourceVersion = DataRowVersion.Current
adapter.UpdateCommand.Parameters.Add( parm )
parm = New AseParameter("@city", _
  AseDbType.VarChar, 20)
parm.SourceColumn = "city"
parm.SourceVersion = DataRowVersion.Current
adapter.UpdateCommand.Parameters.Add( parm )
parm = New AseParameter("@state", _
  AseDbType.Char, 2)
parm.SourceColumn = "state"
parm.SourceVersion = DataRowVersion.Current
adapter.UpdateCommand.Parameters.Add( parm )
```

5. `DataTable` に **Select** 文の結果を書き込みます。

   C# の場合:

   ```
   DataTable dataTable = new DataTable( "publishers" );
   int rowCount = adapter.Fill( dataTable );
   ```

   Visual Basic .NET の場合:

   ```
   Dim dataTable As New DataTable( "publishers" )
   Dim rowCount As Integer = adapter.Fill( dataTable )
   ```

6. ローの更新値で `DataTable` を更新して、変更をデータベースに適用します。

   C# の場合:

   ```
   foreach ( DataRow row in dataTable.Rows )
   {
      row[1] = ( string ) row[1] + "_Updated";
   }
   int recordsAffected = adapter.Update( dataTable );
   ```

   Visual Basic .NET の場合:

   ```
   Dim row as DataRow
   For Each row in dataTable.Rows
      row(1) = row(1) + "_Updated"
   ```

```
Next
Dim recordsAffected As Integer = _
adapter.Update( dataTable )
```

7. 結果をウィンドウのグリッドにバインドします。

   C# の場合:

   ```
   dataTable.Clear();
   adapter.SelectCommand.CommandText =
      "SELECT * FROM publishers";
   rowCount = adapter.Fill( dataTable );
   dataGrid.DataSource = dataTable;
   ```

   Visual Basic .NET の場合:

   ```
   dataTable.Clear()
   adapter.SelectCommand.CommandText = _
      "SELECT * FROM publishers";
   rowCount = adapter.Fill( dataTable )
   dataGrid.DataSource = dataTable
   ```

8. 接続をクローズします。

   C# の場合:

   ```
   conn.Close();
   ```

   Visual Basic .NET の場合:

   ```
   conn.Close()
   ```

AseDataAdapter オブジェクトの使用によるテーブルからのローの削除

AseDataAdapter オブジェクトを使用してテーブルからローを削除します。

1. `AseConnection` オブジェクトを宣言して初期化します。

   C# の場合:

   ```
   AseConnection conn = new AseConnection( c_connStr );
   ```

   Visual Basic .NET の場合:

   ```
   Dim conn As New AseConnection( _
      c_connStr )
   ```

2. 接続をオープンします。

   C# の場合:

   ```
   conn.Open();
   ```

   Visual Basic .NET の場合:

   ```
   conn.Open()
   ```

3. `AseDataAdapter` オブジェクトを作成します。

   C# の場合:

```
AseDataAdapter adapter = new AseDataAdapter();
adapter.MissingMappingAction =
   MissingMappingAction.Passthrough;
adapter.MissingSchemaAction =    MissingSchemaAction.AddWithKey;
```

Visual Basic .NET の場合:

```
Dim adapter As New AseDataAdapter()
adapter.MissingMappingAction = _
   MissingMappingAction.Passthrough
adapter.MissingSchemaAction = _
   MissingSchemaAction.AddWithKey
```

4. 必要な `AseCommand` オブジェクトを作成して、必要なパラメータをすべて定義します。

   次のコードは、**Select** コマンドと **Delete** コマンドを作成して、**Delete** コマンドのパラメータを定義します。

   C# の場合:

```
adapter.SelectCommand = new AseCommand(
   "SELECT * FROM publishers WHERE pub_id > '9900'",
   conn );
adapter.DeleteCommand = new AseCommand(
   "DELETE FROM publishers WHERE pub_id = @pub_id",
   conn );
adapter.DeleteCommand.UpdatedRowSource =
   UpdateRowSource.None;
AseParameter parm = new AseParameter("@pub_id",
   AseDbType.Char, 4);
parm.SourceColumn = "pub_id";
parm.SourceVersion = DataRowVersion.Original;
adapter.DeleteCommand.Parameters.Add( parm );
```

   Visual Basic .NET の場合:

```
adapter.SelectCommand = New AseCommand( _
   "SELECT * FROM publishers WHERE pub_id > '9900'", _
   conn )
adapter.DeleteCommand = New AseCommand( _
   "DELETE FROM publishers WHERE pub_id = @pub_id", conn )
adapter.DeleteCommand.UpdatedRowSource = _
   UpdateRowSource.None
Dim parm As New AseParameter("@pub_id", _
   AseDbType.Char, 4)
parm.SourceColumn = "pub_id"
parm.SourceVersion = DataRowVersion.Original
adapter.DeleteCommand.Parameters.Add( parm )
```

5. `DataTable` に **Select** 文の結果を書き込みます。

   C# の場合:

```
DataTable dataTable = new DataTable( "publishers" );
int rowCount = adapter.Fill( dataTable );
```

Visual Basic .NET の場合:

```
Dim dataTable As New DataTable( "publishers" )
Dim rowCount As Integer = adapter.Fill( dataTable )
```

6. `DataTable` を変更して、データベースに変更を適用します。

   C# の場合:

```
foreach ( DataRow row in dataTable.Rows )
{
   row.Delete();
}
int recordsAffected = adapter.Update( dataTable );
```

   Visual Basic .NET の場合:

```
Dim row as DataRow
For Each row in dataTable.Rows
   row.Delete()
Next
Dim recordsAffected As Integer =_
       adapter.Update( dataTable )
```

7. 結果をウィンドウのグリッドにバインドします。

   C# の場合:

```
dataTable.Clear();
rowCount = adapter.Fill( dataTable );
dataGrid.DataSource = dataTable;
```

   Visual Basic .NET の場合:

```
dataTable.Clear()
rowCount = adapter.Fill( dataTable )dataGrid.
DataSource = dataTable
```

8. 接続をクローズします。

   C# の場合:

```
conn.Close();
```

   Visual Basic .NET の場合:

```
conn.Close()
```

AseDataAdapter スキーマ情報の取得

FillSchema メソッドを **AseDataAdapter** とともに使用することで、DataSet 内の結果セットに関するスキーマ情報を取得します。

FillSchema メソッドは、標準の .NET `DataTable` オブジェクトを返します。このオブジェクトは、結果セット内の全カラムの名前を提供します。

参照：

- FillSchema メソッド (162 ページ)

FillSchema メソッドの使用による DataSet スキーマ情報の取得

`FillSchema` メソッドを使用してデータセットスキーマ情報を取得します。

1. `AseConnection` オブジェクトを宣言して初期化します。

   C# の場合:

   ```
   AseConnection   conn = new AseConnection(
      c_connStr );
   ```

   Visual Basic .NET の場合:

   ```
   Dim conn As New AseConnection( _
      c_connStr )
   ```

2. 接続をオープンします。

   C# の場合:

   ```
   conn.Open();
   ```

   Visual Basic .NET の場合:

   ```
   conn.Open()
   ```

3. 目的の **Select** 文を使用して `AseDataAdapter` を作成します。 このクエリの結果セットのスキーマが返されます。

   C# の場合:

   ```
   AseDataAdapter  adapter = new AseDataAdapter(
      "SELECT * FROM employee", conn );
   ```

   Visual Basic .NET の場合:

   ```
   Dim adapter As New AseDataAdapter( _
      "SELECT * FROM employee", conn )
   ```

4. スキーマを書き込む新しい `DataTable` オブジェクトを作成します。この例では「Table」という名前で作成します。

   C# の場合:

   ```
   DataTable dataTable = new DataTable( "Table" );
   ```

   Visual Basic .NET の場合:

   ```
   Dim dataTable As New DataTable( "Table" )
   ```

5. データソースからのスキーマを `DataTable` に書き込みます。

   C# の場合:

   ```
   adapter.FillSchema( dataTable, SchemaType.Source );
   ```

Visual Basic .NET の場合:

```
adapter.FillSchema( dataTable, SchemaType.Source )
```

6. `AseConnection` オブジェクトをクローズします。

   C# の場合:

   ```
   conn.Close();
   ```

   Visual Basic .NET の場合:

   ```
   conn.Close()
   ```

7. DataSet をウィンドウのグリッドにバインドします。

   C# の場合:

   ```
   dataGrid.DataSource = dataTable;
   ```

   Visual Basic .NET の場合:

   ```
   dataGrid.DataSource = dataTable
   ```

Adaptive Server ADO.NET Data Provider でのバルクロードのサポート

Adaptive Server ADO.NET Data Provider では、バルクロードインタフェースがサポートされており、Adaptive Server への大量のローセットへの挿入を高速化できます。

ENABLEBULKLOAD 接続プロパティを設定することによって、**ASEBulkCopy** でバルクロードインタフェースを起動できます。 次の 2 つのタイプのバルクロードがサポートされています。

- 配列挿入 - このタイプのバルクロードは、単一または複数文トランザクション内で使用できます。
- バルクコピー - これは単一文トランザクションのみでサポートされています。また、Adaptive Server 上で **select into/bulkcopy** オプションをオンにする必要があります。
  ターゲットテーブルが高速バージョンの **bulk copy** の条件を満たす場合、Adaptive Server はこのバージョンの **bulk copy** を使用してローを挿入します。

注意： **select into/bulkcopy** オプションを有効にして、バルクコピーモードを使用する場合、データベースのリカバリ性に影響します。 管理者は、**bulk copy** の処理が完了したら、今後のリカバリ性を保証するためにデータベースをダンプする必要があります。

バルクロードオプションの使用法

使用ケース	その他の注意事項	使用するバルクロードオプション	注意
単一または少数のローの挿入。		なし	
多数のローのバッチ挿入。	バッチは複数文トランザクションの一部。	配列挿入	バルクロードが無効な場合よりも、ローが高速に挿入される。
	リカバリ性を考慮する必要があるため、Adaptive Server **select into** または **bulkcopy** オプションを有効化できない。	配列挿入	バルクロードが無効な場合よりも、ローが高速に挿入される。
	バッチは単一トランザクションであり、Adaptive Server**select into/bulkcopy** オプションが有効化されている。	バルクコピー	Adaptive Server で、配列挿入よりも速い、高速バルクコピーを使用可能。 バルクコピーのパフォーマンスは、高速バルクコピーを使用しない場合でも、配列挿入よりもわずかに高速。

select into/bulkcopy を有効化した場合の影響と、高速またはログ出力 **bulk copy** の使用条件については、『Adaptive Server Enterprise ユーティリティガイド』を参照してください。

パフォーマンスの考慮事項
この機能ではサーバを設定する必要は特にありませんが、より大きなページサイズやネットワークパケットサイズにより、パフォーマンスは大幅に向上します。クライアントメモリに応じて、バッチサイズを大きくすることでもパフォーマンスが向上します。

サポートされている ASEBulkCopy オプション

ASEBulkCopy のオプション	サポートされているバルクロードモード
デフォルト	配列挿入、バルクコピー、オフ
KeepIdentity	配列挿入、バルクコピー、オフ
KeepNulls	配列挿入、バルクコピー、オフ

ASEBulkCopy のオプション	サポートされているバルクロードモード
UseInternalTransaction	配列挿入、バルクコピー、オフ
CheckConstraints	オフ
FireTriggers	オフ
TableLock	サポートされていない

制限事項

- 計算カラムと暗号化カラムはサポートされていません。 また、バルクロード用に選択されたテーブルでは、トリガは無視されます。
- AseBulkCopy の CheckConstraints、FireTriggers、および TableLock オプションは、デフォルト値としてのみサポートされています。バルクロードの無効化時にはこれらの値はサポートされていません。

ENABLEBULKLOAD C 接続プロパティ

`ENABLEBULKLOAD` 接続プロパティを使用して、バルクロードサポートの有効と無効を切り替えることができます。

`ENABLEBULKLOAD` の設定可能値は次のとおりです。

- 0 - オフモード。デフォルト値。
- 1 - **array insert** を使用したバルクロードが有効。
- 2 - **bulk copy** インタフェースを使用したバルクロードが有効。
- 3 - 高速ログ出力 **bulk copy** インタフェースを使用したバルクロードが有効。

ADO.NET 接続文字列を使用したバルクロードの有効化

ADO .NET 接続文字列を使用してバルクロードを有効化する手順を確認します。

1. SQLDriverConnect を使用して、接続文字列を指定します。
2. **ENABLEBULKLOAD** 接続文字列プロパティを、必要に応じて 0、1、または 2 に設定します。

   次に例を示します。

```
Data Source=server1;port=port1;UID=sa;PWD=;
Driver=AdaptiveServerEnterprise;
ENABLEBULKLOAD=1;
```

プライマリキー値の取得

更新するテーブルに自動インクリメントプライマリキーが存在する場合、またはプライマリキーがプライマリキープールから取得される場合は、ストアドプロシージャを使用して、データソースで生成された値を取得します。

AseDataAdapter を使用する場合、この手法を利用して、データソースで生成されるプライマリキー値を DataSet のカラムに書き込むことができます。
`AseCommand` オブジェクトでこの手法を使用するには、パラメータからキーカラムを取得するか、DataReader を再オープンします。

テーブル「adodotnet_primarykey」を使用します。このテーブルには、「id」と「name」という 2 つのカラムがあります。 テーブルのプライマリキーは「id」で、自動インクリメント値を含む NUMERIC(8) です。name カラムは CHAR(40) です。

以下の例では、次のストアドプロシージャを呼び出して自動インクリメントプライマリキー値をデータベースから取得します。

```
create procedure sp_adodotnet_primarykey
@p_name char(40),
@p_id int output
as
begin
   insert into adodotnet_primarykey(name)
      VALUES(@p_name)
   select @p_id = @@identity
END
```

AseCommand オブジェクトを使用した、自動インクリメントプライマリキーを持つ新しいローの挿入

AseCommand オブジェクトを使用して、自動インクリメントプライマリキーを持つ新しいローを挿入します。

1. データベースに接続します。

   C# の場合:

   ```
   AseConnection conn = new AseConnection( c_connStr );
   conn.Open();
   ```

   Visual Basic .NET の場合:

   ```
   Dim conn As New AseConnection( _
      c_connStr )
   conn.Open()
   ```

2. `DataTable` に新しいローを挿入する `AseCommand` オブジェクトを新規作成します。 次のコードでは、行 `int id1 = ( int ) parmId.Value;` によってローのプライマリキー値を検証します。

   C# の場合:

```
AseCommand      cmd = conn.CreateCommand();
cmd.CommandText = "sp_adodotnet_primarykey";
cmd.CommandType = CommandType.StoredProcedure;
AseParameter parmId = new AseParameter(
   "@p_id", AseDbType.Integer);
parmId.Direction = ParameterDirection.Output;
cmd.Parameters.Add( parmId );
AseParameter parmName = new AseParameter(
   "@p_name", AseDbType.Char );
parmName.Direction = ParameterDirection.Input;
cmd.Parameters.Add( parmName );
parmName.Value = "R & D --- Command";
cmd.ExecuteNonQuery();
int id1 = ( int ) parmId.Value;
parmName.Value = "Marketing --- Command";
cmd.ExecuteNonQuery();
int id2 = ( int ) parmId.Value;
parmName.Value = "Sales --- Command"; cmd.ExecuteNonQuery();
int id3 = ( int ) parmId.Value;
parmName.Value = "Shipping --- Command"; cmd.ExecuteNonQuery();
int id4 = ( int ) parmId.Value;
```

Visual Basic .NET の場合:

```
Dim cmd As AseCommand = conn.CreateCommand()
cmd.CommandText = "sp_adodotnet_primarykey"
cmd.CommandType = CommandType.StoredProcedure
Dim parmId As New AseParameter("@p_id", _        AseDbType.Integer)
parmId.Direction = ParameterDirection.Output
cmd.Parameters.Add( parmId )
Dim parmName As New AseParameter("@p_name", _
   AseDbType.Char)
parmName.Direction = ParameterDirection.Input
cmd.Parameters.Add(parmName )

parmName.Value = "R & D --- Command"
cmd.ExecuteNonQuery()
Dim id1 As Integer = parmId.Value
parmName.Value = "Marketing --- Command"
cmd.ExecuteNonQuery()
Dim id2 As Integer = parmId.Value
parmName.Value = "Sales --- Command"
cmd.ExecuteNonQuery()
Dim id3 As Integer = parmId.Value
parmName.Value = "Shipping --- Command"
cmd.ExecuteNonQuery()
dim id4 As Integer = parmId.Value
```

3. 結果をウィンドウのグリッドにバインドして、変更をデータベースに適用します。

   C# の場合:

```
cmd.CommandText = "select * from " +
   "adodotnet_primarykey";
```

```
cmd.CommandType = CommandType.Text;
AseDataReader dr = cmd.ExecuteReader(); dataGrid.DataSource = dr;
```

Visual Basic .NET の場合:

```
cmd.CommandText = "select * from " + _
   "adodotnet_primarykey"
cmd.CommandType = CommandType.Text
Dim dr As AseDataReader = cmd.ExecuteReader()
dataGrid.DataSource = dr
```

4. 接続をクローズします。

   C# の場合:

   ```
   conn.Close();
   ```

   Visual Basic .NET の場合:

   ```
   conn.Close()
   ```

AseDataAdapter オブジェクトを使用した、自動インクリメントプライマリキーを持つ新しいローの挿入

AseDataAdapter オブジェクトを使用して、自動インクリメントプライマリキーを持つ新しいローを挿入します。

1. 新しい AseDataAdapter を作成します。

   C# の場合:

   ```
   AseConnection conn = new AseConnection(
      c_connStr );
   conn.Open();
   DataSet dataSet = new DataSet();
   AseDataAdapter adapter = new AseDataAdapter();
   adapter.MissingMappingAction =
      MissingMappingAction.Passthrough;
   adapter.MissingSchemaAction =
      MissingSchemaAction.AddWithKey;
   ```

   Visual Basic .NET の場合:

   ```
   Dim conn As New AseConnection( _
      c_connStr )
   conn.Open()
   Dim dataSet As New DataSet()
   Dim adapter As New AseDataAdapter()
   adapter.MissingMappingAction = _
      MissingMappingAction.Passthrough
   adapter.MissingSchemaAction = _
      MissingSchemaAction.AddWithKey
   ```

2. データと DataSet のスキーマを書き込みます。 次のコードでは、**AseDataAdapter.Fill** メソッドによって `SelectCommand` を呼び出して、この操

作を実行します。 既存のレコードが不要な場合は、**Fill** メソッドと `SelectCommand` を使用せずに、DataSet を手動で作成することもできます。

C# の場合:

```
adapter.SelectCommand = new AseCommand(
   "select * from adodotnet_primarykey",
   conn );
```

Visual Basic .NET の場合:

```
adapter.SelectCommand = New AseCommand( _
   "select * from adodotnet_primarykey", conn )
```

3. 新しい **AseCommand** を作成して、データベースからプライマリキー値を取得します。

C# の場合:

```
adapter.InsertCommand = new AseCommand(
   "sp_adodotnet_primarykey", conn );
adapter.InsertCommand.CommandType =
   CommandType.StoredProcedure;
adapter.InsertCommand.UpdatedRowSource =
   UpdateRowSource.OutputParameters;
AseParameter parmId = new AseParameter(
   "@p_id", AseDbType.Integer);
parmId.Direction = ParameterDirection.Output;
parmId.SourceColumn = "id";
parmId.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parmId );
AseParameter parmName = new AseParameter(
   "@p_name", AseDbType.Char);
parmName.Direction = ParameterDirection.Input;
parmName.SourceColumn = "name";
parmName.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parmName );
```

Visual Basic .NET の場合:

```
adapter.InsertCommand = new AseCommand( _
   "sp_adodotnet_primarykey", conn )
adapter.InsertCommand.CommandType = _
   CommandType.StoredProcedure
adapter.InsertCommand.UpdatedRowSource = _
   UpdateRowSource.OutputParameters
Dim parmId As New AseParameter( _
   "@p_id", AseDbType.Integer)
parmId.Direction = ParameterDirection.Output
parmId.SourceColumn = "id"
parmId.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parmId )
Dim parmName As New AseParameter( _
   "@p_name", AseDbType.Char)
parmName.Direction = ParameterDirection.Input
parmName.SourceColumn = "name"
```

```
parmName.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parmName )
```

4. `DataSet` に書き込みます。

   C# の場合:

   ```
   adapter.Fill( dataSet );
   ```

   Visual Basic .NET の場合:

   ```
   adapter.Fill( dataSet )
   ```

5. `DataSet` に新しいローを挿入します。

   C# の場合:

   ```
   DataRow row = dataSet.Tables[0].NewRow();
   row[0] = -1;
   row[1] = "R & D --- Adapter";
   dataSet.Tables[0].Rows.Add( row );
   row = dataSet.Tables[0].NewRow();
   row[0] = -2;
   row[1] = "Marketing --- Adapter";
   dataSet.Tables[0].Rows.Add( row );
   row = dataSet.Tables[0].NewRow();
   row[0] = -3;
   row[1] = "Sales --- Adapter";
   dataSet.Tables[0].Rows.Add( row );
   row = dataSet.Tables[0].NewRow();
   row[0] = -4;
   row[1] = "Shipping --- Adapter";
   dataSet.Tables[0].Rows.Add( row );
   ```

   Visual Basic .NET の場合:

   ```
   Dim row As DataRow = dataSet.Tables(0).NewRow()
   row(0) = -1
   row(1) = "R & D --- Adapter"
   dataSet.Tables(0).Rows.Add( row )
   row = dataSet.Tables(0).NewRow()
   row(0) = -2
   row(1) = "Marketing --- Adapter"
   dataSet.Tables(0).Rows.Add( row )
   row = dataSet.Tables(0).NewRow()
   row(0) = -3
   row(1) = "Sales --- Adapter"
   dataSet.Tables(0).Rows.Add( row )
   row = dataSet.Tables(0).NewRow()
   row(0) = -4
   row(1) = "Shipping --- Adapter"
   dataSet.Tables(0).Rows.Add( row )
   ```

6. `DataSet` に加えられた変更をデータベースに適用します。**Update()** メソッドが呼び出されると、プライマリキー値が、データベースから取得された値に変わります。

C# の場合:

```
adapter.Update( dataSet );
dataGrid.DataSource = dataSet.Tables[0];
```

Visual Basic .NET の場合:

```
adapter.Update( dataSet )
dataGrid.DataSource = dataSet.Tables(0)
```

`DataTable` に新しいローを追加して **Update** メソッドを呼び出すと、AseDataAdapter は `InsertCommand` を呼び出して、追加された新しいローそれぞれのキーカラムに出力パラメータをマップします。 **Update** メソッドが呼び出されるのは 1 回だけですが、`InsertCommand` は、追加される新しいローごとに必要な回数だけ **Update** メソッドによって呼び出されます。

7. データベースへの接続をクローズします。

   C# の場合:

   ```
   conn.Close();
   ```

   Visual Basic .NET の場合:

   ```
   conn.Close()
   ```

BLOB の処理

長い文字列値またはバイナリデータをフェッチする場合、データを分割してフェッチできるメソッドがあります。

バイナリデータには `GetBytes` メソッドを、文字列データには `GetChars` メソッドを使用します。 これらを使用しない場合、BLOB データはデータベースからフェッチされるその他のデータと同様の方法で処理されます。

参照：

- GetBytes メソッド (171 ページ)
- GetChar メソッド (173 ページ)

GetChars メソッドを使用して文字列を返すコマンドの発行

`GetChars` メソッドを使用して、文字列を返すコマンドを発行します。

1. Connection オブジェクトを宣言して初期化します。
2. 接続をオープンします。
3. Command オブジェクトを追加して、SQL 文を定義し、実行します。

   C# の場合:

   ```
   AseCommand cmd = new AseCommand(
      "select au_id, copy from blurbs", conn );
   ```

Visual Basic .NET の場合:

```
Dim cmd As New AseCommand( _
   "select au_id, copy from blurbs", conn)
```

4. [ExecuteReader] メソッドを呼び出して、DataReader オブジェクトを返します。

C# の場合:

```
AseDataReader reader = cmd.ExecuteReader();
```

Visual Basic .NET の場合:

```
Dim reader As AseDataReader = cmd.ExecuteReader()
```

次のコードは、結果セットから 2 つのカラムを読み込みます。 最初のカラムは varchar で、2 番目のカラムは Text です。 GetChars は、Text カラムから 1 度に 100 文字を読み込みます。

C# の場合:

```
int length = 100;
char[] buf = new char[ length ];
String au_id;
long dataIndex = 0;
long charsRead = 0;
long blobLength = 0;
while( reader.Read() )
{
   au_id = reader.GetString(0);
   do
   {
      charsRead = reader.GetChars(
         1, dataIndex, buf, 0, length);
      dataIndex += length;
      // do something with the chars read
      //.... some code
      //
      // reinitialize char array
      buf = new char[ length ];
   } while ( charsRead == length );
   blobLength = dataIndex + charsRead;
}
```

Visual Basic .NET の場合:

```
Dim length As Integer = 100
Dim buf(length) As Char
Dim au_id As String
Dim dataIndex As Long = 0
Dim charsRead As Long = 0
Dim blobLength As Long = 0
While reader.Read()
   au_id = reader.GetString(0)
   Do
```

```
      charsRead = reader.GetChars( _
         1, dataIndex, buf, 0, length)
      dataIndex = dataIndex + length
      ' do something with the data read
      '
      '  use code
      '
      ' reinitialize the char array
      ReDim buf(length)
   Loop While (charsRead = length)
   blobLength = dataIndex + charsRead
End While
```

5. DataReader オブジェクトと Connection オブジェクトをクローズします。

   C# の場合:

   ```
   reader.Close();
   conn.Close();
   ```

   Visual Basic .NET の場合:

   ```
   reader.Close()
   conn.Close()
   ```

時刻値の取得

.NET Framework には、Time 構造体がないため、Adaptive Server から時刻値をフェッチする場合は GetDateTime() メソッドを使用します。 このメソッドを使用すると、.NET Framework の DateTime オブジェクトとしてデータが返されます。

GetDateTime メソッドを使用した時刻値の変換

GetDateTime メソッドを使用して時刻値を変換します。

1. Connection オブジェクトを宣言して初期化します。

   C# の場合:

   ```
   AseConnection conn = new AseConnection(
      c_connStr );
   ```

   Visual Basic .NET の場合:

   ```
   Dim conn As New AseConnection( _
      c_connStr )
   ```

2. 接続をオープンします。

   C# の場合:

   ```
   conn.Open();
   ```

   Visual Basic .NET の場合:

```
conn.Open()
```

3. `Command` オブジェクトを追加して、SQL 文を定義し、実行します。

   C# の場合:

   ```
   AseCommand cmd = new AseCommand(
      "SELECT title_id, title, pubdate FROM titles",
      conn );
   ```

   Visual Basic .NET の場合:

   ```
   Dim cmd As New AseCommand( _
      "SELECT title_id, title, pubdate FROM titles", _
      conn)
   ```

4. `ExecuteReader` メソッドを呼び出して、`DataReader` オブジェクトを返します。

   C# の場合:

   ```
   AseDataReader reader = cmd.ExecuteReader();
   ```

   Visual Basic .NET の場合:

   ```
   Dim reader As AseDataReader = cmd.ExecuteReader()
   ```

   次のコードは、**GetDateTime** メソッドを使用して、`DateTime` 値を返します。

   C# の場合:

   ```
   while ( reader.Read() )
   {
      String tid = reader.GetString(0);
      String title = reader.GetString(1);
      DateTime time = reader.GetDateTime(2);
      // do something with the data
   }
   ```

   Visual Basic .NET の場合:

   ```
   While reader.Read()
      Dim tid As String = reader.GetString(0)
      Dim title As String = reader.GetString(1)
      Dim time As DateTime = reader.GetDateTime(2)
      ' do something with the data....
   End While
   ```

5. `DataReader` オブジェクトと `Connection` オブジェクトをクローズします。

   C# の場合:

   ```
   reader.Close();
   conn.Close();
   ```

   Visual Basic .NET の場合:

```
reader.Close()
conn.Close()
```

ストアドプロシージャ

Adaptive Server ADO.NET Data Provider ではストアドプロシージャを使用できます。 結果セットを返すストアドプロシージャを呼び出す場合は、`ExecuteReader` メソッドを使用します。

注意： ストアドプロシージャを使用してデータベースからデータを取得する場合、ストアドプロシージャから出力パラメータ値と結果セットの両方が返されると、結果セットはリセットされ、出力パラメータ値が参照されると同時に結果セットのローは参照できなくなります。 Sybase では、このような場合、結果セットのローすべてを参照して使い終わるまで、参照側の出力パラメータ値を最後までそのままにしておくことをおすすめします。

結果セットを返さないストアドプロシージャを呼び出す場合は、`ExecuteNonQuery` メソッドを使用します。 単一の値のみを返すストアドプロシージャを呼び出す場合は、`ExecuteScalar` メソッドを使用します。

ストアドプロシージャでパラメータが必要な場合は、対応する `AseParameter` オブジェクトを作成する必要があります。 `CommandType` を **StoredProcedure** に指定した場合は、`CommandText` をストアドプロシージャの名前に設定してください。 次に例を示します。

```
sp_producttype
```

参照：

- AseParameter クラス (203 ページ)

ストアドプロシージャの実行

ストアドプロシージャを実行します。

1. `AseConnection` オブジェクトを宣言して初期化します。

   C# の場合:

   ```
   AseConnection conn = new AseConnection(
      c_connStr );
   ```

   Visual Basic .NET の場合:

   ```
   Dim conn As New AseConnection( _
      c_connStr )
   ```

2. 接続をオープンします。

C# の場合:

```
conn.Open();
```

Visual Basic .NET の場合:

```
conn.Open()
```

3. SQL 文を定義して実行する AseCommand オブジェクトを追加します。 次のコードは、CommandType プロパティを使用して、コマンドがストアドプロシージャであることを識別します。

   C# の場合:

   ```
   AseCommand cmd = new AseCommand(
      "titleid_proc", conn );
   cmd.CommandType = CommandType.StoredProcedure;
   ```

   Visual Basic .NET の場合:

   ```
   Dim cmd As New AseCommand( _
      "titleid_proc", conn )
   cmd.CommandType = CommandType.StoredProcedure
   ```

4. AseParameter オブジェクトを追加して、ストアドプロシージャのパラメータを定義します。 ストアドプロシージャに必要なパラメータそれぞれに、新しく AseParameter オブジェクトを作成してください。

   C# の場合:

   ```
   AseParameter param = cmd.CreateParameter();
   param.ParameterName = "@title_id";
   param.AseDbType = AseDbType.VarChar;
   param.Direction = ParameterDirection.Input;
   param.Value = "BU";
   cmd.Parameters.Add( param );
   ```

   Visual Basic .NET の場合:

   ```
   Dim param As AseParameter = cmd.CreateParameter()
   param.ParameterName = "@title_id"
   param.AseDbType = AseDbType.VarChar
   param.Direction = ParameterDirection.Input
   param.Value = "BU"
   cmd.Parameters.Add( param )
   ```

   Parameter オブジェクトの詳細については、「AseParameter クラス」を参照してください。

5. ExecuteReader メソッドを呼び出して、DataReader オブジェクトを返します。 **Get** メソッドを使用して、結果を任意のデータ型で返します。

   C# の場合:

```
AseDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
   string title = reader.GetString(0);
   string id = reader.GetString(1);
   decimal price = reader.GetDecimal(2);
   // do something with the data....
}
```

Visual Basic .NET の場合:

```
Dim reader As AseDataReader = cmd.ExecuteReader()
While reader.Read()
   Dim title As String = reader.GetString(0)
   Dim id As String = reader.GetString(1)
   Dim price As Decimal = reader.GetDecimal(2)
   ' do something with the data....
End While
```

6. `AseDataReader` オブジェクトと `AseConnection` オブジェクトをクローズします。

   C# の場合:

   ```
   reader.Close();
   conn.Close();
   ```

   Visual Basic .NET の場合:

   ```
   reader.Close()
   conn.Close()
   ```

参照：

- AseParameter クラス (203 ページ)

ストアドプロシージャを呼び出すもう 1 つの方法

次のように、呼び出し構文を使用してストアドプロシージャを呼び出すこともできます。 この構文は ODBC および JDBC と互換性があります。

次に例を示します。

```
AseCommand cmd = new AseCommand("{ call sp_product_info(?) }", conn);
```

この場合、コマンドの種類を `CommandType.StoredProcedure` に設定しないでください。 この構文は、名前付きパラメータを使用しないで、`AseCommand.NamedParameters` プロパティを「false」に設定している場合に使用できます。

参照：

- AseCommand オブジェクトを使用したデータの取得 (46 ページ)

- AseCommand オブジェクトを使用したローの挿入、更新、削除 (51 ページ)

トランザクション処理

Adaptive Server ADO.NET Data Provider では、`AseTransaction` オブジェクトを使用して複数の文をグループ化できます。

それぞれのトランザクションは、変更を永続的にデータベースに適用する **COMMIT**、またはトランザクションの操作すべてを取り消す **ROLLBACK** で終了します。 トランザクションが完了した後、さらに変更を行う場合は、新しく `AseTransaction` オブジェクトを作成してください。 この動作は、ODBC や Embedded SQL の場合とは異なり、**COMMIT** または **ROLLBACK** の実行後も、トランザクションがクローズされるまで存続します。

トランザクションを作成しない場合、Adaptive Server ADO.NET Data Provider はデフォルトで **autocommit** モードで動作します。 **insert**、**update**、または **delete** が実行されるたびに **Commit** が暗黙的に実行され、操作が完了すると変更がデータベースに反映されます。 この場合、変更はロールバックできません。

参照：

- AseTransaction クラス (222 ページ)

トランザクションの独立性レベルの設定

トランザクションの開始時に独立性レベルを指定するように選択できます。 この独立性レベルは、そのトランザクション内で実行されるすべてのコマンドに適用されます。

独立性レベルの詳細については、『Adaptive Server Enterprise パフォーマンス & チューニングガイド』を参照してください。

Select 文を入力したときに Adaptive Server で使用されるロックは、トランザクションの独立性レベルに応じて異なります。

次の例は、`AseTransaction` オブジェクトを使用して SQL 文を発行してからロールバックを実行します。 このトランザクションは独立性レベル 2 (RepeatableRead) を使用し、変更中のローに **Write** ロックを適用して、他のデータベースユーザがローを更新できないようにします。

AseTransaction オブジェクトを使用したコマンドの発行

`AseTransaction` オブジェクトを使用してコマンドを発行します。

1. `AseConnection` オブジェクトを宣言して初期化します。

   C# の場合:

```
AseConnection conn = new AseConnection(
   c_connStr );
```

Visual Basic .NET の場合:

```
Dim conn As New AseConnection( _
   c_connStr )
```

2. 接続をオープンします。

   C# の場合:

   ```
   conn.Open();
   ```

   Visual Basic .NET の場合:

   ```
   conn.Open()
   ```

3. 「T シャツ (Tee shirts)」の価格を変更する SQL 文を発行します。

   C# の場合:

   ```
   string stmt = "update product " +
      " set unit_price = 2000.00 " +
      " where name = 'Tee shirt'";
   ```

   Visual Basic .NET の場合:

   ```
   Dim stmt As String = "update product " + _
      " set unit_price = 2000.00 " + _
      " where name = 'Tee shirt'"
   ```

4. `Command` オブジェクトを使用して SQL 文を発行する `AseTransaction` オブジェクトを作成します。

   トランザクションを使用することで、独立性レベルを指定できるようになります。 この例では独立性レベル 2 (RepeatableRead) を使用して、他のデータベースユーザがローを更新できないようにします。

   C# の場合:

   ```
   AseTransaction trans = conn.BeginTransaction(
      IsolationLevel.RepeatableRead );
   AseCommand cmd = new AseCommand( stmt, conn, trans );
   int rows = cmd.ExecuteNonQuery();
   ```

   Visual Basic .NET の場合:

   ```
   Dim trans As AseTransaction = _
      conn.BeginTransaction( _
      IsolationLevel.RepeatableRead )
   Dim cmd As New AseCommand( _
      stmt, conn, trans )
   Dim rows As Integer = cmd.ExecuteNonQuery()
   ```

5. 変更をロールバックします。

   C# の場合:

```
trans.Rollback();
```

Visual Basic .NET の場合:

```
trans.Rollback()
```

`AseTransaction` オブジェクトを使用すると、データベースに対する変更をコミットしたりロールバックしたりできます。 トランザクションを使用しない場合、Adaptive Server ADO.NET Data Provider は **autocommit** モードで動作するため、データベースに加えた変更をロールバックできなくなります。 変更を永続的に残すには、次のように指定します。

C# の場合:

```
trans.Commit();
```

Visual Basic .NET の場合:

```
trans.Commit()
```

6. `AseConnection` オブジェクトをクローズします。

C# の場合:

```
conn.Close();
```

Visual Basic .NET の場合:

```
conn.Close()
```

COMPUTE 句を含む Transact-SQL クエリのサポート

Adaptive Server ADO.NET Data Provider は、**COMPUTE** 句を含む Transact-SQL クエリをサポートします。

COMPUTE 句を使用すると、単一の **select** 文にディテールと計算結果を組み込むことができます。 計算ローは、次に示すように特定グループのディテールローの後に表示されます。

```
select type, price, advance from titles order by type compute
sum(price), sum(advance) by type
```

```
type          price      advance
------------  ---------  ----------
UNDECIDED     NULL       NULL
Compute Result:
----------------------- ------------------------
NULL                     NULL
type          price      advance
------------  ---------  ----------
business      2.99       10,125.00
business      11.95      5,000.00
business      19.99      5,000.00
business      19.99      5,000.00
```

```
Compute Result:
----------------------- ------------------------
54.92                   25,125.00
...
...

(24 rows affected)
```

Adaptive Server ADO.NET Data Provider が **COMPUTE** 句を含む **select** 文を実行すると、複数の結果セットがクライアントに返されます。 結果セットの数は、使用できるユニークなグループの数によって異なります。 各グループには、ディテールローの結果セットが 1 つと計算結果の結果セットが 1 つ組み込まれます。 クライアントが返されたローを完全に処理するには、すべての結果セットを処理する必要があります。このように処理しない場合は、最初のデータグループのディテールローのみが、最初に返される結果セットに組み込まれます。 **COMPUTE** 句の詳細については、『Adaptive Server Enterprise Transact-SQL ユーザーズガイド』を参照してください。

エラー処理

アプリケーションは、ADO.NET エラーも含め、発生するすべてのエラーを処理できるように設計する必要があります。

実行時にエラーが発生すると、Adaptive Server ADO.NET Data Provider は `AseException` オブジェクトを返します。 各 `AseException` オブジェクトは `AseError` オブジェクトのリストで構成され、これらのエラーオブジェクトにはエラーメッセージとコードが組み込まれています。 これ以外に、`IndexOutOfRangeException` や `NotSupportedException` などの例外も発生することがあります。

エラーは、変更をデータベースに適用するときに発生する競合とは異なります。アプリケーションには、競合が発生した場合に正しい値を計算したりログに記録したりする処理が必要です。

Adaptive Server ADO.NET Data Provider のエラー処理の例

次の例は、Simple サンプルプロジェクトにもとづくものです。

実行時に発生したエラーや Adaptive Server ADO.NET Data Provider オブジェクトのエラーはすべて、メッセージボックスに表示されて処理されます。 次のコードは、エラーを検出してメッセージを表示します。

C# の場合:

```
catch( AseException ex )
{
```

```
   MessageBox.Show( ex.Message );
}
```

Visual Basic .NET の場合:

```
Catch ex As AseException
   MessageBox.Show(ex.Message)
End Try
```

接続エラーの処理例
次の例は、Table Viewer サンプルプロジェクトにもとづくものです。アプリケーションがデータベースへの接続を試行しているときにエラーが発生した場合、次のコードは try-catch ブロックを使用してエラーを検出し、メッセージを表示します。

C# の場合:

```
try
{
   _conn = new AseConnection(
      txtConnectString.Text );
   _conn.Open();
}
catch( AseException ex )
{
   MessageBox.Show(ex.Message, "Failed to connect");
}
```

Visual Basic .NET の場合:

```
Try
   Dim  _conn As New AseConnection( _
      txtConnectString.Text )
   conn.Open()
Catch ex As AseException
   MessageBox.Show(ex.Message, "Failed to connect")
End Try
```

参照：

- Simple サンプルプロジェクトの理解 (13 ページ)
- Table Viewer サンプルプロジェクトの理解 (19 ページ)
- AseException クラス (200 ページ)
- AseError クラス (194 ページ)

パフォーマンスの考慮事項

Adaptive Server ADO.NET Data Provider を使用してアプリケーションの開発と配備を行う際に役立つヒントを確認します。

DbType.String と DbType.AnsiString

`DbType.String` と `DbType.AnsiString` は、いずれも文字データを処理しますが、処理方法がそれぞれに異なるため、不適切なデータ型を使用すると、アプリケーションのパフォーマンスに悪影響を及ぼす可能性があります。

`DbType.String` では、パラメータは 2 バイトの Unicode 値として識別されて、サーバに送信されます。 `DbType.AnsiString` では、パラメータはマルチバイトの文字列として送信されます。 不要な文字列変換を避けるため、次のように使用します。

- `char` または `varchar` のカラムおよびパラメータに対しては、`DbType.AnsiString` を使用します。
- `unichar` および `univarchar` のカラムおよびパラメータに対しては、`DbType.String` 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

複数の Adaptive Server が共有のディスクセットと高速プライベート相互接続に接続する、クラスタエディション環境をサポートする Adaptive Server ADO .NET ドライバの機能を使用すると、複数の物理および論理ホストを使用して Adaptive Server をスケーリングできます。

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

ログインリダイレクト

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。

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。

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機能を使用すると、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、可用性の高い「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

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。 クラスタ環境では、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ます。

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、常に最適なフェールオーバリストを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

HASession 接続文字列プロパティを 1 に設定して、クラスタエディションの接続フェールオーバを有効化します。

次に例を示します。

```
Data Source=server1;Port=port1;User ID=sa;Password=;
Initial Catalog=sdc;HASession=1;
AlternateServers=server2:port2,...,serverN:portN;
```

この例では、Data Source がプライマリサーバとポートを定義します。 Sybase が提供する ADO.NET Provider は、最初にプライマリサーバへの接続を試行します。接続に失敗した場合は、AlternateServers にリストされているサーバへの接続を順次試行します。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： 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。 使用可能ないずれかの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の最新リストをサーバから受信します。 この新しいリストで、指定されたリストが上書きされます。

分散トランザクション

Adaptive Server ADO.NET Data Provider を使用し、それを 2 フェーズコミットトランザクションに含めることができます。 この機能では、分散トランザクションを管理する .NET Enterprise Services を使用する必要があります。

Enterprise Services を使用するプログラミング

アンマネージコード内のサービスは、COM+ サービスと呼ばれます。 COM+ サービスインフラストラクチャには、マネージコードとアンマネージコードからアクセスできます。 .NET では、これらのサービスが Enterprise Services と呼ばれます。 ADO.NET を使用する Enterprise Services 内でトランザクションを扱うのは簡単です。

1. `System.EntrepriseService.ServicedComponent` からコンポーネントを抽出します。
2. 必要なサービスとそのオプションを指定するための独自の属性 (`Transaction`、`AutoComplete` など) を指定します。 属性の全リストについては、Enterprise Services のマニュアルを参照してください。

   注意： .NET トランザクション属性の `Timeout Option` には、明示的に *-1* または非常に大きな数を設定する必要があります。 .NET のマニュアルでは、ADO.NET トランザクションのデフォルトのタイムアウト値が *0* であることが記載されていますが、これはタイムアウトしないことを意味します。 ただし、実際には即時にトランザクションがタイムアウトし、トランザクション全体がロールバックされます。
3. アセンブリに署名し、ビルドします。
4. アセンブリを登録します。

分散トランザクションサポートの接続プロパティ

分散トランザクションサポートで使用される接続プロパティです。

- 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、XA インタフェース標準、MS DTC OLE ネイティブプロトコルのサポートに使用するプロトコルを指定し、接続文字列でプロパティ `DistributedTransactionProtocol=`*OLE* ネイティブプロトコルを設定します。 デフォルトのプロトコルは *XA* です。
- 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」と呼ばれる状態になります。 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` を設定する必要があります。

- 登録 - `AseConnection` オブジェクトは、トランザクションがアクティブであることを特定した場合に、既存の分散トランザクションに自動的に登録します。 接続を開くか、接続プールから取得したときに、自動トランザクション登録が行われます。 `AseConnection` の接続文字列パラメータとして **Enlist=0** を指定することにより、この自動登録機能を無効にできます。
  自動登録が無効である場合、`AseConnection` で既存トランザクションへの参照である `ITransaction` パラメータとともに **EnlistDistributedTransaction** メソッドを呼び出すことによって、既存の分散トランザクションに登録できます。 **EnlistDistributedTransaction** を呼び出した後、`AseConnection` のこのインスタンスを使用したすべての更新は、このグローバルトランザクションの一部として適用されます。 したがって、グローバルトランザクションがコミットまたはロールバックされると、それに伴ってこのトランザクションもコミットまたはロールバックされます。

注意： AseConnection オブジェクトは、EnlistDistributedTransaction を呼び出す前にオープンである必要があります。

ビジネスオブジェクトをプールする場合、**EnlistDistributedTransaction** を使用できます。 ビジネスオブジェクトが開いている接続とともにプールされている場合、その接続が開いているか、または接続プールから取得される場合に限り、自動トランザクション登録が行われます。 プールされたビジネスオブジェクトにより複数のトランザクションが実行される場合、そのオブジェクトの開いている接続は新しく開始したトランザクションで自動的に登録されません。 この場合は、`AseConnection` の自動トランザクション登録を無効にした後、**EnlistDistributedTransaction** を使用して `AseConnection` をトランザクションに登録できます。

警告！ `AseConnection` が **BeginTransaction** を使用するか、または `AseCommand` で **BEGIN TRANSACTION** 文を明示的に実行することによりすでにトランザクションを開始している場合、**EnlistDistributedTransaction** は例外を返します。

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 ADO.NET Data Provider は一元的な LDAP サーバから接続やその他の情報を取得して Adaptive Server サーバに接

続できます。ここでは、DSURL (Directory Service URL) 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の格納および管理を行います。
LDAP サーバと Adaptive Server やクライアントのプラットフォームは異なっていてもかまいません。 LDAP では通信プロトコル、およびクライアント/サーバ間で交換されるメッセージのコンテンツを定義します。 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 - Data Provider は、DSURL または **ConnectString** の *DSPrincipal* と *DSPassword* プロパティに指定されたユーザ名とパスワードを使用します。

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、DSURL プロパティと Servername プロパティを **ConnectString** に追加します。

```
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 は複数の URL を受け入れます。 各 URL は次のようにセミコロンで区切ります。 次に例を示します。

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

DSURL は次のように指定します。

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

構文の説明は次のとおりです。

- *hostport* は、次のような、ホスト名とオプションの `portnumber` です。たとえば、 `SYBLDAP1:389` です。
- *dn* は、`dc=sybase,dc-com` などの検索ベースです。
- *attrs* は、LDAP サーバ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* は検索フィルタです。 通常は、**sybaseServername** です。 ここをブランクにして、**ConnectionString** で Data Source または Server Name プロパティを設定することもできます。
- *userdn* は、ユーザの識別名 (DN: Distinguished Name) 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** で **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** で **DSPassword** プロパティを設定します。

マイクロ秒の精度の time データ

Adaptive Server ADO.NET Data Provider 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 00:00:00.000000 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 以降に接続する場合、Adaptive Server ADO.NET Data Provider 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 は、Adaptive Server カラムに合わせるために、Adaptive Server ADO.NET Data Provider 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 ADO.NET Data Provider は `datetime` および `time` データ型を使用してデータを転送します。

パスワードの暗号化

Adaptive Server ADO.NET Data Provider は、対称および非対称のパスワード暗号化をサポートします。 この機能を使用して、デフォルトの動作を変更してパスワードを暗号化してから、ネットワークに送信することができます。

Adaptive Server ADO.NET Data Provider 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。

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化され、サーバで非対称暗号化がサポートされる場合は、非対称暗号化形式が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して、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化されていると、ログインがネゴシエートされてからのみ、パスワードがネットワークに送信されます。パスワードの暗号化が有効化されてから、パスワードが暗号化され、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は、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は、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。

例

この例では、ログインがネゴシエートされ、パスワードの暗号化と送信が行われるまで、ネットワークに sapass は送信されません。

```
AseConnection.ConnectionString=
”Data Source=MANGO;" +
   "Port = 5000;" +
   "Database=pubs2;" +
   "UID=sa;" +
   "PWD=sapass;" +
   "EncryptPassword=1;”;
```

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発生し、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。

Adaptive Server ADO.NET Data Provider は、**ChangePassword** メソッドをサポートしています。これにより、管理者による操作を必要とせずに、期限切れになったパスワードをアプリケーション側で変更できます。 詳細については、Microsoft Developer Network で ChangePassword メソッドを検索してください。

Secure Sockets Layer (SSL)

SSL は、クライアントからサーバ、およびサーバからサーバへの接続で、ネットワーク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。

SSL ハンドシェイク
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。 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. クライアントは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。
2. サーバは、証明書とサポートされている CipherSuite 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がお互いに CipherSuite に合意すると、安全で暗号化されたセッションが確立されます。

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 の Web サイト (http://www.ietf.org) を参照してください。

パフォーマンス
安全なセッションの確立に必要な、追加のオーバヘッドが生じます。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。

CipherSuites
SSL ハンドシェイクの際は、CipherSuite を介してクライアントとサーバが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 プロトコルで使用するキー交換アルゴリズム、ハッシュ方式、暗号化方式の優先リストです。 CipherSuites の総合的な説明については、IETF 組織の Web サイト (http://www.ietf.org) を参照してください。

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。 サーバ接続属性は、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 ADO.NET Data Provider と 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 Corp 製) で使用可能な CipherSuite 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。 TLS は、SSL 3.0 を拡張したものであり、SSL バージョン 3.0 CipherSuite の別名です。

Adaptive Server ADO.NET Data Provider でサポートしている CipherSuite は、次のとおりです。

- 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

Adaptive Server ADO.NET Data Provider の SSL

SSL には、いくつかのセキュリティレベルがあります。

- 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な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比較して、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 ADO.NET Data Provider が SSL 対応サーバにクライアント接続する場合、サーバは証明書ファイルが必要です。これは、サーバの証明書と暗号化されたプライベートキーで構成されます。

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。 Adaptive Server ADO.NET Data Provider のクライアントアプリケーションが Adaptive Server へのソケット接続を確立する方法は、既存のクライアント接続の確立方法と同じです。 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。

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。

信頼された (Trusted) 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場所を指定します。 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るルートファイルは `%SYBASE%¥ini¥trusted.txt` にインストールされています。

SSL 接続の有効化

Data Provider で SSL を有効化するには、ConnectionString プロパティに Encryption=ssl; TrustedFile=<trusted file> を追加します。

これで **AseConnection** は Adaptive Server と SSL 接続のネゴシエートを行います。

次に例を示します。

```
AseConnection.ConnectionString=
     "Data Source=MANGO;" +
     "Port = 5000;" +
     "Database=pubs2;" +
     "UID=sa;" +
     "PWD=sapass;" +
     "Encryption=ssl;" +
     "TrustedFile='c:¥sybase¥ini¥trusted.txt';";
```

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。

高可用性システムのフェールオーバ

高可用性クラスタには、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) がダウンした場合にもう 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。

このようなマシンのそれぞれを、高可用性クラスタのノードといいます。 一般的に、高可用性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。

フェールオーバによって、Adaptive Server のアクティブ/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の運用が可能になります。

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** プロパティと **SecondaryPort** プロパティも設定します。

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 のマニュアル『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。

トランザクションで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。 フェールオーバが発生すると、プロバイダは、セカンダリサーバへの再接続を試行します。 セカンダリサーバへの接続が確立されると、ADO.NET Data Provider は、フェールオーバの発生を示すメッセージとともに **AseFailoverException** を返します。 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。 セカンダリサーバへの接続が確立できない場合は、接続が失われたことを示すメッセージとともに、ADO.NET Data Provider で通常の **AseException** が発生します。 次に例を示します。

```
AseConnection.ConnectionString =
   ”Data Source='tpsun1';" +
   "Port = 5000;" +
   "Database=pubs2;" +
   "User ID=sa;" +
   "Password=sapass;" +
   "HASession=1;" +
   "Secondary Data Source='tpsun2';" +
   "Secondary Server Port=5000";
```

次のコードは、**AseFailoverException** の検出方法を示しています。

```
....
Open connection
...more code

try
{   using (AseDataReader rdr =
   selectCmd.ExecuteReader())
   {
   ....
   }
}
catch (AseFailoverException)
{
   //Make sure that you catch AseFailoverException
   //before AseException as AseFailoverException is
   //derived from AseException

   //HA has occured. The application has successfully
   //connected to the secondary server. All uncommitted
   //transactions have been rolled back.

   //You could retry your transactions or prompt user
   //for a retry operation
}
catch (AseException)
{
//Either some other problem or the Failover did not
//successfully connect to the secondary server. Apps.
//should react accordingly
}
```

Kerberos 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 を使用すると、非常に安全な環境でさまざまなアプリケーション間のシングルサインオンを実現できます。

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の維持にも暗号化が使用されます。

Adaptive Server と Adaptive Server ADO.NET Data Provider は、Kerberos 接続をサポートします。 具体的にいうと、Adaptive Server ADO.NET Data Provider では、MIT、CyberSafe、Active Directory の KDC (Key Distribution Center) がサポートされます。

Kerberos の処理の概要

Kerberos の認証処理を確認します。

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、ユーザの認証が完了します。
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で暗号化し、それをクライアントに返します。
10. クライアントは、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てパケットを復号化し、サービスがそれ自体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。 これ以降、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。

Kerberos の要件

Kerberos を認証システムとして使用するには、認証が kerberos に委任されるように Adaptive Server Enterprise を設定する必要があります。

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。 Windows では、クライアントライブラリとともに Kerberos クライアントライブラリがインストールされます。

Adaptive Server ADO.NET Data Provider で Kerberos を使用するには、MIT/CyberSafe Client ライブラリを設定し、Adaptive Server で Kerberos を有効にする必要があります。

Kerberos 認証の有効化

Adaptive Server ADO.NET Data Provider で Kerberos 認証を有効化します。
プログラムに次の命令を追加します。

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

ここで、*<ASE server name>* は KDC (Key Distribution Center) で設定された論理名またはプリンシパルです。 Adaptive Server ADO.NET Data Provider 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。 Windows では、**activedirectory** を選択することで追加設定を不要にできます。

Kerberos クライアントライブラリは、さまざまな KDC 間で互換性を持ちます。 たとえば、KDC が Microsoft Active Directory の場合でも、`mitkerberos` と同じ **AuthenticationClient** を設定できます。 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。

```
"Driver=Adaptive Server Enterprise;UID=sa;
PWD='';Server=sampleserver;
Port=4100;Database=pubs2;
AuthenticationClient=mitkerberos;
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。

結果の説明

- **チケットキャッシュ** – 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** – TGT を所有するユーザ (この場合はユーザ自身) 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 時間有効であることに注意してください。

MIT Kerberos クライアントライブラリ用の TGT の生成
MIT Kerberos クライアントライブラリ用の TGT を生成する手順を確認します。

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. my_password など、your_name@YOUR.REALM のパスワードを入力します。 パスワードを入力すると、**kinit** ユーティリティにより TGT (Ticket Granting Ticket) を求め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. コマンドラインに次のように入力して、TGT が取得されていることを確認します。

   ```
   % 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
   ```

SECURECONNECTIONSTRING プロパティ

SECURECONNECTIONSTRING は、Adaptive Server ADO.NET Data Provider の接続プロパティの 1 つであり、接続文字列からパスワードプロパティを削除します。

この接続プロパティは、**AseConnection.ConnectionString** を使用した接続文字列へのアクセス時にパスワードが公開されるのを防ぎます。 値:

- 0 - デフォルト値。Adaptive Server ADO.NET Data Provider は接続文字列にパスワードを維持します。
- 1 - Adaptive Server ADO.NET Data Provider は、接続文字列からパスワードを削除します。

サポートされている Microsoft ADO.NET 2.0、3.0、3.5、および 4.0 の機能

Adaptive Server ADO.NET Data Provider は、Microsoft ADO.NET 2.0、3.0、3.5、および 4.0 の以下の機能をサポートしています。

- プロバイダファクトリ
- プロバイダ統計情報
- バルク更新
- バルクコピー
- 非同期コマンド
- プールのクリアに対応する拡張プールサポート
- 共通ベースクラス
- データベースメタデータ

これらの機能の詳細については、Microsoft Developer Network (http://msdn.microsoft.com) を参照してください。

Adaptive Server 用の Microsoft Enterprise Library DAAB

Adaptive Server ADO.NET Data Provider 2.157 は、Enterprise Library 4.1 DAAB (Data Access Application Block) の機能を拡張して Adaptive Server をサポートします。

DAAB はクラスのコレクションで、データベースインスタンスの作成やデータベースレコードの更新など、一般的なデータベース機能を簡素化します。 また、DAAB は、データベース固有の機能のカプセル化も行います。これにより、データベースを意識しないアプリケーション設計が可能になります。

Adaptive Server クラス用の DAAB は、Microsoft .NET Framework 3.5、および Microsoft Visual Studio 2008 でサポートされます。 Adaptive Server 用 DAAB を使用するには、Microsoft Enterprise Library 4.1 を更新する必要があります。

Adaptive Server ADO.NET Data Provider 用に導入された DAAB アセンブリは、`Sybase.EnterpriseLibrary.AseClient.dll` ファイルとしてインストールに付属しています。 また、このアセンブリのビルド用に導入された DAAB ソースコードをインストールすることもできます。 DAAB ソースコードは、ADO.NET サンプルのインストールを選択したときに、`Sybase.EnterpriseLibrary.AseClient` ディレクトリにインストールされます。

Enterprise Library Data Access Application Block の詳細については、Microsoft Developer Network (http://msdn.microsoft.com) を参照してください。

Microsoft ADO.NET Entity Framework と LINQ

Adaptive Server ADO.NET Data Provider は、Visual Studio Language-Integrated Query (LINQ)、Entity Framework 4.0 で定義されている拡張 Entity Data Model (EDM) の正規の関数、および Microsoft ADO.NET Entity Framework (LINQ-to-SQL コンポーネントを含む) をサポートしています。

ただし、Microsoft ADO.NET Entity Framework の制限により、次の処理はサポートされていません。

- LINQ **Contains** 拡張メソッドの使用。 **Contains** は SQL の **IN** 句に対応します。
- LINQ 拡張メソッドの作成。
- エンティティクラス間の関連付けの作成。

ADO.NET Entity Framework と LINQ の利点としては、リレーショナルストレージスキーマの概念モデルを操作できることが挙げられます。これにより、データ指向アプリケーションの開発および保守に伴う作業を簡素化できます。 Microsoft ADO.NET Entity Framework と LINQ を使用するには、`Sybase.AdoNet2.AseClient.dll` または `Sybase.AdoNet4.AseClient.dll` を参照します。

ADO.NET Entity Framework および LINQ の詳細については、Microsoft Developer Network (http://msdn.microsoft.com) を参照してください。

Adaptive Server ADO.NET Data Provider API リファレンス

Adaptive Server ADO.NET Data Provider の API についての情報を確認します。

最新の API のマニュアルと、Microsoft ADO.NET インタフェースの実装に該当するプロパティとメソッドの詳細については、Adaptive Server ADO.NET のオンラインヘルプを参照してください。 オンラインヘルプにアクセスするには、Microsoft Document Explorer を開いて、[Sybase ASE ADO.NET Data Provider] を選択します。詳細とその使用例については、Microsoft .NET Framework のドキュメントも参照してください。

AseBulkCopy クラス

データソースから Adaptive Server テーブルにデータをコピーします。

実装
IDisposable

AseBulkCopy コンストラクタ

AseBulkCopy クラスの新しいインスタンスを初期化します。

構文 1

```
void AseBulkCopy( AseConnection connection)
```

構文 2

```
void AseBulkCopy( string connString )
```

構文 3

```
void AseBulkCopy( string connString, AseBulkCopyOptions copyOptions)
```

構文 4

```
void AseBulkCopy( AseConnection connection, AseBulkCopyOptions
copyOptions, AseTransaction externalTransaction )
```

パラメータ

- **connection –** バルクコピー操作の実行対象の Adaptive Server 接続。
- **connString – AseBulkCopy** の実行対象の接続に対する接続文字列。

- **copyOptions –** AseBulkCopy オプション。
- **externalTransaction –** Adaptive Server 接続で実行するトランザクション。

参照：
- AseBulkCopyOptions 列挙型 (118 ページ)

Close メソッド

AseBulkCopy オブジェクトで使用されるリソースを解放します。

構文
```
void Close()
```

Dispose メソッド

AseBulkCopy オブジェクトで使用されるリソースを解放します。

構文
```
void Dispose()
```

実装
```
IDisposable.Dispose()
```

Finalize メソッド

AseBulkCopy オブジェクトで使用されるリソースを解放します。

構文
```
void Finalize()
```

WriteToServer メソッド

データソースからコピー先テーブルにデータを書き込みます。

構文 1
```
void WriteToServer( DataRow[ ] rows )
```

構文 2
```
void WriteToServer( DataTable table )
```

構文 3
```
void WriteToServer( IDataReader reader )
```

構文 4

```
void WriteToServer( DataTable table, DataRowState rowState)
```

パラメータ

- **reader – IDataReader**。 インタフェースのデータがデータソースに書き込まれます。
- **rows – DataTable** 内のデータロー。 データがデータソースに書き込まれます。
- **rowState –** コピーの対象にできるデータローの状態を指定します。
- **table – DataTable**。 このテーブルのデータがデータソースに書き込まれます。

BatchSize プロパティ

バッチ内のロー数を表します。

構文

```
int BatchSize
```

アクセス
読み込み/書き込み

BulkCopyTimeout プロパティ

操作がタイムアウトになるまでの実行時間 (秒数) を表します。

構文

```
int BulkCopyTimeout
```

アクセス
読み込み/書き込み

ColumnMappings プロパティ

カラムマッピングのコレクションを表します。

構文

```
AseBulkCopyColumnMappingCollection ColumnMappings
```

アクセス
読み込み専用

DestinationTableName プロパティ

データの書き込み先のテーブル名を表します。

構文

```
string DestinationTableName
```

アクセス
読み込み/書き込み

NotifyAfter プロパティ

AseRowsCopied イベントをトリガする前に処理するロー数を表します。

構文

```
int NotifyAfter
```

アクセス
読み込み/書き込み

AseRowsCopied イベント

NotifyAfter で指定したロー数が処理されるたびに発生します。

構文

```
event AseRowsCopiedEventHandler AseRowsCopied
```

AseBulkCopyColumnMapping クラス

データソース内のカラムをコピー先テーブル内のカラムにマッピングします。

AseBulkCopyColumnMapping コンストラクタ

AseBulkCopyColumnMapping クラスの新しいインスタンスを初期化します。

構文 1

```
void AseBulkCopyColumnMapping()
```

構文 2

```
void AseBulkCopyColumnMapping( int sourceInx, int dbInx )
```

構文 3

```
void AseBulkCopyColumnMapping( string sourceName, string dbName )
```

パラメータ

- **dbInx –** コピー先テーブル内のカラムインデックスを表します。
- **sourceInx –** データソース内のカラムインデックスを表します。
- **dbName –** コピー先テーブル内のカラム名を表します。
- **sourceName –** データソース内のカラム名を表します。

Equals メソッド

2 つのカラムマッピングが同じであるかどうかを判別します。

構文

```
bool Equals( Object obj )
```

パラメータ
obj - 比較対象のオブジェクト。

戻り値
オブジェクトが同じ場合は true、それ以外の場合は false。

DestinationColumn プロパティ

マッピング先のカラム名を表します。

構文

```
string DestinationColumn
```

アクセス
読み込み/書き込み

DestinationOrdinal プロパティ

マッピング先のカラムインデックスを表します。

構文

```
int DestinationOrdinal
```

アクセス
読み込み/書き込み

SourceColumn プロパティ

データソース内のカラム名を表します。

構文

```
string SourceColumn
```

アクセス
読み込み/書き込み

SourceOrdinal プロパティ

データソース内のカラムインデックスを表します。

構文

```
int SourceOrdinal
```

アクセス
読み込み/書き込み

AseBulkCopyColumnMappingCollection クラス

カラムマッピングのコレクションを表します。

AseBulkCopyColumnMappingCollection コンストラクタ

AseBulkCopyColumnMappingCollection クラスの新しいインスタンスを初期化します。

構文

```
void AseBulkCopyColumnMappingCollection()
```

Add メソッド

新しいマッピングをコレクションに追加します。

構文

```
int Add( AseBulkCopyColumnMapping value )
```

パラメータ
value - 追加するカラムマッピング。

戻り値
カラムマッピングの追加先のインデックス。

Contains メソッド

指定されたマッピングに対応するコレクションを検索します。

構文

```
bool Contains( AseBulkCopyColumnMapping value )
```

パラメータ
value - 検索対象のカラムマッピング。

戻り値
指定されたマッピングがコレクションに含まれている場合は true、それ以外の場合は false。

IndexOf メソッド

指定されたマッピングのインデックスを取得します。

構文

```
int IndexOf( AseBulkCopyColumnMapping value )
```

パラメータ
value - インデックスが取得されるカラムマッピング。

戻り値
指定されたマッピングのインデックス。

Insert メソッド

指定されたインデックスでマッピングを挿入します。

構文

```
void Insert( int index, AseBulkCopyColumnMapping value )
```

パラメータ

- **index –** 新しいマッピングの挿入先となるコレクション内の位置。
- **value –** 追加対象のカラムマッピング。

Validate メソッド

カスタムプロセスを実行して、指定されたオブジェクトをコレクションに追加できることを検証します。

構文

```
void OnValidate( Object value )
```

パラメータ

value - 検証対象のオブジェクト。

Remove メソッド

コレクションからマッピングを削除します。

構文

```
void Remove( AseBulkCopyColumnMapping value )
```

パラメータ

value - 削除するカラムマッピング。

AseBulkCopyOptions 列挙型

AseBulkCopy クラスで使用可能な動作オプションを指定します。

構文

```
enum AseBulkCopyOptions
```

メンバ

メンバ名	AseBulkCopy の動作
CheckConstraints	挿入操作時の制約をチェックする。
Default	デフォルトでは、すべてのオプションがオフ。
FireTriggers	挿入操作時のトリガを起動をサーバに指示。
KeepIdentity	データソースの ID 値を使用する。 KeepIdentity が選択されていない場合、Adaptive Server が ID を割り当てる。
KeepNulls	null 値を挿入する。 KeepNulls が選択されていない場合、デフォルト値で null 値が置き換えられる。

メンバ名	AseBulkCopy の動作
TableLock	バルクコピー操作時にテーブル全体をロックする。
UseInternalTransaction	設定した場合、バルクコピーの各バッチはトランザクション内に配置される。

AseClientFactory クラス

Adaptive Server ADO.NET Data Provider データソースクラスのインスタンスの作成に使用できる一連のメソッドを表します。

基本クラス
DbProviderFactory

AseClientFactory コンストラクタ

AseClientFactory クラスの新しいインスタンスを初期化します。

構文

```
void AseClientFactory()
```

Instance フィールド

厳密に型指定されたデータオブジェクトの取得に使用できる AseClientFactory のインスタンスを表します。

CreateCommand メソッド

System.Data.Common.DbCommand クラスを実装する Adaptive Server ADO.NET Data Provider クラスの新しいインスタンスを返します。

構文

```
DbCommand CreateCommand()
```

戻り値
System.Data.Common.DbCommand の新しいインスタンス。

CreateCommandBuilder メソッド

System.Data.Common.DbCommandBuilder クラスを実装する Adaptive Server ADO.NET Data Provider クラスの新しいインスタンスを返します。

構文

```
DbCommandBuilder CreateCommandBuilder()
```

戻り値

System.Data.Common.DbCommandBuilder の新しいインスタンス。

CreateConnection メソッド

System.Data.Common.DbConnection クラスを実装する Adaptive Server ADO.NET Data Provider クラスの新しいインスタンスを返します。

構文

```
DbConnection CreateConnection()
```

戻り値

System.Data.Common.DbConnection の新しいインスタンス。

CreateConnectionStringBuilder メソッド

System.Data.Common.DbConnectionStringBuilder クラスを実装する Adaptive Server ADO.NET Data Provider クラスの新しいインスタンスを返します。

構文

```
DbConnectionStringBuilder CreateConnectionStringBuilder()
```

戻り値

System.Data.Common.DbConnectionStringBuilder の新しいインスタンス。

CreateDataAdapter メソッド

System.Data.Common.DbDataAdapter クラスを実装する Adaptive Server ADO.NET Data Provider クラスの新しいインスタンスを返します。

構文

```
DbDataAdapter CreateDataAdapter()
```

戻り値

System.Data.Common.DbDataAdapter の新しいインスタンス。

CreateDataSourceEnumerator メソッド

System.Data.Common.DbDataSourceEnumerator クラスを実装する Adaptive Server ADO.NET Data Provider クラスの新しいインスタンスを返します。

構文

```
DbDataSourceEnumerator CreateDataSourceEnumerator)
```

戻り値

System.Data.Common.DbDataSourceEnumerator の新しいインスタンス。

CreateParameter メソッド

System.Data.Common.DbParameter クラスを実装する Adaptive Server ADO.NET Data Provider クラスの新しいインスタンスを返します。

構文

```
DbParameter CreateParameter()
```

戻り値

System.Data.Common.DbParameter の新しいインスタンス。

CreatePermission メソッド

System.Data.Common.CreatePermission クラスを実装する Adaptive Server ADO.NET Data Provider クラスの新しいインスタンスを返します。

構文

```
CodeAccessPermission CreatePermission( PermissionState state )
```

パラメータ

state - パーミッションにリソースへのアクセス権を設定するかどうかを指定します。

戻り値

System.Data.Common.CreatePermission の新しいインスタンス。

CanCreateDataSourceEnumerator プロパティ

特定の System.Data.Common.DbProviderFactory が System.Data.Common.DbDataSourceEnumerator クラスをサポートするかどうかを指定します。

構文

```
bool CanCreateDataSourceEnumerator
```

アクセス
読み込み専用

プロパティ値
System.Data.Common.DbProviderFactory のインスタンスが System.Data.Common.DbDataSourceEnumerator クラスをサポートする場合は true、それ以外の場合は false。

AseClientPermission クラス

Adaptive Server ADO.NET Data Provider で、適切なセキュリティレベルのユーザのみに Adaptive Server Enterprise データソースへのアクセスが許可されることを保証できるようにします。

基本クラス

```
DBDataPermission
```

AseClientPermission コンストラクタ

AseClientPermission クラスの新しいインスタンスを初期化します。

構文 1

```
void AseClientPermission( )
```

構文 2

```
void AseClientPermission( PermissionState state )
```

構文 3

```
void AseClientPermission( PermissionState state, bool
allowBlankPassword )
```

パラメータ

- **state –** PermissionState 値のいずれか。
- **allowBlankPassword –** ブランクのパスワードが許可されるかどうかを指定します。

AseClientPermissionAttribute クラス

カスタムセキュリティ属性にセキュリティアクションを関連付けます。

基本クラス

```
DBDataPermissionAttribute
```

AseClientPermissionAttribute コンストラクタ

AseClientPermissionAttribute クラスの新しいインスタンスを初期化します。

構文

```
void AseClientPermissionAttribute( SecurityAction action )
```

パラメータ

action - 宣言セキュリティを使用して実行可能なアクションを表す SecurityAction 値のいずれか。

戻り値

AsePermissionAttribute オブジェクト。

CreatePermission メソッド

属性のプロパティに応じて設定される AsePermission オブジェクトを返します。

構文

```
IPermission CreatePermission( )
```

AseCommand クラス

Adaptive Server に対して実行されるコマンドを表し、動的 SQL 文またはストアドプロシージャをカプセル化します。

次のメソッドで、Adaptive Server データベースに対してコマンドを実行します。

- `ExecuteNonQuery` - resultset を返さない **Execute** コマンド

- **ExecuteScalar** - resultset を返さない **Execute** コマンド
- `ExecuteReader` - 単一値を返す **Execute** コマンド
- `ExecuteXmlReader` - XML を返す **Execute** コマンド

基本クラス

```
Component
```

実装

```
IDbCommand, IDisposable
```

注意： パラメータを取るストアドプロシージャを呼び出す場合は、そのストアドプロシージャのパラメータを指定する必要があります。

参照：

- AseCommand を使用したデータの取得と操作 (46 ページ)
- ストアドプロシージャ (83 ページ)
- AseParameter クラス (203 ページ)

AseCommand コンストラクタ

`AseCommand` オブジェクトを初期化します。

構文 1

```
void AseCommand( )
```

構文 2

```
void AseCommand( string cmdText )
```

構文 3

```
void AseCommand( string cmdText, AseConnection connection )
```

構文 4

```
void AseCommand( string cmdText, AseConnection connection,
AseTransaction transaction )
```

パラメータ

- **cmdText –** SQL 文またはストアドプロシージャのテキスト。
- **connection –** 現在の接続。
- **transaction – AseConnection** を実行する AseTransaction。

Cancel メソッド

AseCommand オブジェクトの実行をキャンセルします。

構文

```
void Cancel( )
```

使用法

キャンセルする対象がない場合は、何も実行されません。 処理中のコマンドがあり、キャンセル操作に失敗した場合、例外は生成されません。

実装

```
IDbCommand.Cancel
```

CommandText プロパティ

SQL 文またはストアドプロシージャのテキストを表します。

構文

```
string CommandText
```

アクセス

読み込み/書き込み

プロパティ値

実行する SQL 文またはストアドプロシージャの名前。 デフォルトは空の文字列です。

実装

```
IDbCommand.CommandText
```

参照：

- AseCommand コンストラクタ (124 ページ)

CommandTimeout プロパティ

コマンド実行の試行を終了してエラーを生成するまでの待機時間を秒単位で表します。

構文

```
int CommandTimeout
```

アクセス

読み込み/書き込み

実装
`IDbCommand.CommandTimeout`

デフォルト
30 秒

使用法
0 を指定すると、待機時間は制限されません。 コマンド実行の試行待機時間が永続化するため、0 は指定しないでください。

CommandType プロパティ

AseCommand によって示されるコマンドの種類を表します。

構文
`CommandType` **CommandType**

アクセス
読み込み/書き込み

実装
`IDbCommand.CommandType`

使用法
サポートされるコマンドの種類は次のとおりです。

- **CommandType.StoredProcedure** - この CommandType を指定する場合、コマンドテキストをストアドプロシージャ名とし、引数を `AseParameter` オブジェクトとして指定する必要があります。
- **CommandType.Text** - デフォルト値。

CommandType プロパティを `StoredProcedure` に設定したときは、**CommandText** プロパティにストアドプロシージャの名前を設定してください。 **Execute** メソッドの 1 つを呼び出すと、このストアドプロシージャが実行されます。

Connection プロパティ

`AseCommand` オブジェクトで使用する接続オブジェクトを表します。

構文
`AseConnection` **Connection**

アクセス
読み込み/書き込み

デフォルト
デフォルト値は null 参照です。 Visual Basic の場合は「Nothing」です。

CreateParameter メソッド

`AseCommand` オブジェクトにパラメータを渡す `AseParameter` オブジェクトを作成します。

構文
```
AseParameter CreateParameter( )
```

戻り値
`AseParameter` オブジェクトとしての新しいパラメータ。

使用法

- ストアドプロシージャや一部の SQL 文では、文のテキストで ***@name*** パラメータで指定したパラメータを取ることができます。
- **CreateParameter** メソッドは `AseParameter` オブジェクトを提供します。AseParameter のプロパティを設定することで、パラメータの値やデータ型などを指定できます。

参照：
- AseParameter クラス (203 ページ)

ExecuteNonQuery メソッド

Insert、**Update**、**Delete**、データ定義文など、結果セットを返さない文を実行します。

構文
```
int ExecuteNonQuery( )
```

戻り値
影響を受けたローの数。

実装
```
IDbCommand.ExecuteNonQuery
```

使用法

- **ExecuteNonQuery** では、DataSet を使わずにデータベース内のデータを変更できます。 データを変更するには、**Update**、**Insert**、または **Delete** 文を実行します。
- **ExecuteNonQuery** はローを返しませんが、出力パラメータ、またはパラメータにマッピングされた戻り値にはデータが格納されます。
- **Update**、**Insert**、**Delete** 文では、戻り値はコマンドによって影響を受けたローの数です。 他の種類の文では、戻り値は -1 になります。

ExecuteReader メソッド

結果セットを返す SQL 文を実行します。

構文 1

```
AseDataReader ExecuteReader( )
```

構文 2

```
AseDataReader ExecuteReader( CommandBehavior behavior )
```

パラメータ

- **動作 – CloseConnection**、**Default**、**KeyInfo**、**SchemaOnly**、**SequentialAccess**、**SingleResult**、または **SingleRow** のいずれか。

  デフォルト値は **CommandBehavior.Default** です。 このパラメータの詳細については、.NET Framework のドキュメントで CommandBehavior Enumeration (CommandBehavior 列挙体) に関する説明を参照してください。

戻り値

`AseDataReader` オブジェクトとしての結果セット。

使用法

文は、必要に応じて CommandText や Parameters を設定した、現在の `AseCommand` オブジェクトです。 `AseDataReader` オブジェクトは、前方向のみの読み込み専用結果セットです。 変更可能な結果セットが必要な場合は、`AseDataAdapter` クラスを使用してください。

参照：

- AseDataAdapter クラス (158 ページ)
- AseDataReader クラス (168 ページ)

ExecuteScalar メソッド

単一の値を返す文を実行します。 複数のローとカラムを返すクエリに対してこのメソッドを呼び出すと、最初のローの最初のカラムだけが返されます。

構文

```
object ExecuteScalar( )
```

戻り値
結果セットの最初のローの最初のカラム。結果セットが空の場合は、null 参照が返されます。

実装

```
IDbCommand.ExecuteScalar
```

ExecuteXmlReader メソッド

有効な **FOR XML** 句を持つ SQL 文を実行し、結果セットを返します。 XML が格納された 1 つのテキストカラムを返す文を使用して、ExecuteXmlReader を呼び出すこともできます。

構文 1

```
XmlReader ExecuteXmlReader( )
```

パラメータ
なし。

戻り値
`XmlReader` オブジェクトとしての結果セット。

使用法
文は、必要に応じて CommandText や Parameters を設定した、現在の `AseCommand` オブジェクトです。

参照
Microsoft .NET ドキュメントの `XmlReader`

NamedParameters

このプロパティは、この接続に関連付けられている `AseCommand` オブジェクトのデフォルトの動作を決定します。

構文

```
bool NamedParameters
```

プロパティ値

デフォルト値は、関連付けられている `AseConnection` オブジェクトで設定されている値と同じです。 このプロパティを「true」 (`AseConnection` のデフォルト) にすると、Provider ではパラメータマーカ (「?」) ではなくパラメータ名が使用されていると判断されます。 次に例を示します。

```
select total_sales from titles where title_id = @title_id
```

パラメータマーカを使用する場合は、このプロパティを false に設定します。 これは、ODBC および JDBC と互換性があります。

次に例を示します。

```
select total_sales from titles where title_id = ?
```

アクセス

読み込み/書き込み

Parameters プロパティ

現在の文のパラメータコレクションを表します。 パラメータを指定するには、CommandText で ***@name*** パラメータまたは疑問符を使用します。

構文

```
AseParameterCollection Parameters
```

アクセス

読み込み専用

プロパティ値

SQL 文またはストアドプロシージャのパラメータ。 デフォルト値は空のコレクションです。

使用法

- CommandType が Text に設定されている場合は、***@name*** パラメータを使用します。 次に例を示します。

```
SELECT * FROM Customers WHERE CustomerID = @name
```

NamedParameters が「false」に設定されている場合は、? パラメータマーカを使用します。 次に例を示します。

```
SELECT * FROM Customers WHERE CustomerID = ?
```

- 疑問符のプレースホルダを使用する場合、`AseParameter` オブジェクトが AseParameterCollection に追加される順序と、コマンドテキスト内のパラメータの疑問符プレースホルダの位置を完全に一致させてください。
- コレクション内のパラメータと実行されるクエリの要件が一致しない場合は、エラーが発生するか、例外が返されます。
- 出力パラメータには、`AseDbType` を指定してください (準備のありなしに関係なく)。

参照：

- AseParameterCollection クラス (210 ページ)

Prepare メソッド

データソースに対する **AseCommand** を準備またはコンパイルします。

構文

```
void Prepare( )
```

実装

```
IDbCommand.Prepare
```

使用法

- **Prepare** を呼び出す前に、準備対象となる文の各パラメータのデータ型を指定します。
- **Prepare** の呼び出し後に **Execute** メソッドを呼び出すと、`Size` プロパティに指定した値よりも大きいパラメータ値は、最初に指定したパラメータのサイズに自動的にトランケートされます。このとき、トランケートエラーは返されません。

Transaction プロパティ

現在のコマンドをトランザクションに関連付けます。

構文

```
AseTransaction Transaction
```

アクセス

読み込み/書き込み

使用法
デフォルト値は null 参照です。 Visual Basic の場合は Nothing です。

Transaction プロパティに特定の値がすでに設定されていて、コマンドが実行中の場合、このプロパティは設定できません。 `AseCommand` オブジェクトと同じ AseConnection に接続されていない `AseTransaction` オブジェクトをトランザクションプロパティに設定すると、次回、文を実行する際に例外が返されます。

UpdatedRowSource プロパティ

AseDataAdapter の **Update** メソッドで使用されるときに DataRow に適用されるコマンドの結果を表します。

構文
```
UpdateRowSource UpdatedRowSource
```

アクセス
読み込み/書き込み

実装
```
IDbCommand.UpdatedRowSource
```

プロパティ値
`UpdatedRowSource` 値の 1 つ。 コマンドが自動的に生成される場合、デフォルトは None です。 コマンドが自動的に生成されない場合、デフォルトは「Both」になります。

AseCommandBuilder クラス

SQL Select 文に基づいて単一のテーブルに対する SQL Insert、Update、Delete 文を自動的に作成します。

基本クラス
```
Component
```

実装
```
IDisposable
```

参照：
- AseDataAdapter クラス (158 ページ)

AseCommandBuilder コンストラクタ

AseCommandBuilder オブジェクトを初期化します。

構文 1

```
void AseCommandBuilder( )
```

構文 2

```
void AseCommandBuilder( AseDataAdapter adapter )
```

パラメータ
adapter - 調整のための文の生成対象となる AseDataAdapter オブジェクトです。

DataAdapter プロパティ

文の生成対象となる **AseDataAdapter** を表します。

構文

```
AseDataAdapter DataAdapter
```

アクセス
読み込み/書き込み

プロパティ値
AseDataAdapter オブジェクトです。

使用法
AseCommandBuilder の新しいインスタンスを作成すると、この **AseDataAdapter** に関連付けられている既存のすべての AseCommandBuilder が解放されます。

DeleteCommand プロパティ

Update() を呼び出したときに、データベースに対して実行される AseCommand オブジェクト。DataSet 内の削除されたローと対応するデータベース内のローを削除します。

構文

```
AseCommand DeleteCommand
```

アクセス
読み込み/書き込み

使用法
既存の AseCommand オブジェクトに DeleteCommand を割り当てた場合、AseCommand オブジェクトのクローンは作成されません。DeleteCommand は、既存の AseCommand に対する参照を維持します。

DeriveParameters メソッド

指定した AseCommand オブジェクトの Parameters コレクションを作成します。 これは、AseCommand で指定したストアドプロシージャで使用されます。

構文
```
void DeriveParameters( AseCommand command )
```

パラメータ
command - パラメータ抽出の対象となる AseCommand オブジェクトです。

使用法

- DeriveParameters は、**AseCommand** のすべての既存パラメータ情報を上書きします。
- DeriveParameters では、データベースサーバへの追加呼び出しが必要になります。 パラメータ情報が事前にわかっている場合は、情報を明示的に設定することによって Parameters コレクションを作成する方が効率的です。

Dispose メソッド

オブジェクトに関連付けられたリソースを解放します。

説明

構文
```
void Dispose( )
```

GetDeleteCommand メソッド

生成された AseCommand オブジェクト、**AseDataAdapter.Update()** が呼び出されたときに、データベースに対して **Delete** 操作を実行します。

構文
```
AseCommand GetDeleteCommand( )
```

戻り値
削除の実行に必要な、自動生成された AseCommand オブジェクト。

使用法

- **GetDeleteCommand** メソッドは、実行対象の `AseCommand` オブジェクトを返します。これは、情報入手またはトラブルシューティングの目的で役立つ場合があります。
- 修正するコマンドのベースとして **GetDeleteCommand** を使用することもできます。 たとえば、**GetDeleteCommand** を呼び出し、CommandTimeout 値を変更した上で、その値を明示的に AseDataAdapter に設定できます。
- アプリケーションが **Update** または **GetDeleteCommand** を呼び出すと、最初に SQL 文が生成されます。 アプリケーションが何らかの方法で SQL 文を変更する場合は、最初の SQL 文が生成された後に **RefreshSchema** を明示的に呼び出す必要があります。 呼び出しを行わない場合、**GetDeleteCommand** は以前の文の情報を引き続き使用します。

GetInsertCommand メソッド

生成された `AseCommand` オブジェクト、AseDataAdapter.Update() が呼び出されたときに、データベースに対して **Insert** 操作を実行します。

構文

```
AseCommand GetInsertCommand( )
```

戻り値

挿入の実行に必要な、自動生成された `AseCommand` オブジェクト。

使用法

- **GetInsertCommand** メソッドは、実行対象の `AseCommand` オブジェクトを返します。これは、情報入手またはトラブルシューティングの目的で役立つ場合があります。
- 修正するコマンドのベースとして **GetInsertCommand** を使用することもできます。 たとえば、**GetInsertCommand** を呼び出し、CommandTimeout 値を変更した上で、その値を明示的に AseDataAdapter に設定できます。
- アプリケーションが **Update** または **GetInsertCommand** のいずれかを呼び出すと、SQL 文が生成されます。 アプリケーションが何らかの方法で SQL 文を変更する場合は、最初の SQL 文が生成された後に **RefreshSchema** を明示的に呼び出す必要があります。 呼び出しを行わない場合、**GetInsertCommand** は引き続き、以前の文の情報 (正しくないことがあります) を使用します。

GetUpdateCommand メソッド

生成された AseCommand オブジェクト、**AseDataAdapter.Update()** が呼び出されたときに、データベースに対して **Update** 操作を実行します。

構文

```
AseCommand GetUpdateCommand( )
```

戻り値

更新の実行に必要な、自動生成された AseCommand オブジェクト。

使用法

- **GetUpdateCommand** メソッドは、実行対象の AseCommand オブジェクトを返します。これは、情報入手またはトラブルシューティングの目的で役立つ場合があります。
- 修正するコマンドのベースとして **GetUpdateCommand** を使用することもできます。 たとえば、**GetUpdateCommand** を呼び出し、CommandTimeout 値を変更した上で、その値を明示的に **AseDataAdapter** に設定できます。
- アプリケーションが Update または **GetUpdateCommand** を呼び出すと、SQL 文が最初に生成されます。 アプリケーションが何らかの方法で SQL 文を変更する場合は、最初の SQL 文が生成された後に **RefreshSchema** を明示的に呼び出す必要があります。 呼び出しを行わない場合、**GetUpdateCommand** は、以前の文の情報 (正しくないことがあります) を引き続き使用します。

InsertCommand プロパティ

Update() が呼び出されたときにデータベースに対して実行されるコマンド。 DataSet に挿入されたローと対応させるためのローをデータベースに追加します。

構文

```
AseCommand InsertCommand
```

アクセス

読み込み/書き込み

使用法

既存の AseCommand オブジェクトに InsertCommand を割り当てた場合、AseCommand のクローンは作成されません。 InsertCommand は、既存の AseCommand に対する参照を維持します。

このコマンドによって複数のローが返された場合、AseCommand オブジェクトの UpdatedRowSource プロパティの設定に応じて、それらのローが DataSet に追加されます。

参照：
- Update メソッド (167 ページ)

PessimisticUpdate プロパティ

ペシミスティック更新またはオプティミスティック更新のどちらを実装するかを示します。

構文
```
public bool PessimisticUpdate
```

アクセス
読み込み/書き込み

プロパティ値
ペシミスティック更新の場合は true。 オプティミスティック更新の場合は false。

使用法
- ペシミスティック更新では、レコードがロックされます。ロックされたレコードは、すべてのユーザが表示できますが、編集できるユーザは 1 人だけに制限されます。
- オプティミスティック更新では、複数のユーザが同じレコードを編集できます。

QuotePrefix プロパティ

スペースなどの文字を含むデータベースオブジェクト名を指定する場合に使用する先頭文字 (複数可) です。

構文
```
string QuotePrefix
```

アクセス
読み込み/書き込み

プロパティ値
使用する先頭文字 (複数可)。 角かっこを指定できます。また、ASE QUOTED_IDENTIFIER オプションが「off」に設定されている場合は、二重引用符も使用できます。 デフォルトは空の文字列です。

使用法

- ASE オブジェクトには、スペース、カンマ、セミコロンなどの文字を含めることができます。 QuotePrefix プロパティと QuoteSuffix プロパティには、オブジェクト名をカプセル化するためのデリミタを指定します。
- **Insert**、**Update**、または **Delete** 操作の後に QuotePrefix プロパティまたは QuoteSuffix プロパティを変更することはできませんが、**DataAdapter** の **Update** メソッドを呼び出した後でその設定を変更できます。

参照

- 識別子の詳細については、『ASE リファレンスマニュアル』を参照してください。
- QUOTED IDENTIFIER オプションの詳細については、『ASE リファレンスマニュアル』を参照してください。

QuoteSuffix プロパティ

スペースなどの文字を含むデータベースオブジェクト名を指定する場合に使用する末尾文字 (複数可) です。

構文

```
string QuoteSuffix
```

アクセス
読み込み/書き込み

プロパティ値
使用する末尾文字 (複数可)。 角かっこを指定できます。また、ASE QUOTED_IDENTIFIER オプションが off に設定されている場合は、二重引用符も使用できます。 デフォルトは空の文字列です。

使用法

- ASE オブジェクトには、スペース、カンマ、セミコロンなどの文字を含めることができます。 `QuotePrefix` プロパティと `QuoteSuffix` プロパティは、オブジェクト名をカプセル化するためのデリミタを指定します。
- **Insert**、**Update**、または **Delete** 操作の後に QuotePrefix プロパティまたは QuoteSuffix プロパティを変更することはできませんが、`DataAdapter` の **Update** メソッドを呼び出した後でその設定を変更できます。

参照

- 識別子の詳細については、『ASE リファレンスマニュアル』を参照してください。
- QUOTED IDENTIFIER オプションの詳細については、『ASE リファレンスマニュアル』を参照してください。

RefreshSchema メソッド

Insert 文、**Update** 文、または **Delete** 文の生成で使用されるデータベーススキーマ情報を更新します。

構文

```
void RefreshSchema( )
```

使用法

関連する **AseDataAdapter** の `SelectCommand` の値が変更された場合は、必ず **RefreshSchema** を呼び出してください。

SelectCommand プロパティ

Fill または **FillSchema** で、`DataSet` にコピーする結果セットをデータベースから取得するために使用する **AseCommand** を表します。

構文

```
AseCommand SelectCommand
```

アクセス

読み込み/書き込み

使用法

- 事前に作成された `AseCommand` に `SelectCommand` を割り当てた場合、`AseCommand` のクローンは作成されません。 `SelectCommand` は、その `AseCommand` オブジェクトに対する参照を維持します。
- `SelectCommand` がローを返さない場合は、DataSet にテーブルが追加されず、例外も発生しません。
- **Select** 文は、AseDataAdapter コンストラクタ内でも指定できます。

UpdateCommand プロパティ

Update() が呼び出されたときにデータベースに対して実行される **AseCommand** コマンド。DataSet 内の更新されたローと対応するデータベースのローを更新します。

構文

```
AseCommand UpdateCommand
```

アクセス
読み込み/書き込み

使用法

- 事前に作成された **AseCommand** に UpdateCommand を割り当てた場合、**AseCommand** のクローンは作成されません。 UpdateCommand は、その AseCommand オブジェクトに対する参照を維持します。
- このコマンドによって複数のローが返された場合、AseCommand オブジェクトの UpdatedRowSource プロパティの設定に応じて、それらのローが DataSet にマージされます。

参照：

- Update メソッド (167 ページ)

AseConnection クラス

Adaptive Server データベースへの接続を表します。

基本クラス

```
Component
```

実装

```
IDbConnection, IDisposable
```

参照：

- Adaptive Server データベースへの接続 (30 ページ)

AseConnection コンストラクタ

`AseConnection` オブジェクトを初期化します。 データベースに対して操作を実行するには、最初に接続をオープンします。

構文 1

```
AseConnection( )
```

構文 2

```
AseConnection( string connectionString )
```

パラメータ

connectionString - Adaptive Server 接続文字列。 接続文字列は、キーワードと値をペアごとにセミコロンで区切ったリストです。

注意： Data Source、DataSource、Secondary Data Source、Secondary DataSource は特殊なキーワードです。 これらは、サーバ名を指定するだけでなく、次の形式でも使用できます。

```
DataSource=servername,port
```

```
DataSource=servername:port
```

たとえば、`DataSource=gamg:4100` では、サーバ名が「gamg」に、ポートが「4100」に設定されます。 この場合、`Port` キーワードは接続文字列で不要になります。

例

次の文は、「HR-001」という名前の Adaptive Server データベースサーバで実行されている「policies」という名前のデータベースへの接続に使用する `AseConnection` オブジェクトを初期化します。 この接続では、ユーザ ID に「admin」、パスワードに「money」を使用します。

```
"Data Source='HR-001';
Port=5000; UID='admin';
PWD='money';
Database='policies';"
```

参照：

- 分散トランザクション (95 ページ)

接続文字列パラメータ

次の表で、接続文字列パラメータを確認してください。

接続プロパティ	説明	必須	デフォルト値
UID、UserID、User ID、User	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	ブランク
PWD、Password	Adaptive Server サーバへの接続に使用するパスワード。大文字と小文字を区別する。	いいえ (ユーザ名にパスワードが不要な場合)	ブランク
Server、Data Source、DataSource、Address、Addr、Network Address、Server Name	Adaptive Server サーバの名前または IP アドレス。	はい	ブランク
Port、Server Port	Adaptive Server サーバのポート番号。	はい (ポート番号が Datasource に指定されていない場合)	ブランク
AnsiNull	ODBC に厳格に準拠し、"= NULL" を使用不可とする。 代わりに "IsNull" を使用する。 デフォルトの動作を変更する場合は 1 に設定する。	いいえ	0
ApplicationName、Application Name	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	ブランク
BufferCacheSize	入力/出力バッファをプール内に保持する。 結果セットが非常に大きい場合、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

接続プロパティ	説明	必須	デフォルト値
CharSet	指定されている文字セット。 Adaptive Server ADO.NET Data Provider はデフォルトで 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。 デフォルトの文字セットは ServerDefault。	いいえ	ServerDefault の値
ClientHostName	サーバへのログインレコードで渡されるクライアントホストの名前。たとえば、 `ClientHost-Name='MANGO'`	いいえ	ブランク
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。たとえば、 `ClientHostProc='MANGO-PROC'`	いいえ	ブランク
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
ConnectionIdleTimeout、Connection IdleTimeout、Connection Idle Timeout	ドライバが接続をクローズする前に、接続が接続プール内でアイドル状態を維持できる時間 (秒数)。 ゼロを指定すると、接続は無期限にアイドル状態を維持できる。	いいえ	0

接続プロパティ	説明	必須	デフォルト値
Connection Lifetime	接続がオープン状態を維持できる時間 (秒数)。 デフォルトの Connection Lifetime に到達 (超過) した接続をクライアントがクローズしたときに、ドライバは接続を接続プールに返すのではなくクローズする。 アイドル状態の接続は、デフォルトの Connection Lifetime に到達した時点でクローズされ接続プールから削除される。 Connection Lifetime のデフォルト値は 0 であり、接続は無期限にオープン状態のままであることを示す。	いいえ	0
CumulativeRecordCount、Cumulative Record Count、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 (ADO.NET のプロバイダを使用する場合) 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が含まれる。 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
Database、Initial Catalog	接続するデータベース。	いいえ	ブランク
DSPassword、Directory Service Password	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	ブランク
DSPrincipal、Directory Service Principal	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	ブランク

接続プロパティ	説明	必須	デフォルト値
DSURL、Directory Service URL	LDAP サーバへの URL。	いいえ	ブランク
DTCProtocol (Windows のみ)	分散トランザクションを使用する場合に、ドライバで XA プロトコルまたは OleNative プロトコルのどちらかを使用することを許可する。「Adaptive Server の高度な機能」の「分散トランザクションの使用」を参照。	いいえ	XA
EnableBulkLoad	**ASEBulkCopy** によるバルクロードインタフェースの起動を許可する。 • 0 - オフモード。デフォルト値。 • 1 - **array insert** を使用したバルクロードが有効。 • 2 - **bulk copy** インタフェースを使用したバルクロードが有効。 • 3 - 高速ログ出力 **bulk copy** インタフェースを使用したバルクロードが有効。	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 サーババージョン 15.0 以降を許可する。	いいえ	1
EncryptPassword、EncryptPwd、Encrypt Password	パスワードの暗号化を有効にするかどうかを指定する。 0 はパスワードの暗号化を無効にし、1 はパスワードの暗号化を有効にする。	いいえ	0
Encryption	指定されている暗号化方式。 指定できる値: **ssl**、**none**。	いいえ	ブランク
FetchArraySize	サーバから結果をフェッチする場合にドライバが取得するロー数を指定する。	いいえ	25

接続プロパティ	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
IgnoreErrorsIfRS Pending	エラーメッセージが表示された場合に、ドライバの処理を続行するかどうかを指定する。 1 に設定すると、ドライバでエラーが無視され、サーバからさらに結果が取得可能な場合は結果の処理が続行される。 0 に設定すると、ドライバではエラーが発生した場合に保留中の結果があっても結果の処理を停止する。	いいえ	0
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、Connect Timeout、Connection Timeout	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
min pool size	Adaptive Server への接続を強制的にクローズし、オープンしている接続総数が最小プールサイズに達しないようにできる。 プール内の接続数が最小プールサイズと等しくなると、AseConnection.Close() で接続がクローズされる。	いいえ	20
max pool size	指定した最大プールサイズよりも接続プールが大きくならないように制限できる。 この制限に達すると、AseConnection.Open() で AseException が返される。	いいえ	100

接続プロパティ	説明	必須	デフォルト値
NamedParameters	名前付きパラメータではなくパラメータマーカを使用する場合は、false に設定する。 名前付きパラメータの場合、SQL は `SELECT * from titles where title_id = @title_id` のようになる。パラメータマーカーを使用する場合は、同じ SQL が `SELECT * from titles where title_id = ?` となる。	いいえ	true
PacketSize、Packet Size	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	512
Ping Server	接続プールにある接続を使用する前に、その接続が有効であることを確認する必要がない場合は、false に設定する。	いいえ	true
pooling	接続プールを無効にする場合は、false に設定する。	いいえ	true
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
RestrictMaximum Packet-Size	EnableServerPacketSize が 1 に設定され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

接続プロパティ	説明	必須	デフォルト値
SecondaryPort、Secondary Port、Secondary Server Port	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 サーバのポート番号。	はい (HASession が 1 に設定され、Secondary DataSource でポート番号が指定されていない場合)	ブランク
SecondaryServer、Secondary Data Source、Secondary DataSource、Secondary Server、Secondary Address、Secondary Addr、Secondary Network Address	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。「Seconcary Data Source」には、SecondaryServer:SecondaryPort または SecondaryServer, SecondaryPort の使用が可能。	はい (HASession が 1 に設定されている場合)	ブランク
SupressRowFormat2	可能な限り TDS_ROWFMT2 バイトシーケンスではなく、TDS_ROWFMT バイトシーケンスでデータを送信するように Adaptive Server に強制する。	いいえ	0
SQL Initialization String、SQLInitializationString、SQL Init String、SQLInitString、Initialization String、InitializationString	データベースサーバに渡されて、接続の直後に実行されるコマンドをスペースで区切って定義する。指定するコマンドは、結果の問い合わせには使用されない SQL コマンドであること。	いいえ	ブランク
TextSize	Adaptive Server との間で送受信されるバイナリまたはテキストデータのバイト単位での最大サイズ。たとえば、`TextSize=64000` では、最大サイズが 64 KB に制限される。	いいえ	ブランク Adaptive Server のデフォルトは 32 KB。

接続プロパティ	説明	必須	デフォルト値
TightlyCoupled Transaction (Windows のみ y)	分散トランザクションを使用するときに、同一の Adaptive Server サーバに接続する 2 つの DSN を使用している場合は、このプロパティを 1 に設定する。「Adaptive Server の高度な機能」の「分散トランザクションの使用」を参照。	いいえ	0
TrustedFile	Encryption を **ssl** に設定した場合は、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	ブランク
UseAseDecimal	numeric および decimal 値の取得での `AseDecimal` 構造体の使用を有効化する。 `AseDecimal` 構造体では 78 桁の精度/位取りをサポートできる。	いいえ	0
UseCursor、Use Cursor	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
WindowsCharsetConvert-er	ユーザに使用する変換ライブラリの選択を許可する。 • 0 - Adaptive Server ADO.NET Data Provider は文字セット変換に Unilib ライブラリを使用する。 • 1 - Adaptive Server ADO.NET Data Provider は Windows オペレーティングシステムの文字セット変換に Microsoft Unicode 変換ルーチンを使用する。	はい (ユーザが Windows オペレーティングシステムで文字セット変換に Microsoft Unicode 変換ルーチンを使用する場合)	0

BeginTransaction メソッド

トランザクションオブジェクトを返します。 1 つのトランザクションオブジェクトに関連付けられているすべてのコマンドは、単一のトランザクションとして実

行されます。 トランザクションは、**Commit()** または **Rollback()** によって終了します。

構文 1

```
AseTransaction BeginTransaction( )
```

構文 2

```
AseTransaction BeginTransaction( IsolationLevel isolationLevel )
```

パラメータ

- **isolationLevel –** IsolationLevel 列挙体のメンバ。 デフォルト値は `ReadCommitted` です。

戻り値
新しいトランザクションを表すオブジェクト。

使用法
トランザクションオブジェクトにコマンドを関連付けるには、**AseCommand.Transaction** プロパティを使用します。

例

```
AseTransaction tx =
conn.BeginTransaction(IsolationLevel.ReadUncommitted );
```

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

参照：
- トランザクション処理 (86 ページ)
- Commit メソッド (222 ページ)
- Rollback メソッド (223 ページ)

ChangeDatabase メソッド

オープンしている **AseConnection** に対する現在のデータベースを変更します。

構文

```
void ChangeDatabase( string database )
```

パラメータ

- **database –** 現在のデータベースの代わりに使用するデータベースの名前。

実装
```
IDbConnection.ChangeDatabase
```

Close メソッド

データベース接続をクローズします。

構文
```
void Close( )
```

実装
```
IDbConnection.Close
```

使用法
Close メソッドは、保留中のすべてのトランザクションをロールバックします。次に、接続を接続プールに解放します。接続プールが無効の場合は、接続をクローズします。 StateChange イベントの処理中に **Close** を呼び出すと、以降の StateChange イベントは起動されません。 1 つのアプリケーションで **Close** を複数回呼び出すことができます。

ConnectionString プロパティ

データベースへの接続文字列を表します。

構文
```
string ConnectionString
```

アクセス
読み込み/書き込み

実装
```
IDbConnection.ConnectionString
```

使用法

- ConnectionString は ODBC 接続文字列のフォーマットができる限り同じになるように設計されています。
- ConnectionString プロパティを設定できるのは、接続がクローズされているときだけです。 接続文字列値の多くには、対応する読み込み専用プロパティがあります。 接続文字列を設定すると、これらのプロパティのすべてが更新されます。 ただし、エラーが検出された場合は、どのプロパティも更新されません。 **AseConnection** プロパティは、ConnectionString に含まれている設定だけを返します。

- クローズされている接続に対して ConnectionString をリセットすると、パスワードを含むすべての接続文字列値と関連するプロパティがリセットされます。
- プロパティを設定すると、接続文字列の予備検証が実行されます。 アプリケーションが **Open** メソッドを呼び出すと、接続文字列は完全に検証されます。 接続文字列に無効またはサポートされないプロパティが検出された場合は、ランタイム例外が生成されます。
- 値は一重引用符または二重引用符で囲みます。 一重引用符または二重引用符を接続文字列の値の一部として使用する場合は、もう一方の引用符で値を囲みます。 たとえば、`name="value's"` または `name= 'value"s'` は有効ですが、`name='value's'` または `name= ""value""` は無効です。
  ブランク文字は、値または引用符の中で使用されていない限り無視されます。
  キーワードと値の各ペアはセミコロンで区切ります。 セミコロンが値の一部の場合は、引用符も使用して区切ってください。
  エスケープシーケンスはサポートされず、値の型は無関係です。
  名前では、大文字と小文字を区別しません。 接続文字列内に同じプロパティ名が複数回、記述されている場合は、最後の記述に指定されている値が使用されます。
- ユーザ ID やパスワードをダイアログボックスから取得して接続文字列に挿入する場合など、ユーザ入力に基づいて接続文字列を構築するときは注意が必要です。 ユーザが規定外の接続文字列をこれらの値に組み込むことを、アプリケーションは禁止する必要があります。

例
次の文は、サーバ `mango` で実行されている `pubs2` という名前の Adaptive Server データベースに接続する接続文字列を設定し、その接続をオープンします。

```
AseConnection conn = new AseConnection( “Data Source=mango;
Port=5000; UID=sa; PWD=''; Database='pubs2'; " ):
conn.Open();
```

ConnectionTimeout プロパティ

接続要求がタイムアウトし、エラーが発生するまでの秒数を表します。

構文
```
int ConnectionTimeout
```

アクセス
読み込み専用

デフォルト
15 秒

実装
```
IDbConnection.ConnectionTimeout
```

例
次の文は、ConnectionTimeout を 30 秒に変更します。
```
conn.ConnectionTimeout = 30;
```

CreateCommand メソッド

AseCommand オブジェクトを初期化します。 **AseCommand** のプロパティを使用して、その動作を制御できます。

構文
```
AseCommand CreateCommand( )
```

戻り値
AseCommand オブジェクト。

使用法
Command オブジェクトは **AseConnection** に関連付けられます。

Database プロパティ

接続がオープンされた後に使用される現在のデータベースの名前を表します。

構文
```
string Database
```

アクセス
読み込み専用

実装
```
IDbConnection.Database
```

使用法
```
if (conn.Database != "pubs2") conn.ChangeDatabase("pubs2");
```

InfoMessage イベント

Adaptive Server ADO.NET Data Provider が警告メッセージまたは情報メッセージを送信したときに発生します。

構文

```
event AseInfoMessageEventHandler InfoMessage
```

使用法

イベントハンドラが、このイベントに関連するデータを含む AseInfoMessageEventArgs 型の引数を受け取ります。 **Errors** および **Message** プロパティに、このイベントに固有の情報が記載されます。

NamedParameters

この接続に関連付けられている `AseCommand` オブジェクトのデフォルトの動作を決定します。

構文

```
bool NamedParameters
```

プロパティ値

プロパティ値は `ConnectString` (**NamedParameters='true'/'false'**) によって設定されます。または、ユーザが `AseConnection` のインスタンスを介して直接設定できます。

アクセス

読み込み/書き込み

参照：

- NamedParameters (130 ページ)

Open メソッド

事前に指定された接続文字列を使用して、データベースへの接続をオープンします。

構文

```
void Open( )
```

実装

```
IDbConnection.Open
```

使用法

- 接続プール内にあるオープンされている接続を使用できる場合、**AseConnection** はその接続を使用します。 使用できない場合は、データソースへの新しい接続を確立します。
- スコープ外にある **AseConnection** はクローズされません。 そのため、**Close** または **Dispose** を呼び出して、明示的に接続をクローズする必要があります。

State プロパティ

接続の現在のステータスを表します。

構文

```
ConnectionState State
```

アクセス
読み込み専用

デフォルト
Closed

実装

```
IDbConnection.State
```

参照：

- 接続ステータスの確認 (33 ページ)

StateChange イベント

接続のステータスが変わったときに発生します。

構文

```
event StateChangeEventHandler StateChange
```

使用法
イベントハンドラが、このイベントに関連するデータを含む **StateChangeEventArgs** 型の引数を受け取ります。 **StateChangeEventArgs** プロパティには、このイベントに固有の情報が記載されます (**CurrentState** および **OriginalState**)。

TraceEnter、TraceExit イベント

デバッグ目的で、アプリケーション内でのデータベースアクティビティをトレースします。

構文

```
public delegate void TraceEnterEventHandler(AseConnection
   connection, object source, string method, object[] parameters);
```

```
public delegate void TraceExitEventHandler(AseConnection
   connection, object source, string method, object returnValue);
```

使用法

TraceEnter と **TraceExit** イベントは、独自のトレース方法を組み込むときに使用します。 このイベントは、個々の接続インスタンスに固有です。 これによって、接続ごとのログをそれぞれ異なるファイルに記録できます。 このイベントは無視することも、他のトレース用にプログラムすることもできます。 また .NET イベントを使用することで、1 つの接続オブジェクトに対して複数のイベントハンドラを設定できます。 これによって、ウィンドウとファイルの両方に、同時にイベントのログを出力できます。

Adaptive Server ADO.NET Data Provider アクティビティをトレースするには、ENABLETRACING 接続プロパティを有効にします。 トレースが不要な通常の実行時のパフォーマンスを高めるために、このプロパティはデフォルトでは無効に設定されます。 このプロパティが無効の場合、**TraceEnter** イベントと **TraceExit** イベントはトリガされず、トレースイベントは実行されません。 ENABLETRACING は、以下の値を使用して接続文字列に設定することができます。

- true - **TraceEnter** イベントおよび **TraceExit** イベントをトリガします。
- false - デフォルト値。Adaptive Server ADO.NET Data Provider は **TraceEnter** および **TraceExit** イベントを無視します。

AseConnectionPool クラス

同じ接続のプールを管理します。

Available プロパティ

プール内で使用可能な接続数を表します。

構文

```
int Available
```

アクセス
読み込み専用

Size プロパティ

プール内の接続数を表します。

構文

```
int Size
```

アクセス
読み込み専用

AseConnectionPoolManager クラス

すべての接続プールを管理します。

AseConnectionPoolManager コンストラクタ

AseConnectionPoolManager クラスの新しいインスタンスを初期化します。

説明

構文

```
void AseConnectionPoolManager()
```

GetConnectionPool メソッド

指定された接続文字列の接続を管理する接続プールを取得します。

構文

```
AseConnectionPool GetConnectionPool( string connectionString )
```

パラメータ
connectionString - 取得対象のプールを特定する接続文字列。

戻り値
connectionString を管理する接続プール。

NumberOfOpenConnections プロパティ

すべての接続プール内でオープンしている接続の数を表します。

構文

```
int NumberOfOpenConnections
```

アクセス
読み込み専用

AseDataAdapter クラス

DataSet クラスと Adaptive Server の間でリンクの機能を果たします。

説明
このクラスでは、次の 2 つの重要メソッドが使用されます。

- **Fill()** - サーバからのデータを DataSet に書き込む
- **Update()** - DataSet での変更をサーバに書き戻して適用する

AseDataAdapter Fill または **Update** メソッドを使用するときに接続がオープンされていない場合は、このメソッドで接続をオープンできます。

基本クラス

```
Component
```

実装

```
IDbDataAdapter, IDisposable
```

使用法
DataSet を使用すると、データをオフラインで操作できます。 **AseDataAdapter** は、DataSet に一連の SQL 文を関連付けるメソッドを提供します。

参照：

- AseDataAdapter を使用したデータへのアクセスと操作 (58 ページ)
- データへのアクセスおよび操作方法 (45 ページ)

AseDataAdapter コンストラクタ

AseDataAdapter オブジェクトを初期化します。

構文 1

```
AseDataAdapter( )
```

構文 2

```
AseDataAdapter( AseCommand selectCommand )
```

構文 3

```
AseDataAdapter( string selectCommandText, AseConnection
selectConnection )
```

構文 4

```
AseDataAdapter( string selectCommandText, string
selectConnectionString )
```

パラメータ

- **selectCommand –** Fill 時に、DataSet に格納するレコードのデータソースからの選択に使用される AseCommand オブジェクト。
- **selectCommandText –** AseDataAdapter の SelectCommand プロパティで使用する **Select** 文またはストアドプロシージャ。
- **selectConnection –** データベースへの接続を定義する AseConnection オブジェクト。
- **selectConnectionString –** Adaptive Server データベースへの接続文字列。

例
次のコードは AseDataAdapter オブジェクトを初期化します。

```
AseDataAdapter da = new AseDataAdapter( "SELECT emp_id, emp_lname
FROM employee," conn );
```

AcceptChangesDuringFill プロパティ

DataTable に DataRow が追加された後で、そのローに対して AcceptChanges を呼び出すかどうかを指定する値を表します。

構文

```
bool AcceptChangesDuringFill
```

アクセス
読み込み/書き込み

使用法
このプロパティを「true」に設定すると、DataAdapter は DataRow に対して **AcceptChanges** 関数を呼び出します。 「false」に設定すると、AcceptChanges は呼び出されず、新しく追加されたローは挿入対象のローとして扱われます。 デフォルトは「true」です。

ContinueUpdateOnError プロパティ

ローの更新中にエラーが発生したときに、例外を生成するかどうかを指定する値を表します。

構文
```
bool ContinueUpdateOnError
```

アクセス
読み込み/書き込み

使用法

- デフォルトは「false」です。 このプロパティを「true」に設定すると、例外を生成せずに更新が続行されます。
- `ContinueUpdateOnError` が「true」の場合は、ローの更新中にエラーが発生しても例外は返されません。 ローの更新がスキップされ、そのローの RowError プロパティにエラー情報が記録されます。 DataAdapter は後続のローの更新を継続します。
- `ContinueUpdateOnError` が「false」の場合は、エラーの発生時に例外が返されます。

DeleteCommand プロパティ

Update() を呼び出したときに、データベースに対して実行される `AseCommand` オブジェクト。DataSet 内の削除されたローと対応するデータベース内のローを削除します。

構文
```
AseCommand DeleteCommand
```

アクセス
読み込み/書き込み

使用法
既存の AseCommand オブジェクトに DeleteCommand を割り当てた場合、AseCommand オブジェクトのクローンは作成されません。DeleteCommand は、既存の AseCommand に対する参照を維持します。

Fill メソッド

データベースから取得したデータで、DataSet または DataTable オブジェクトのローを追加またはリフレッシュします。

構文 1
```
int Fill( DataSet dataSet )
```

構文 2
```
int Fill( DataSet dataSet, string srcTable )
```

構文 3
```
int Fill( DataSet dataSet, int startRecord, int maxRecords, string
srcTable )
```

構文 4
```
int Fill( DataTable dataTable )
```

パラメータ

- **dataSet –** レコードとスキーマ (オプション) が書き込まれる DataSet。
- **srcTable –** テーブルマッピングに使用するソーステーブルの名前。
- **startRecord –** 処理を開始するレコード番号 (0 から始まる)。
- **maxRecords –** DataSet に読み込まれるレコードの最大数。
- **dataTable –** レコードとスキーマ (オプション) が書き込まれる DataTable。

戻り値
DataSet での追加またはリフレッシュが成功したローの数。

使用法

- **StartRecord** 引数を使用して、DataSet にコピーされるレコード数を制限している場合でも、AseDataAdapter クエリでは、すべての該当レコードがデータベースからクライアントにフェッチされます。 結果セットが大きい場合は、パフォーマンスに重大な影響を与える可能性があります。
- 結果セットの前方向への読み込み専用で間に合う場合は、代替方法として、SQL 文 (**ExecuteNonQuery**) などとともに AseDataReader を使用して、変更を

実行します。 別の代替方法として、必要な結果セットだけを返すストアドプロシージャを記述することもできます。
- `SelectCommand` がローを返さない場合は、`DataSet` にテーブルは追加されず、例外も発生しません。

参照
サポートされている fill メソッドの一覧については、.NET Framework のドキュメントで **DdDataAdapter.fill()** の説明を参照してください。

FillError イベント

fill 操作中にエラーが発生したときに返されます。

構文
```
event FillErrorEventHandler FillError
```

使用法
`FillError` イベントを使用すると、エラーの発生後に **Fill** 操作を続行するかどうかをユーザが指定できます。 `FillError` イベントは、次のような場合に発生する可能性があります。

- `DataSet` に追加するデータを共通言語ランタイム型に変換することで、データの精度が失われる場合。
- 追加するローに含まれるデータが、`DataSet` 内の `DataColumn` に強制される制約に違反している場合。

FillSchema メソッド

1 つ以上の DataTable を DataSet に追加して、データソース内のスキーマと一致するようにスキーマを設定します。

構文 1
```
DataTable[ ] FillSchema( DataSet dataSet, SchemaType schemaType )
```

構文 2
```
DataTable[ ] FillSchema( DataSet dataSet, SchemaType schemaType,
string srcTable )
```

構文 3
```
DataTable FillSchema( DataTable dataTable, SchemaType schemaType )
```

パラメータ

- **dataSet –** レコードとスキーマ (オプション) が書き込まれる `DataSet`。
- **schemaType –** スキーマの挿入方法を指定する `SchemaType` 値のいずれか。
- **srcTable –** テーブルマッピングに使用するソーステーブルの名前。
- **dataTable –** `DataTable`。

戻り値
構文 1 と 2 の場合、戻り値は `DataSet` に追加された `DataTable` オブジェクトのコレクションに対する参照です。 構文 3 の場合、戻り値は特定の `DataTable` に対する参照です。

詳細は、.NET Framework のヘルプ (http://msdn.microsoft.com/) を参照してください。

参照：
- AseDataAdapter スキーマ情報の取得 (69 ページ)

GetFillParameters

Select 文の実行時にユーザによって設定されるパラメータを取得します。

構文
```
AseParameter[ ] GetFillParameters( )
```

戻り値
ユーザが設定したパラメータを含む `IDataParameter` オブジェクトの配列。

実装
```
IDataAdapter.GetFillParameters
```

InsertCommand プロパティ

Update() が呼び出されたときにデータベースに対して実行されるコマンド。
DataSet に挿入されたローと対応させるためのローをデータベースに追加します。

構文
```
AseCommand InsertCommand
```

アクセス
読み込み/書き込み

使用法

- 既存の `AseCommand` オブジェクトに `InsertCommand` を割り当てた場合、`AseCommand` のクローンは作成されません。 `InsertCommand` は、既存の `AseCommand` に対する参照を維持します。
- このコマンドによって複数のローが返された場合、`AseCommand` オブジェクトの `UpdatedRowSource` プロパティの設定に応じて、それらのローが `DataSet` に追加されます。

参照：

- AseDataAdapter オブジェクトを使用したローの挿入、更新、削除 (60 ページ)
- Update メソッド (167 ページ)
- AseCommand オブジェクトを使用したローの挿入、更新、削除 (51 ページ)

MissingMappingAction プロパティ

一致するテーブルまたはカラムが受信データにない場合に実行するアクションを決定します。

構文

```
MissingMappingAction MissingMappingAction
```

アクセス
読み込み/書き込み

プロパティ値
`MissingMappingAction` 値の 1 つ。 デフォルトは **Passthrough** です。

実装

```
IDataAdapter.MissingMappingAction
```

MissingSchemaAction プロパティ

既存の `DataSet` スキーマが受信データと一致しないときに実行するアクションを決定します。

構文

```
MissingSchemaAction MissingSchemaAction
```

アクセス
読み込み/書き込み

プロパティ値
`MissingSchemaAction` 値の 1 つ。 デフォルトは `Add` です。

実装
```
IDataAdapter.MissingSchemaAction
```

RowUpdated イベント

更新時、データソースに対してコマンドが実行された後に発生します。 更新が試行され、イベントが初期化されます。

構文
```
event AseRowUpdatedEventHandler RowUpdated
```

使用法
イベントハンドラが、このイベントに関連するデータを含む `AseRowUpdatedEventArgs` 型の引数を受け取ります。 次の `AseRowUpdatedEventArgs` プロパティには、このイベントに固有の情報が記載されます。

- `Command`
- `Errors`
- `RecordsAffected`
- `Row`
- `StatementType`
- `Status`
- `TableMapping`

詳細については、.NET Framework のドキュメントで `OleDbDataAdapter.RowUpdated Event` の説明を参照してください。

RowUpdated イベント

更新時、データソースに対してコマンドが実行される前に発生します。 更新が試行され、イベントが初期化されます。

構文
```
event AseRowUpdatingEventHandler RowUpdating
```

使用法
イベントハンドラが、このイベントに関連するデータを含む `AseRowUpdatingEventArgs` 型の引数を受け取ります。 次の

AseRowUpdatingEventArgs プロパティには、このイベントに固有の情報が記載されます。

- Command
- Errors
- Row
- StatementType
- Status
- TableMapping

詳細については、.NET Framework のドキュメントで OleDbDataAdapter.RowUpdating Event の説明を参照してください。

SelectCommand プロパティ

Fill または **FillSchema** で、DataSet にコピーする結果セットをデータベースから取得するために使用する **AseCommand** を表します。

構文

```
AseCommand SelectCommand
```

アクセス
読み込み/書き込み

使用法

- 事前に作成された AseCommand に SelectCommand を割り当てた場合、AseCommand のクローンは作成されません。 SelectCommand は、その AseCommand オブジェクトに対する参照を維持します。
- SelectCommand がローを返さない場合は、DataSet にテーブルが追加されず、例外も発生しません。
- **Select** 文は、AseDataAdapter コンストラクタ内でも指定できます。

TableMappings プロパティ

ソーステーブルと DataTable とのマスタマッピングを提供するコレクションを表します。

構文

```
DataTableMappingCollection TableMappings
```

アクセス
読み込み専用

使用法

- デフォルト値は空のコレクションです。
- 変更を調整するときに、AseDataAdapter は DataTableMappingCollection コレクションを使用して、データソースで使われるカラム名と DataSet で使われるカラム名を関連付けます。

Update メソッド

データベース内のテーブルを、DataSet に対して行われた変更で更新します。

構文 1

```
int Update( DataSet dataSet )
```

構文 2

```
int Update( DataSet dataSet, string srcTable )
```

構文 3

```
int Update( DataTable dataTable )
```

構文 4

```
int Update( DataRow[ ] dataRows )
```

パラメータ

- **dataSet –** 更新するレコードとスキーマ (オプション) を持つ DataSet。
- **srcTable –** テーブルマッピングに使用するソーステーブルの名前。
- **dataTable –** レコードとスキーマ (オプション) で更新する DataTable。
- **dataRows –** データソースの更新に使用する DataRow オブジェクトの配列。

戻り値

DataSet からの更新に成功したローの数。

使用法

Update は、データセット内にある挿入、更新、削除された各ローに対して、それぞれ InsertCommand、UpdateCommand、DeleteCommand プロパティを使用して実行されます。

.NET Framework のドキュメントを参照してください。

参照：

- AseDataAdapter オブジェクトを使用したローの挿入、更新、削除 (60 ページ)

- DeleteCommand プロパティ (160 ページ)
- InsertCommand プロパティ (163 ページ)
- UpdateCommand プロパティ (168 ページ)

UpdateCommand プロパティ

Update() が呼び出されたときにデータベースに対して実行される **AseCommand**。 DataSet 内の更新されたローと対応するデータベースのローを更新します。

構文

```
AseCommand UpdateCommand
```

アクセス
読み込み/書き込み

使用法

- 事前に作成された **AseCommand** に UpdateCommand を割り当てた場合、**AseCommand** のクローンは作成されません。 UpdateCommand は、その AseCommand オブジェクトに対する参照を維持します。
- このコマンドの実行によって複数のローが返された場合、AseCommand オブジェクトの UpdatedRowSource プロパティの設定に応じて、それらのローが DataSet にマージされます。

参照：

- Update メソッド (167 ページ)

AseDataReader クラス

クエリまたはストアドプロシージャで、前方向への読み込み専用の結果セットが取得されます。

基本クラス

```
MarshalByRefObject
```

実装

```
IDataReader, IDisposable, IDataRecord, IEnumerable, IListSource
```

使用法

- AseDataReader にコンストラクタはありません。 AseDataReader オブジェクトを取得するには、次のように **AseCommand** を実行します。

```
AseCommand cmd = new AseCommand("Select emp_id from employee",
conn );
AseDataReader reader = cmd.ExecuteReader();
```

- `AseDataReader` では、前方向にのみ移動できます。より柔軟な方法で結果を操作するオブジェクトが必要な場合は、`AseDataAdapter` を使用します。
- カーソルの使用時、**AseDataReader** は必要なだけのローを取得します。 詳細については、「AseConnection Constructors」の「ConnectionString プロパティ」で、「`UseCursor`」パラメータを参照してください。

参照：

- データへのアクセスおよび操作方法 (45 ページ)
- ExecuteReader メソッド (128 ページ)

Close メソッド

AseDataReader クラスをクローズします。

構文

```
void Close( )
```

実装

```
IDataReader.Close
```

使用法

AseDataReader を使用した後は、**Close** メソッドを明示的に呼び出してください。

Depth プロパティ

現在のローのネストの深さを示す値を表します。 一番外側のテーブルの深さはゼロです。

構文

```
int Depth
```

アクセス

読み込み専用

プロパティ値

現在のローのネストの深さ。

実装

```
IDataReader.Depth
```

Dispose メソッド

オブジェクトに関連付けられたリソースを解放します。

構文

```
void Dispose( )
```

FieldCount プロパティ

結果セット内のカラム数を表します。

構文

```
int FieldCount
```

アクセス
読み込み専用

プロパティ値
有効なレコードセットに位置していない場合は 0、それ以外の場合は現在のレコードのカラム数です。 デフォルトは -1 です。

実装

```
IDataRecord.FieldCount
```

使用法
有効なレコードセットに位置していない場合、このプロパティの値は 0 です。それ以外の場合は、現在のレコードのカラム数になります。 デフォルトは -1 です。

GetBoolean メソッド

指定したカラムの値を Boolean として取得します。

構文

```
bool GetBoolean( int ordinal )
```

パラメータ
ordinal - 値を取得するカラムを表す序数。 番号付けは 0 から始まります。

戻り値
カラムの値。

実装

```
IDataRecord.GetBoolean
```

使用法
変換は実行されません。 このメソッドは、bit 型のカラムからのデータの取得に使用します。

GetByte メソッド

指定したカラムの値を Byte として取得します。

構文
```
byte GetByte( int ordinal )
```

パラメータ
ordinal - 値を取得するカラムを表す序数。 番号付けは 0 から始まります。

戻り値
カラムの値。

実装
```
IDataRecord.GetByte
```

使用法
変換は実行されません。 このメソッドは、tinyint 型のカラムからのデータの取得に使用します。

GetBytes メソッド

指定したカラムオフセットからのバイトストリームを、配列としてバッファに読み込みます。読み込みは、指定したバッファオフセットから開始されます。

構文
```
long GetBytes( int ordinal, long dataIndex, byte[ ] buffer, int
bufferIndex, int length )
```

パラメータ

- **ordinal –** 値を取得するカラムを表す序数。 番号付けは 0 から始まります。
- **dataIndex –** バイトの読み込みを開始するカラム値内のインデックス。
- **buffer –** データの格納先となる配列。
- **bufferIndex –** データのコピーを開始する配列内のインデックス。
- **length –** 指定したバッファにコピーする最大長。

戻り値
読み込まれたバイト数。

実装
```
IDataRecord.GetBytes
```

使用法

- **GetBytes** は、フィールド内の使用可能なバイト数を返します。 ほとんどの場合、これは正確なフィールド長になります。 ただし、**GetBytes** を使用してフィールドからバイトをすでに取得している場合は、返されるバイト数がフィールドの実際の長さよりも小さくなることがあります。 この現象は、**AseDataReader** クラスが大規模なデータ構造体をバッファに読み込んでいるときなどに起こります。
- null 参照 (Visual Basic の場合は「Nothing」) のバッファを渡すと、**GetBytes** はフィールド長をバイト単位で返します。
- 変換は実行されません。 このメソッドは、`image`、`binary`、`timestamp`、`varbinary` 型のカラムからのデータの取得に使用します。

GetChar メソッド

指定したカラムの値を文字として取得します。

構文
```
char GetChar( int ordinal )
```

パラメータ
ordinal - 値を取得するカラムを表す序数。 番号付けは 0 から始まります。

戻り値
カラムの値。

実装
```
IDataRecord.GetChar
```

使用法

- 変換は実行されません。 このメソッドは、`tinyint`、`char(1)`、`varchar(1)` 型のカラムからのデータの取得に使用します。
- このメソッドを呼び出す前に、**AseDataReader.IsDBNull** を呼び出して null 値をチェックしてください。

参照：

- IsDBNull メソッド (184 ページ)

GetChar メソッド

指定したカラムオフセットからの文字ストリームを、配列としてバッファに読み込みます。読み込みは、指定したバッファオフセットから開始されます。

構文

```
long GetChars( int ordinal, long dataIndex, char[ ] buffer, int
bufferIndex, int length )
```

パラメータ

- **ordinal –** カラムを表す、0 から始まる序数。
- **dataIndex – read** 操作を開始するロー内のインデックス。
- **buffer –** データのコピー先のバッファ。
- **bufferIndex – read** 操作を開始するバッファのインデックス。
- **length –** 読み込む文字数。

戻り値
実際に読み込まれた文字数。

実装

```
IDataRecord.GetChars
```

使用法
GetChars は、フィールド内の使用可能な文字数を返します。 ほとんどの場合、これは正確なフィールド長になります。 ただし、**GetChars** を使用してフィールドから文字をすでに取得している場合は、返される文字数がフィールドの実際の長さよりも小さくなることがあります。 この現象は、`AseDataReader` が大規模なデータ構造体をバッファに読み込んでいるときなどに起こります。

null 参照 (Visual Basic の場合は Nothing) のバッファを渡すと、**GetChars** はフィールド長を文字単位で返します。

変換は実行されません。 変換は実行されません。 このメソッドは、`text`、`char`、`varchar` 型のカラムからのデータの取得に使用します。

参照：

- BLOB の処理 (79 ページ)

GetDataTypeName メソッド

ソースのデータ型の名前を取得します。

構文

```
string GetDataTypeName( int index )
```

パラメータ

index - カラムを表す、0 から始まる序数。

戻り値

バックエンドのデータ型の名前。

実装

```
IDataRecord.GetDataTypeName
```

GetDateTime メソッド

指定したカラムの値を `DateTime` オブジェクトとして取得します。

構文

```
DateTime GetDateTime( int ordinal )
```

パラメータ

ordinal - カラムを表す、0 から始まる序数。

戻り値

指定されたカラムの値。

実装

```
IDataRecord.GetDateTime
```

使用法

- 変換は実行されません。そのため、取得するデータは `DateTime` オブジェクトである必要があります。
- このメソッドを呼び出す前に、**AseDataReader.IsDBNull** を呼び出して null 値をチェックしてください。
- このメソッドは、データベース内の対応する Adaptive Server 型が、`datetime`、`smalldatetime`、`date`、および `time` である場合に使用します。

参照：

- IsDBNull メソッド (184 ページ)

GetDecimal メソッド

指定したカラムの値を Decimal オブジェクトとして取得します。

構文

```
decimal GetDecimal( int ordinal )
```

パラメータ

ordinal - 値を取得するカラムを表す序数。 番号付けは 0 から始まります。

戻り値

指定されたカラムの値。

実装

```
IDataRecord.GetDecimal
```

使用法

- 変換は実行されません。 このメソッドは、decimal、numeric、smallmoney、money 型のカラムからのデータの取得に使用します。
- このメソッドを呼び出す前に、**AseDataReader.IsDBNull** を呼び出して null 値をチェックしてください。

参照：

- IsDBNull メソッド (184 ページ)

GetDouble メソッド

指定したカラムの値を倍精度浮動小数点数として取得します。

構文

```
double GetDouble( int ordinal )
```

パラメータ

ordinal - 値を取得するカラムを表す序数。 番号付けは 0 から始まります。

戻り値

指定されたカラムの値。

実装

```
IDataRecord.GetDouble
```

使用法

- このメソッドを呼び出す前に、**AseDataReader.IsDBNull** を呼び出して null 値をチェックしてください。
- 変換は実行されません。そのため、取得するデータは倍精度浮動小数点数である必要があります。
- 精度が 16 以上の Adaptive Server データ型 `float` には `GetDouble` を使用してください。精度が 16 未満の Adaptive Server データ型 `real` と `float` には `GetFloat` を使用してください。

参照：

- IsDBNull メソッド (184 ページ)

GetFieldType メソッド

オブジェクトのデータ型を取得します。

構文

```
Type GetFieldType( int index )
```

パラメータ
index - カラムを表す、0 から始まる序数。

戻り値
オブジェクトのデータ型を取得します。

実装

```
IDataRecord.GetFieldType
```

GetFloat メソッド

指定したカラムの値を単精度浮動小数点数として取得します。

構文

```
float GetFloat( int ordinal )
```

パラメータ
ordinal - 値を取得するカラムを表す序数。 番号付けは 0 から始まります。

戻り値
指定されたカラムの値。

実装

```
IDataRecord.GetFloat
```

使用法

- 変換は実行されません。そのため、取得するデータは単精度浮動小数点数である必要があります。
- このメソッドを呼び出す前に、**AseDataReader.IsDBNull** を呼び出して null 値をチェックしてください。
- 精度が 16 未満の Adaptive Server データ型 **real** と **float** には **GetFloat** を使用してください。精度が 16 以上の Adaptive Server データ型 **float** には **GetDouble** を使用してください。

参照：

- IsDBNull メソッド (184 ページ)

GetInt16 メソッド

指定したカラムの値を、16 ビットの符号付き整数として取得します。

構文

```
short GetInt16( int ordinal )
```

パラメータ

ordinal - 値を取得するカラムを表す序数。 番号付けは 0 から始まります。

戻り値

指定されたカラムの値。

実装

```
IDataRecord.GetInt16
```

使用法

変換は実行されません。 このメソッドは、`smallint` 型のカラムからデータの取得に使用します。

GetInt32 メソッド

指定したカラムの値を、32 ビットの符号付き整数として取得します。

構文

```
int GetInt32( int ordinal )
```

パラメータ
ordinal - 値を取得するカラムを表す序数。 番号付けは 0 から始まります。

戻り値
指定されたカラムの値。

実装
```
IDataRecord.GetInt32
```

使用法
変換は実行されません。 このメソッドは、`int[eger]` 型のカラムからのデータの取得に使用します。

GetList メソッド

IListSource を実装します。

構文
```
IList GetList();
```

実装
```
IListSource
```

使用法
.NET DataGrid オブジェクトの `DataSource` プロパティを `AseDataReader` に設定できます。 設定されたグリッドは、このメソッドを使用して、`AseDataReader` からの結果をそのセルに直接バインドします。通常、このメソッドを直接使用することはありません。 `AseDataReader` はこのメソッドを実装しているので、次のように実行できます。

```
using(AseCommand cmd = new AseCommand(select total_sales from titles
   where title_id = 'BU1032', conn)
{
   using(AseDataReader rdr = cmd.ExecuteReader())
   {
      MyGrid.DataSource = rdr;
   }
}
```

GetName メソッド

指定したカラムの名前を取得します。

構文
```
string GetName( int index )
```

パラメータ
index - 0 から始まるカラムインデックス。

戻り値
指定されたカラムの名前。

実装
```
IDataRecord.GetName
```

GetOrdinal メソッド

名前を指定したカラムの序数を取得します。

構文
```
int GetOrdinal( string name )
```

パラメータ
name - カラム名。

戻り値
カラムを表す、0 から始まる序数。

実装
```
IDataRecord.GetOrdinal
```

使用法
GetOrdinal は、最初に大文字と小文字を区別して検索を実行します。 それに失敗した場合は、大文字と小文字を区別せずに 2 回目の検索を実行します。

GetOrdinal は、日本語のかなの全角と半角を区別しません。

名前ベースの検索よりも序数ベースの検索の方が効率的なため、ループ内での **GetOrdinal** の呼び出しは非効率的です。 **GetOrdinal** を 1 回呼び出して、その結果を整数の変数に割り当ててループ内で使用すると、処理時間を短縮できます。

GetSchemaTable メソッド

`AseDataReader` のカラムメタデータを説明する DataTable を返します。

構文
```
DataTable GetSchemaTable( )
```

戻り値
カラムメタデータを記述する `DataTable`。

実装
```
IDataReader.GetSchemaTable
```

使用法
このメソッドは、各カラムのメタデータを次の順序で返します。

- `ColumnName`
- `ColumnOrdinal`
- `ColumnSize`
- `DataType`
- `ProviderType`
- `IsLong`
- `AllowDBNull`
- `IsReadOnly`
- `IsRowVersion`
- `IsUnique`
- `IsKeyColumn`
- `IsAutoIncrement`
- `BaseSchemaName`
- `BaseCatalogName`
- `BaseTableName`
- `BaseColumnName`

これらのカラムの詳細については、.NET Framework のドキュメントで `SqlDataReader.GetSchemaTable` の説明を参照してください。

参照：
- DataReader スキーマ情報の取得 (56 ページ)

GetString メソッド

指定したカラムの値を文字列として取得します。

構文
```
string GetString( int ordinal )
```

パラメータ
ordinal - 値を取得するカラムを表す序数。 番号付けは 0 から始まります。

戻り値
指定されたカラムの値。

実装
```
IDataRecord.GetString
```

使用法
変換は実行されません。そのため、取得するデータは文字列である必要があります。

このメソッドを呼び出す前に、**AseDataReader.IsDBNull** を呼び出して null 値をチェックしてください。

参照：
- IsDBNull メソッド (184 ページ)

GetUInt16 メソッド

指定したカラムの値を、16 ビットの符号なし整数として取得します。

構文
```
UInt16 GetUInt16( int ordinal )
```

パラメータ
ordinal - 値を取得するカラムを表す序数。 番号付けは 0 から始まります。

戻り値
指定されたカラムの値。

使用法
変換は実行されません。そのため、取得するデータは 16 ビットの整数である必要があります。

GetUInt32 メソッド

指定したカラムの値を、32 ビットの符号なし整数として取得します。

構文
```
UInt32 GetUInt32( int ordinal )
```

パラメータ
ordinal - 値を取得するカラムを表す序数。 番号付けは 0 から始まります。

戻り値
指定されたカラムの値。

使用法
変換は実行されません。そのため、取得するデータは 32 ビットの整数である必要があります。

GetUInt64 メソッド

指定したカラムの値を、64 ビットの符号なし整数として取得します。

構文
```
UInt64 GetUInt64( int ordinal )
```

パラメータ
ordinal - 値を取得するカラムを表す序数。 番号付けは 0 から始まります。

戻り値
指定されたカラムの値。

GetValue メソッド

指定した序数に対応するカラムの値を、ネイティブフォーマットで取得します。

構文
```
object GetValue( int ordinal )
```

パラメータ
ordinal - 値を取得するカラムを表す序数。 番号付けは 0 から始まります。

戻り値
対応するカラムの値。

実装
```
IDataRecord.GetValue
```

使用法
このメソッドでは、null のデータベースカラムに対して DBNull が返されます。

GetValues メソッド

現在のローのすべての属性カラムを取得します。

構文

```
int GetValues( object[ ] values )
```

パラメータ

value - 結果セット内のすべてのローを保持するオブジェクトの配列。

戻り値

配列内のオブジェクトの数。

実装

```
IDataRecord GetValues
```

使用法

- ほとんどのアプリケーションでは、各カラムを個別に取得するよりも、**GetValues** メソッドを使用してすべてのカラムを取得する方が効率的です。
- 対象のローに含まれるカラム数よりも配列長が短い `Object` 配列を渡すことができます。 `Object` 配列が保持しているデータ量だけが、配列にコピーされます。 対象のローに含まれるカラム数よりも配列長が長い `Object` 配列も渡すことができます。
- このメソッドでは、null のデータベースカラムに対して DBNull が返されます。
- 指定した序数に対応するカラムの値を、ネイティブフォーマットで取得します。

IsClosed プロパティ

`AseDataReader` がクローズされている場合は「true」を、それ以外の場合は「false」を返します。

構文

```
bool IsClosed
```

アクセス

読み込み専用

プロパティ値

AseDataReader がクローズされている場合は「true」、それ以外の場合は「false」。

実装

```
IDataReader.IsClosed
```

使用法

AseDataReader をクローズした後で呼び出すことができるのは、`IsClosed` と `RecordsAffected` プロパティだけです。

IsDBNull メソッド

カラムに null 値が含まれるかどうかを示す値を返します。

構文

```
bool IsDBNull( int ordinal )
```

パラメータ

ordinal - カラムを表す、0 から始まる序数。

戻り値

指定したカラムの値が DBNull と等価の場合は「true」、それ以外の場合は「false」。

実装

```
IDataRecord.IsDbNull
```

使用法

型指定された **Get** メソッド (**GetByte**、**GetChar** など) を呼び出す前に、例外の発生を防止する目的で、このメソッドを呼び出して null のカラム値をチェックしてください。

Item プロパティ

ネイティブフォーマットでのカラムの値を表します。 C# で、このプロパティは **AseDataReader** クラスのインデクサになります。

構文 1

```
object this[ int index ]
```

構文 2

```
object this[ string name ]
```

パラメータ

- **index –** カラムの序数。
- **name –** カラム名。

アクセス
読み込み専用

実装
```
IDataRecord.Item
```

NextResult メソッド

バッチ SQL 文の結果を読み込むときに、`AseDataReader` を次の結果へ進めます。

構文
```
bool NextResult( )
```

戻り値
さらに結果セットがある場合は「true」。 それ以外の場合は「false」。

実装
```
IDataReader.NextResult
```

使用法
バッチ SQL 文の実行によって生成される複数の結果を処理する目的で使用します。

デフォルトでは、データリーダは最初の結果に置かれます。

Read メソッド

結果セットの次のローを読み込み、`AseDataReader` をそのローに進めます。

構文
```
bool Read( )
```

戻り値
さらにローがある場合は「true」を返します。 それ以外の場合は「false」を返します。

実装
```
IDataReader.Read
```

使用法
AseDataReader のデフォルトの位置は、最初のレコードの前です。 そのため、データへのアクセスを開始するには Read を呼び出す必要があります。

例
次のコードは、結果内の 1 つのカラムの値をリストボックスに書き込みます。

```
while( reader.Read() )
{
   listResults.Items.Add( reader.GetValue( 0 ).ToString() );
}
listResults.EndUpdate();
reader.Close();
```

RecordsAffected プロパティ

SQL 文の実行によって変更、挿入、削除されたローの数を表します。

構文

```
int RecordsAffected
```

アクセス
読み込み専用

プロパティ値
変更、挿入、または削除されたローの数。 影響を受けたローがない場合または文が失敗した場合は 0、**Select** 文の場合は -1 になります。

実装

```
IDataReader.RecordsAffected
```

使用法

- 変更、挿入、または削除されたローの数。 影響を受けたローがない場合または文が失敗した場合、値は 0 です。**Select** 文の場合、値は -1 です。
- このプロパティの値は累積です。 たとえば、バッチモードで 2 つのレコードが挿入された場合、RecordsAffected の値は 2 になります。
- AseDataReader をクローズした後で呼び出すことができるのは、IsClosed と RecordsAffected プロパティだけです。

AseDbType 列挙型

Adaptive Server のデータ型を指定します。

データ型のマッピングの詳細については、表を参照してください。

メンバ

`Binary`

`Bit`

`Char`

`Date`

`DateTime`

`Decimal`

`Double`

`Float`

`Integer`

`Image`

`LongVarChar`

`Money`

`Nchar`

`Numeric`

`NVarChar`

`Real`

`SmallDateTime`

`SmallMoney`

`Text`

`Time`

`TimeStamp`

`TinyInt`

`UniChar`

`UniVarChar`

`VarBinary`

```
VarChar
```

注意： Numeric と Decimal の精度は、Adaptive Server の精度の 38 ではなく 26 に制限されています。

Adaptive Server ADO.NET でのデータ型のマッピング

Adaptive Server ADO.NET Data Provider でのデータ型のマッピング。

Adaptive Server のデータベースのデータ型	AseDbType 列挙型	.NET Dbtype 列挙型	.Net クラス名
binary	Binary	Binary	Byte[]
bigint	BigInt	Int64	Int64
bit	Bit	Boolean	Boolean
char	Char	AnsiStringFixed-Length	String
date	Date	Date	DateTime
datetime	DateTime	DateTime	DateTime
decimal	Decimal	Decimal	Decimal
double	Double	Double	Double
float(<16)	Real	Single	Single
float(>=16)	Double	Double	Double
image	Image	Binary	Byte[]
int[eger]	Integer	Int32	Int32
money	Money	Currency	Decimal
nchar	NChar	AnsiStringFixed-Length	String
nvarchar	NVarChar	AnsiString	String
numeric	Numeric	VarNumeric	Decimal
real	Real	Single	Single
smalldatetime	SmallDateTime	DateTime	DateTime
smallint	SmallInt	Int16	Int16

Adaptive Server のデータベースのデータ型	AseDbType 列挙型	.NET Dbtype 列挙型	.Net クラス名
smallmoney	SmallMoney	Currency	Decimal
text	Text	AnsiString	String
time	Time	Time	DateTime
timestamp	TimeStamp	Binary	Byte[]
tinyint	TinyInt	Byte	Byte
unichar	UniChar	StringFixedLength	String
unitext	Unitext	String	String
univarchar	UniVarChar	String	String
unsignedbigint	UnsignedBigint	UInt64	UInt64
unsignedint	UnsignedInt	UInt32	UInt32
unsignedsmallint	UnsignedSmallInt	UInt16	UInt16
varbinary	VarBinary	Binary	Byte[]
varchar	VarChar	AnsiString	String

AseDecimal 構造体

AseDecimal は、最大 38 桁の精度を持つ decimal または numeric 型の数値を表します。 AseDecimal は IComparable インタフェースを実装します。

AseDecimal コンストラクタ

AseDecimal 構造体の新しいインスタンスを初期化します。

構文 1

```
AseDecimal( AseDecimal asedecimal )
```

構文 2

```
AseDecimal( Decimal decimal )
```

構文 3

```
AseDecimal( Int32 precision, Int32 scale )
```

構文 4

```
AseDecimal( Int32 precision, Int32 scale, Byte[ ] value )
```

パラメータ

- **asedecimal –** 新しい `AseDecimal` の値を初期化する `AseDecimal` 構造体。
- **decimal –** 新しい `AseDecimal` の値を初期化する decimal 構造体。
- **precision –** ターゲットの精度の Int32 値。
- **scale –** ターゲットの位取りの Int32 値。
- **value –** ターゲットの値の配列 (バイト単位)。

AseDecimal のフィールド

AseDecimal のフィールドに関する情報を確認します。

MAXLENGTH
DataLength の最大値は 33 バイトです。

MAXOUTPUTPRECISION
出力精度の最大値は 77 桁です。

MAXOUTPUTSCALE
出力位取りの最大値は 77 桁です。

MAXPRECISION
許容される精度の最大値は 38 桁です。

MAXSCALE
許容される位取りの最大値は 38 桁です。

CompareTo メソッド

この `AseDecimal` の値をターゲットの `AseDecimal` またはオブジェクトと比較します。

構文

```
CompareTo( AseDecimal )
```

説明
この `AseDecimal` の値を *AseDecimal* 値と比較します。

構文

```
CompareTo( Object )
```

説明
この AseDecimal の値を *Object* の値と比較します。

等号演算子

2 つの AseDecimal インスタンスが等しいかどうかを示す値を返します。

構文
```
static bool operator ==( AseDecimal a, AseDecimal b )
```

戻り値
値が等しい場合は true。

Equals メソッド

この AseDecimal インスタンスと、オブジェクト *value* が示す値が等しいかどうかを示す値を返します。

構文
```
bool Equals( Object value )
```

パラメータ
value - 等しいかどうかがテストされる対象オブジェクト。

GetHashCode メソッド

ハッシュコードを返します。

構文
```
override int GetHashCode( )
```

戻り値
ハッシュコードを表す integer 値。

GreaterThan 演算子

a が *b* より大きいかどうかを示す値を返します。

構文
```
static bool operator >( AseDecimal a, AseDecimal b )
```

戻り値
a が *b* より大きい場合は true。

GreaterThanOrEqual 演算子

a が *b* 以上であるかどうかを示す値を返します。

構文

```
static bool operator >=( AseDecimal a, AseDecimal b )
```

戻り値

a が *b* 以上である場合は true。

IsNegative プロパティ

この AseDecimal が負の値なら true を返します。

構文

```
bool IsNegative { get; }
```

IsNull プロパティ

この AseDecimal が NULL 値なら true を返します。

構文

```
bool  IsNull { get; }
```

IsPositive プロパティ

この AseDecimal が正の値なら true を返します。

構文

```
bool  IsPositive { get; }
```

LessThan 演算子

a が *b* より小さいかどうかを示す値を返します。

構文

```
static bool operator <( AseDecimal a, AseDecimal b )
```

戻り値

a が *b* より小さい場合は true。

LessThanorEqual 演算子

a が *b* 以下であるかどうかを示す値を返します。

構文

```
static bool operator <=( AseDecimal a, AseDecimal b )
```

戻り値
a が *b* 以下である場合は true。

Parse メソッド

文字列値を `AseDecimal` 値に解析します。

構文

```
AseDecimal Parse( string s )
```

パラメータ
s - AseDecimal 値への解析対象の文字列。

戻り値
解析した文字列を表す `AseDecimal` 構造体。

Sign メソッド

`AseDecimal` 値の符号をチェックします。

構文

```
int Sign( AseDecimal value )
```

パラメータ
value - 設計がチェック対象である AseDecimal 構造体。

戻り値
NULL 値の場合は整数 `0`、正の値の場合は `1`、負の値の場合は `-1` をそれぞれ返します。

ToAseDecimal メソッド

指定された精度と位取りで、この AseDecimal を新しい AseDecimal に変換します。

構文

```
AseDecimal ToAseDecimal( int outputPrecision, int outputScale )
```

パラメータ

- **outputPrecision –** 出力のターゲットの精度。
- **outputScale –** 出力のターゲットの位取り。

戻り値
指定された精度と位取りの AseDecimal 構造体。

ToString メソッド

この AseDecimal の文字列表現を返します。

構文

```
string ToString( )
```

戻り値
この AseDecimal の文字列表現。

AseError クラス

データソースから返された警告またはエラーに関する情報を収集します。

基本クラス

```
Object
```

AseError には、コンストラクタはありません。

参照：

- エラー処理 (89 ページ)

ErrorNumber プロパティ

エラーメッセージの番号を表します。

構文

```
public int MessageNumber
```

アクセス
読み込み専用

Message プロパティ

エラーの簡単な説明を表します。

構文

```
public string Message
```

アクセス
読み込み専用

SqlState プロパティ

ANSI SQL 標準に従った、Adaptive Server 固有の 5 文字の SQL ステータスを表します。 エラーが複数の場所で発生している場合、5 文字のエラーコードはエラーの発生元を明らかにします。

構文

```
public string SqlState
```

アクセス
読み込み専用

ToString メソッド

エラーメッセージの完全なテキストを取得します。

構文

```
public string ToString( )
```

使用法
戻り値は、「AseError:」の後にメッセージが続く形式の文字列です。 次に例を示します。

```
AseError:UserId or Password not valid.
```

説明
メッセージのステータスを表します。 MsgNumber に対する変更子として使用されます。

構文
```
public int State
```

説明
メッセージの重大度を表します。

構文
```
public int Severity
```

説明
メッセージを送信しているサーバの名前を表します。

構文
```
public string ServerName
```

説明
メッセージの発生元であるストアドプロシージャまたはリモートプロシージャコール (RPC) の名前を表します。

構文
```
public string ProcName
```

説明
コマンドバッチまたはストアドプロシージャ内でエラーが発生した箇所の行番号を表します (該当する場合)。

構文
```
public int LineNum
```

説明
拡張メッセージに関連付けられます。

構文
```
public int Status
```

説明
このダイアログでアクティブなトランザクションが存在する場合にその現在ステータスを表します。

構文

```
public int TranState
```

説明

Adaptive Server サーバから送信されたエラーメッセージであるかどうかを表します。

構文

```
bool IsFromServer
```

使用法

戻り値は「true」または「false」です。

```
if (ex.Errors[0].IsInformation )
    MessageBox.Show("ASE has reported the following error: " +
ex.Errors[0].Message);
```

説明

Adaptive Server ADO.NET Data Provider から送信されたエラーメッセージであるかどうかを表します。

構文

```
bool IsFromClient
```

使用法

戻り値は「true」または「false」です。

```
if (ex.Errors[0].IsInformation )
    MessageBox.Show("ASE has reported the following error: " +
ex.Errors[0].Message);
```

説明

メッセージがエラーを表すかどうか判断されます。

構文

```
bool IsError
```

使用法

戻り値は「true」または「false」です。

```
if ( ! ex.Errors[0].IsInformation )
    MessageBox.Show("Error: " + ex.Errors[0].Message):
```

説明
メッセージが、状況が正常でない可能性があることを示す警告であるかどうかを表します。

構文
```
bool IsWarning
```

使用法
戻り値は「true」または「false」です。

```
if ( ! ex.Errors[0].IsInformation )
    MessageBox.Show("Error: " + ex.Errors[0].Message):
```

説明
アクティブなカタログが変更されたなどの情報を知らせるメッセージであるかどうかを表します。

構文
```
bool IsInformation
```

使用法
戻り値は「true」または「false」です。

```
if ( ! ex.Errors[0].IsInformation )
    MessageBox.Show("Error: " + ex.Errors[0].Message):
```

AseErrorCollection クラス

Adaptive Server ADO.NET Data Provider によって生成されたすべてのエラーを収集します。

基本クラス
```
Object
```

実装
```
ICollection, IEnumerable
```

AseErrorCollection には、コンストラクタはありません。 通常、**AseErrorCollection** は AseException.Errors または InfoMessageArgs プロパティから取得されます。

参照：
- エラー処理 (89 ページ)
- Errors プロパティ (200 ページ)

CopyTo メソッド

AseErrorCollection の要素を配列にコピーします。配列内の指定したインデックスからコピーが開始されます。

構文

```
void CopyTo( Array array, int index )
```

パラメータ

- **array –** 要素のコピー先となる配列。
- **index –** 配列の開始インデックス。

実装

```
ICollection.CopyTo
```

Count プロパティ

コレクション内のエラーの数を表します。

構文

```
int Count
```

アクセス
読み込み専用

実装

```
ICollection.Count
```

Item プロパティ

指定したインデックスの位置にあるエラーを表します。

構文

```
AseError this[ int index ]
```

パラメータ
index - 取得するエラーの 0 から始まるインデックス。

プロパティ値
指定したインデックスの位置にあるエラーが記録された AseError。

アクセス
読み込み専用

AseException クラス

Adaptive Server が警告またはエラーを返したときに発生する例外を表します。

基本クラス

```
SystemException
```

AseException には、コンストラクタはありません。 通常、AseException オブジェクトは catch 内で宣言されます。 次に例を示します。

```
...
catch (AseException ex )
{
   MessageBox.Show(ex.Message, "Error" );
}
```

参照：

- エラー処理 (89 ページ)

Errors プロパティ

1 つ以上の AseError オブジェクトのコレクションを表します。

構文

```
AseErrorCollection Errors
```

アクセス

読み込み専用

使用法

AseErrorCollection クラスには、AseError クラスのインスタンスが常に 1 つ以上含まれます。

Message プロパティ

エラーを説明するテキストを返します。

構文

```
string Message
```

アクセス

読み込み専用

使用法
このメソッドは、最初の `AseError` のメッセージを返します。

AseFailoverException クラス

HA クラスタに設定されているセカンダリサーバへの Adaptive Server のフェールオーバが成功したときに返される例外を表します。

基本クラス
```
AseException
```

`AseFailoverException` にコンストラクタはありません。 通常、AseFailoverException オブジェクトは catch 内で宣言されます。 次に例を示します。

```
...
catch( AseFailoverException ex )
{
   MessageBox.Show( ex.Message, "Warning!" );
}
```

参照：
- AseException クラス (200 ページ)

Errors プロパティ

データソースから送信された警告のコレクションを表します。

構文
```
AseErrorCollection Errors
```

アクセス
読み込み専用

Message プロパティ

データソースから送信されたエラーの完全なテキストを表します。

構文
```
string Message
```

アクセス
読み込み専用

ToString メソッド

InfoMessage イベントの文字列表現を取得します。

構文

```
string ToString( )
```

戻り値

InfoMessage イベントを表す文字列。

AseInfoMessageEventArgs クラス

InfoMessage イベントハンドラに渡されたイベント引数を表します。

基本クラス

EventArgs

Errors プロパティ

サーバによって返された実際のエラーオブジェクトのコレクションを表します。

構文

```
AseErrorCollection Errors
```

アクセス

読み込み専用

Message プロパティ

エラーメッセージを表します。

構文

```
string Message
```

アクセス

読み込み専用

AseInfoMessageEventHandler デリゲート

AseConnection の **InfoMessage** イベントを処理するメソッドを表します。

構文

```
void AseInfoMessageEventHandler ( object sender, AseInfoMessageEventArgs
e )
```

パラメータ

- **sender** – イベントの発生元。
- **e** – イベントデータが格納された `AseInfoMessageEventArgs` オブジェクト。

AseParameter クラス

AseCommand のパラメータと、オプションとして `DataSet` カラムへのマッピングを表します。

基本クラス

```
MarshalByRefObject
```

実装

```
IDbDataParameter, IDataParameter
```

AseParameter コンストラクタ

AseParameter コンストラクタに関する情報を確認します。

構文 1

```
AseParameter( )
```

構文 2

```
AseParameter( string parameterName, object value )
```

構文 3

```
AseParameter( string parameterName, AseDbType dbType )
```

構文 4

```
AseParameter( string parameterName, AseDbType dbType, int size )
```

構文5

```
AseParameter( string parameterName, AseDbType dbType, int size,
string sourceColumn )
```

構文6

```
AseParameter( string parameterName, AseDbType dbType, int size,
ParameterDirection direction, bool isNullable, byte precision, byte
scale,
string sourceColumn, DataRowVersion sourceVersion, object value )
```

パラメータ

- **value –** パラメータの値であるオブジェクト。
- **size –** パラメータの長さ。
- **sourceColumn –** マッピング対象のソースカラムの名前。
- **parameterName –** パラメータの名前。
- **dbType –** AseDbType 値の 1 つ。
- **direction –** ParameterDirection 値の 1 つ。
- **isNullable –** フィールドの値として null を許可する場合は true、許可しない場合は false。
- **precision –** Value の解決に適用する、小数点以上と以下両方の総桁数。
- **scale –** Value の解決に適用する、小数点以下の桁数。
- **sourceVersion –** DataRowVersion 値の 1 つ。

AseDbType プロパティ

パラメータの `AseDbType`。

構文

```
AseDbType AseDbType
```

アクセス
読み込み/書き込み

使用法

- `AseDbType` と `DbType` はリンクしています。 このため、`DbType` を設定すると、`AseDbType` がサポートされている `AseDbType` に変更されます。
- この値は `AseDbType` 列挙型のメンバである必要があります。

DbType プロパティ

パラメータの `DbType` を表します。

構文

```
DbType DbType
```

アクセス
読み込み/書き込み

使用法

- AseDbType と DbType はリンクしています。 このため、DbType を設定すると、AseDbType がサポートされている AseDbType に変更されます。
- 値は、`DbType` 列挙型のメンバである必要があります。

Direction プロパティ

パラメータが入力専用、出力専用、双方向、またはストアドプロシージャの戻り値パラメータのいずれであるかを示す値を表します。

構文

```
ParameterDirection Direction
```

アクセス
読み込み/書き込み

使用法
ParameterDirection が出力であり、関連付けられた **AseCommand** の実行が値を返さなかった場合、AseParameter には null 値が格納されます。 最後の結果セットの最後のローが読み込まれたら、Output、InputOut、ReturnValue の各パラメータが更新されます。

IsNullable プロパティ

パラメータが null 値を受け入れるかどうかを示す値を表します。

構文

```
bool IsNullable
```

アクセス
読み込み/書き込み

使用法
null 値を受け入れる場合は「true」、それ以外の場合は「false」(デフォルト)。 null 値は、DBNull クラスを使用して操作します。

ParameterName プロパティ

`AseParameter` の名前を表します。

構文
```
string ParameterName
```

アクセス
読み込み/書き込み

実装
```
IDataParameter.ParameterName
```

使用法

- Adaptive Server ADO.NET Data Provider は、***@name*** パラメータによって示される位置指定パラメータを使用します。
- デフォルトは空の文字列です。
- 出力パラメータ (準備のありなしに関係なく) には、ユーザ指定のデータ型が必要です。

Precision プロパティ

Value プロパティの表現に使用される最大桁数を表します。

構文
```
byte Precision
```

アクセス
読み込み/書き込み

実装
```
IDbDataParameter.Precision
```

使用法

- このプロパティの値は、`Value` プロパティの記述に使用される最大桁数です。 デフォルト値は 0 で、これは Adaptive Server ADO.NET Data Provider が `Value` プロパティの精度を設定することを意味します。

- `Precision` プロパティは、decimal と numeric 型の入力パラメータにのみ使用します。

Scale プロパティ

`Value` の解決に適用する小数点以下の桁数を表します。

構文

```
byte Scale
```

アクセス
読み込み/書き込み

実装

```
IDbDataParameter.Scale
```

使用法
デフォルトは 0 です。 Scale プロパティは、decimal と numeric 型の入力パラメータにのみ使用します。

Size プロパティ

カラム内のデータのバイト単位での最大サイズを表します。

構文

```
int Size
```

アクセス
読み込み/書き込み

実装

```
IDbDataParameter.Size
```

使用法

- このプロパティの値は、カラム内のデータのバイト単位での最大サイズです。デフォルト値は、パラメータ値から推測されます。
- `Size` プロパティは、binary と string 型で使用します。
- 可変長のデータ型では、`Size` プロパティはサーバに送信するデータの最大量を示します。 たとえば、`Size` プロパティを使用して、サーバに送信する文字列値のデータ量を、最初の 100 バイトまでに制限できます。
- 明示的に設定しない場合は、指定したパラメータ値の実際のサイズから、このサイズが推測されます。 固定長のデータ型では、`Size` の値は無視されます。この情報は参照目的で取得できます。この場合、パラメータ値をサーバに送信

するときに Adaptive Server ADO.NET Data Provider が使用する最大バイト数が返されます。

SourceColumn プロパティ

DataSet にマッピングされ、値のロードや戻り値に使用されるソースカラムの名前を表します。

構文

```
string SourceColumn
```

アクセス
読み込み/書き込み

実装

```
IDbDataParameter.SourceColumn
```

使用法
SourceColumn に空の文字列以外が設定されている場合、パラメータの値は SourceColumn 名で指定されるカラムから取得されます。 Direction が Input に設定されている場合、値は DataSet から取得されます。 Direction が Output に設定されている場合、値はデータソースから取得されます。 Direction が InputOutput の場合は、両方の組み合わせです。

SourceVersion プロパティ

Value をロードするときに使用する DataRowVersion を表します。

構文

```
DataRowVersion SourceVersion
```

アクセス
読み込み/書き込み

実装

```
IDbDataParameter.SourceVersion
```

使用法
Update の際に、パラメータ値が Current と Original のどちらに設定されているかを判別するために、UpdateCommand によって使用されます。 これによって、プライマリキーが更新できるようになります。 InsertCommand と DeleteCommand では、このプロパティは無視されます。 このプロパティは、Item プロパティ、

または DataRow オブジェクトの **GetChildRows** メソッドによって使用される DataRow のバージョンに設定されます。

ToString メソッド

ParameterName が含まれる文字列を表します。

構文

```
string ToString( )
```

アクセス
読み込み/書き込み

Value プロパティ

パラメータの値を表します。

構文

```
object Value
```

アクセス
読み込み/書き込み

実装

```
IDataParameter.Value
```

使用法

- 入力パラメータでは、サーバに送信される **AseCommand** に値がバインドされます。 出力パラメータと戻り値パラメータでは、**AseCommand** の完了時と AseDataReader のクローズ後に、値が設定されます。
- null パラメータ値をサーバに送信する場合は、null ではなく DBNull を指定する必要があります。システムにおける null 値とは、値のない空のオブジェクトを意味します。
- アプリケーションでデータベースの型を指定する場合は、Adaptive Server ADO.NET Data Provider がデータをサーバに送信するときに、バインドされた値がその型に変換されます。 Adaptive Server ADO.NET Data Provider が IConvertible インタフェースをサポートする場合は、どのような型の値でも変換されます。指定した型が値と互換性を持たないときは、変換エラーが発生する場合があります。
- Value の設定によって、DbType と AseDbType の両方のプロパティが推測されます。
- Value プロパティは Update によって上書きされます。

AseParameterCollection クラス

AseCommand のすべてのパラメータと、オプションとして DataSet カラムへのマッピングを表します。

基本クラス

```
Object
```

実装

```
ICollection, IEnumerable, IDataParameterCollection
```

使用法

AseParameterCollection には、コンストラクタはありません。

AseParameterCollection は **AseCommand.Parameters** プロパティから取得します。

参照：

- Parameters プロパティ (130 ページ)

Add メソッド

AseParameter を AseCommand に追加します。

構文 1

```
public AseParameter Add( AseParameter P )
```

構文 2

```
public AseParameter Add( object P)
```

構文 3

```
public AseParameter Add( string name, AseDbType dataType )
```

構文 4

```
public AseParameter Add( string name, object value )
```

構文 5

```
public AseParameter Add( string name, AseDbType dataType,
AseParameter size)
```

構文 6

```
public AseParameter Add( string name, AseDbType dataType, int size,
string sourceColumn)
```

構文 7

```
public AseParameter Add( string parameterName, AseDbType dbType,
int size, ParameterDirection direction, Boolean isNullable, Byte
precision,
Byte scale, string sourceColumn, DataRowVersion sourceVersion,
object
value)
```

パラメータ

- **value –** 構文 1 と 2 の場合、*value* はコレクションに追加する `AseParameter` オブジェクトです。 構文 3 の場合、*value* は接続に追加するパラメータの値です。
- **parameterName –** パラメータの名前。
- **aseDbType –** `AseDbType` 値の 1 つ。
- **size –** カラムの長さ。
- **sourceColumn –** ソースカラムの名前。

戻り値

Add は、**AseCommand** によって使用されるパラメータリストにパラメータを挿入します。 戻り値は、リストに追加された新しいパラメータです。

Clear メソッド

コレクションからすべての項目を削除します。

構文

```
void Clear( )
```

実装

```
IList.Clear
```

Contains メソッド

`AseParameter` がコレクション内にあるかどうかを示す値を返します。

構文 1

```
bool Contains( object value )
```

構文 2

```
bool Contains( string value )
```

パラメータ

value - 検索対象の AseParameter オブジェクトの値。 構文 2 の場合は名前になります。

戻り値
AseParameter がコレクション内にある場合は 「true」、それ以外の場合は「false」。

実装

- 構文 1 は **IList.Contains** を実装します。
- 構文 2 は **IDataParameterCollection.Contains** を実装します。

CopyTo メソッド

AseParameter オブジェクトを **AseParameterCollection** から、指定した配列にコピーします。

構文

```
void CopyTo( array array int index )
```

パラメータ

- **array –** AseParameter オブジェクトのコピー先となる配列。
- **index –** 配列の開始インデックス。

実装

```
ICollection.CopyTo
```

Count プロパティ

コレクション内にある AseParameter オブジェクトの数を表します。

構文

```
int Count
```

アクセス
読み込み専用

実装

```
ICollection.Count
```

IndexOf メソッド

コレクション内での AseParameter の位置を取得します。

構文 1

```
int IndexOf( object value )
```

構文 2

```
int IndexOf( string parameterName )
```

パラメータ

- **value –** 位置を確認する `AseParameter` オブジェクト。
- **parameterName –** 位置を確認する `AseParameter` オブジェクトの名前。

戻り値
コレクション内での `AseParameter` の 0 を基準にした位置。

実装

- 構文 1 では **IList.IndexOf** を実装します。
- 構文 2 では **IDataParameterCollection.IndexOf** を実装します。

Insert メソッド

コレクション内の指定したインデックスに `AseParameter` を挿入します。

構文

```
void Insert( int index object value )
```

パラメータ

- **index –** パラメータの挿入先となる、コレクション内の 0 から始まるインデックス。
- **value –** コレクションに追加する `AseParameter`。

実装

```
IList.Insert
```

Item プロパティ

指定したインデックスまたは名前を持つ `AseParameter` を表します。

構文 1

```
AseParameter this[ int index ]
```

構文 2

```
AseParameter this[ string parameterName ]
```

パラメータ

- **index –** 取得するパラメータの 0 から始まるインデックス。
- **parameterName –** 取得するパラメータの名前。

プロパティ値
AseParameter

アクセス
読み込み/書き込み

使用法
C# では、このプロパティは **AseParameterCollection** クラスのインデクサになります。

Remove メソッド

メソッドに渡された `AseParameter` をコレクションから削除します。

構文

```
void Remove( object value )
```

パラメータ
value - コレクションから削除する `AseParameter` オブジェクト。

実装

```
IList.Remove
```

RemoveAt メソッド

パラメータのインデックスまたは名前に基づいて、コレクションからパラメータを削除します。

構文 1

```
void RemoveAt( int index )
```

構文 2

```
void RemoveAt( string parameterName )
```

パラメータ

- **index –** 削除するパラメータの 0 から始まるインデックス。
- **parameterName –** 削除する `AseParameter` オブジェクトの名前。

実装

- 構文 1 では **IList.RemoveAt** を実装します。
- 構文 2 では **IDataParameterCollection.RemoveAt** を実装します。

AseRowsCopiedEventArgs クラス

AseRowsCopiedEvent の引数データを表します。

基本クラス
EventArgs

AseRowsCopiedEventArgs コンストラクタ

AseRowsCopiedEventArgs クラスの新しいインスタンスを初期化します。

構文

```
void AseRowsCopiedEventArgs( int num )
```

パラメータ
num - コピーするローの数。

Abort プロパティ

バルクコピー操作をアボートするかどうかを指定します。

構文

```
bool Abort
```

アクセス
読み込み/書き込み

RowCopied プロパティ

コピーされたローの数。

構文

```
int RowCopied
```

アクセス
読み込み/書き込み

AseRowsCopiedEventHandler デリゲート

AseRowsCopied イベントを処理するメソッドを表します。

構文

```
void AseRowsCopiedEventHandler( Object sender, AseRowsCopiedEventArgs e )
```

パラメータ

- **e –** イベント引数。
- **sender –** イベントをトリガしたオブジェクト。

AseRowUpdatedEventArgs クラス

RowUpdated イベントのデータを提供します。

基本クラス

```
RowUpdatedEventArgs
```

AseRowUpdatedEventArgs コンストラクタ

AseRowUpdatedEventArgs クラスの新しいインスタンスを初期化します。

構文

```
AseRowUpdatedEventArgs( DataRow dataRow, IDbCommand
command, StatementType statementType, DataTableMapping
tableMapping )
```

パラメータ

- **dataRow – Update** によって送信された DataRow。
- **command – Update** の呼び出し時に実行される **IDbCommand**。
- **statementType –** 実行するクエリの種類を指定する **StatementType** 値のいずれか。
- **tableMapping – Update** によって送信された DataTableMapping。

Command プロパティ

Update を呼び出したときに実行される **AseCommand** を表します。

構文

```
AseCommand Command
```

アクセス
読み込み専用

Errors プロパティ

コマンドの実行時に、Adaptive Server によって生成されたすべてのエラーを表します。 RowUpdatedEventArgs から継承されます。

構文
```
Exception Errors
```

プロパティ値
コマンドの実行時に、Adaptive Server によって生成されたエラー。

アクセス
読み込み/書き込み

RecordsAffected プロパティ

SQL 文の実行によって変更、挿入、削除されたローの数を表します。 RowUpdatedEventArgs から継承されます。

構文
```
int RecordsAffected
```

プロパティ値
変更、挿入、または削除されたローの数。影響を受けたローがない場合または文が失敗した場合は 0、**Select** 文の場合は -1 です。

アクセス
読み込み専用

Row プロパティ

Update によって送信された DataRow を表します。 **RowUpdatedEventArgs** から継承されます。

構文
```
DataRow Row
```

アクセス
読み込み専用

StatementType プロパティ

実行された SQL 文の種類を表します。 **RowUpdatedEventArgs** から継承されます。

構文

```
StatementType StatementType
```

アクセス
読み込み専用

使用法
StatementType は、**Select**、**Insert**、**Update**、または **Delete** のいずれかです。

Status プロパティ

Command プロパティの UpdateStatus を表します。 **RowUpdatedEventArgs** から継承されます。

構文

```
UpdateStatus Status
```

プロパティ値
次の UpdateStatus 値のいずれか。 **Continue** (デフォルト)、**ErrorsOccurred**、**SkipAllRemainingRows**、または **SkipCurrentRow**。

アクセス
読み込み/書き込み

TableMapping プロパティ

Update によって送信された DataTableMapping。 **RowUpdatedEventArgs** から継承されます。

構文

```
DataTableMapping TableMapping
```

アクセス
読み込み専用

AseRowUpdatingEventArgs クラス

RowUpdating イベントのデータを提供します。

基本クラス

```
RowUpdatingEventArgs
```

AseRowUpdatingEventArgs コンストラクタ

AseRowUpdatingEventArgs クラスの新しいインスタンスを初期化します。

構文

```
AseRowUpdatingEventArgs( DataRow row, IDbCommand command,
StatementType statementType, DataTableMapping tableMapping )
```

パラメータ

- **row –** 更新する **DataRow**。
- **command –** 更新時に実行する **IDbCommand**。
- **statementType –** 実行するクエリの種類を指定する StatementType 値のいずれか。
- **tableMapping – Update** によって送信された DataTableMapping。

Command プロパティ

Update の実行時に実行する **AseCommand** を表します。

構文

```
AseCommand Command
```

アクセス
読み込み/書き込み

Errors プロパティ

コマンドの実行時に、Adaptive Server によって生成されたすべてのエラーを表します。 **RowUpdatingEventArgs** から継承されます。

構文

```
Exception Errors
```

プロパティ値
コマンドの実行時に、Adaptive Server によって生成されたエラー。

アクセス
読み込み/書き込み

Row プロパティ

Update によって送信された DataRow を表します。 **RowUpdatingEventArgs** から継承されます。

構文
```
DataRow Row
```

アクセス
読み込み専用

StatementType プロパティ

実行された SQL 文の種類を表します。 **RowUpdatingEventArgs** から継承されます。

構文
```
StatementType StatementType
```

アクセス
読み込み専用

使用法
StatementType は、**Select**、**Insert**、**Update**、または **Delete** のいずれかです。

Status プロパティ

Command プロパティの UpdateStatus を表します。 **RowUpdatingEventArgs** から継承されます。

構文
```
UpdateStatus Status
```

プロパティ値
次の UpdateStatus 値のいずれか。 **Continue** (デフォルト)、**ErrorsOccurred**、**SkipAllRemainingRows**、または **SkipCurrentRow**。

アクセス
読み込み/書き込み

TableMapping プロパティ

Update によって送信された DataTableMapping を表します。 **RowUpdatingEventArgs** から継承されます。

構文

```
DataTableMapping TableMapping
```

アクセス
読み込み専用

AseRowUpdatedEventHandler デリゲート

AseDataAdapter の **RowUpdated** イベントを処理するメソッドを表します。

構文

```
void AseRowUpdatedEventHandler ( object sender, AseRowUpdatedEventArgs
e )
```

パラメータ

- **sender –** イベントの発生元。
- **e –** イベントデータが格納された **AseRowUpdatedEventArgs**。

AseRowUpdatingEventHandler デリゲート

AseDataAdapter の **RowUpdating** イベントを処理するメソッドを表します。

構文

```
void AseRowUpdatingEventHandler ( object sender, AseRowUpdatingEventArgs
e )
```

パラメータ

- **sender –** イベントの発生元。
- **e –** イベントデータが格納された **AseRowUpdatingEventArgs**。

AseTransaction クラス

SQL トランザクションを表します。

基本クラス

```
Object
```

実装

```
IDbTransaction
```

使用法

- **AseTransaction** にコンストラクタはありません。 `AseTransaction` オブジェクトを取得するには、**AseConnection.BeginTransaction()** メソッドを使用します。
- トランザクションにコマンドを関連付けるには、**AseCommand.Transaction** プロパティを使用します。

参照：

- トランザクション処理 (86 ページ)
- BeginTransaction メソッド (149 ページ)
- AseCommand オブジェクトを使用したローの挿入、更新、削除 (51 ページ)

Commit メソッド

データベーストランザクションをコミットします。

構文

```
void Commit( )
```

実装

```
IDbTransaction.Commit
```

Connection プロパティ

トランザクションに関連付けられている `AseConnection` オブジェクトを表します。トランザクションが無効になっている場合は null 参照 (Visual Basic の場合は「Nothing」) です。

構文

```
AseConnection Connection
```

アクセス
読み込み専用

使用法
1 つのアプリケーションが複数のデータベース接続を保持し、それぞれの接続で 0 または 1 つのトランザクションが実行される場合があります。 このプロパティを使用すると、BeginTransaction によって作成された特定のトランザクションに関連付けられている接続オブジェクトを識別できます。

IsolationLevel プロパティ

このトランザクションの独立性レベルを指定します。

構文
```
IsolationLevel IsolationLevel
```

アクセス
読み込み専用

プロパティ値
このトランザクションの独立性レベル。 **ReadCommitted** (デフォルト)、**ReadUncommitted**、**RepeatableRead**、または **Serializable** を指定できます。

実装
```
IDbTransaction.IsolationLevel
```

Rollback メソッド

データベーストランザクションをロールバックします。

構文
```
void Rollback( )
```

実装
```
IDbTransaction.Rollback
```

使用法
Rollback は同期的です。 最初に `BeginTransaction()` を呼び出してから、返された `AseTransaction` に対して **Rollback** を呼び出します。

TraceEnterEventHandler デリゲート

TraceEnter イベントを処理するメソッドを表します。

構文

```
void TraceEnterEventHandler( AseConnection connection,
   Object source, string method, Object[] parameters)
```

パラメータ

- **connection –** イベントが発生した接続。
- **source –** イベントをトリガしたオブジェクト。
- **method –** 入力されたメソッド。
- **parameters –** 入力されたメソッドのパラメータ。

TraceExitEventHandler デリゲート

TraceExit イベントを処理するメソッドを表します。

構文

```
void TraceExitEventHandler( AseConnection connection, Object source,
string method, Object[] returnValue)
```

パラメータ

- **connection –** イベントが発生した接続。
- **source –** イベントをトリガしたオブジェクト。
- **method –** 終了したメソッド。
- **returnValue –** 終了したメソッドの戻り値。

用語解説

Adaptive Server® Enterprise ADO.NET Data Provider で使用される用語の解説です。

- **Adaptive Server Enterprise –** Sybase のクライアント/サーバアーキテクチャにおけるサーバ。 Adaptive Server Enterprise は、複数のデータベースと複数のユーザを管理します。ディスク上にあるデータの実際のロケーションを監視し、論理データ記述から物理データ記憶領域へのマッピングを管理します。メモリ内のデータキャッシュとプロシージャキャッシュの保守も行います。
- **ADO .Net (ActiveX Data Objects for .NET) –** ADO .Net (ActiveX Data Objects for .NET) は、プログラマがデータおよびデータサービスにアクセスするときに使用できる一連のコンピュータソフトウェアコンポーネントであり、Microsoft .NET Framework に付属している基本クラスライブラリの一部となっています。 通常、ADO .Net は、プログラマがリレーショナルデータベースシステムに格納されているデータにアクセスしたり、このデータを変更したりするときに使用されますが、非リレーショナルソースのデータへのアクセスにも使用できます。
- **ADO.NET データプロバイダ (ADO.NET data provider) –** ADO.NET データプロバイダは、ADO.NET コンシューマとデータソースとの対話を可能にするソフトウェアコンポーネントです。
- **BLOB –** バイナリラージオブジェクト
- **CipherSuite –** SSL 対応アプリケーションで使用されるキー交換アルゴリズム、ハッシュメソッド、暗号化メソッドの優先リスト。
- **CLR (共通言語ランタイム (Common Language Runtime)) –** CLR は Microsoft の .NET framework の仮想マシンコンポーネントで、.NET プログラムの実行を管理します。
- **接続フェールオーバ (connection failover) –** 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。
- **接続マイグレーション (connection migration) –** 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。
- **接続プール (connection pooling) –** 接続プールは、今後、データベースへの要求が必要になったときに接続を再利用できるように管理されたデータベース接続のキャッシュです。

- **DAAB (Data Access Application Block) –** DAAB はクラスのコレクションで、データベースインスタンスの作成やデータベースレコードの更新など、一般的なデータベース機能を簡素化します。
- **データプロバイダ (data provider) –** .Net でサポートされている任意の言語を使用して、サーバからデータにアクセスできるようにします。
- **データ検索 (data retrieval) –** データベース管理において、データ検索とは、データベースから必要なデータを抽出することに関連します。
- **復号化 (decryption) –** 暗号テキストのメッセージをプレーンテキスト形式に変換するプロセス。
- **分散トランザクション (distributed transaction) –** 分散トランザクションは、複数のネットワークホストが関与する 1 つのまとまったオペレーションです。
- **暗号化 (encryption) –** プレーンテキストのメッセージを暗号テキストに変換するプロセス。
- **エラー処理 (error handling) –** Transact-SQL プログラマが利用できる処理。コードに基づいてエラーやエラーメッセージを表示します。
- **GAC (グローバルアセンブリキャッシュ (Global Assembly Cache)） –** GAC は Microsoft のプラットフォームで使用される、マシン全体の .NET アセンブリです。
- **Kerberos –** Kerberos は、業界標準のネットワーク認証システムで、簡単なログイン認証と相互のログイン認証を提供します。
- **LDAP (Lightweight Directory Access Protocol) –** LDAP は、ディレクトリサービスへのアクセスの業界標準です。
- **LINQ (Language-Integrated Query) –** LINQ は、クエリを優れたコンセプトとしてあらゆる Microsoft .NET 言語に導入するプログラミングモデルです。
- **オブジェクト (object) –** 情報が含まれた、または情報を受け取る受動的エンティティ。その中の情報は変更できません。 Adaptive Server のオブジェクトには、ロー、テーブル、データベース、ストアドプロシージャ、トリガ、デフォルト、ビューがあります。「データベースオブジェクト」参照。
- **ODBC (Open Database Connectivity) –** Microsoft によって定義された ODBC インタフェースは、Windows や Windows NT 環境でのデータベース管理システムの標準インタフェースです。
- **OLE DB プロバイダ (OLE DB provider) –** OLE DB プロバイダは、OLE DB コンシューマとデータソースとの対話を可能にするソフトウェアコンポーネントです。
- **パラメータ (parameter) –** パラメータは、サブルーチンでそのサブルーチンの入力として指定されたデータの引数を参照するために使用される特殊な変数です。

- **パスワードの期限 (password expiration) –** 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。
- **プライベートキー (private key) –** 非対称アルゴリズムにおいて秘密にする必要があるシークレットキー。
- **パブリックキー (public key) –** 非対称アルゴリズムにおいて公開できるシークレットキー。
- **スキーマ (schema) –** 特定のスキーマ名とスキーマ権限識別子に関連付けられたオブジェクトの集まり。 オブジェクトには、テーブル、ビュー、ドメイン、制約、アサーション、権限などがあります。 スキーマは、create schema 文によって作成されます。
- **SSL (Secure Sockets Layer) –** SSL は、クライアントからサーバ、およびサーバからサーバへの接続で、ネットワーク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。
- **トランザクション処理 (transaction processing) –** トランザクション処理は、トランザクションと呼ばれる、それ以上分割不可能な個々のオペレーションに分割された情報処理です。 各トランザクションは、完全な単位として終了 (成功または失敗) する必要があります。中間の状態に留まることはできません。
- **ダウン時間ゼロ (zero-downtime) –** 短時間で終了する、システムまたはデータベースの移行またはアップグレード。

索引

A

C

D

E

G

H

K

L

P

S

T

え

お

く

こ

さ

し

せ

ち

て