



## 第1回 C#にふれる(1) - データベースの接続・切断とSQLの実行

沖林 正紀

### 1. はじめに

これまではサーバ・サイドJavaとして知られているサーブレット(Servlet)、JSP(JavaServer Pages)、EJB(Enterprise JavaBeans)やJava言語によるWebサービスについてご紹介してきましたが、本シリーズでは、.NET(ドットネット)環境でのアプリケーション開発言語であるC#(シーシャープ)を取り上げます。データベースに接続してSQLを実行するところから、GUIアプリケーションやXML Webサービスを開発するまでのアプローチをご紹介していきますので、どうぞよろしくお願いたします。

### 2. C#アプリケーションの開発・実行環境

まず、C#とそのアプリケーション開発環境である.NET Framework(ドットネットフレームワーク)について概要にふれておきます。詳細は日本のマイクロソフト株式会社および米国のMicrosoft Corporation(以下総称してマイクロソフト社と記します)の該当するサイトをご参照ください。

#### 2-1. .NETと.NET Framework

.NET(ドットネット)という場合は、広くマイクロソフト社の".NET戦略"を指すとされています。この戦略によって実現しようとしているのは、XML Webサービスです。「Webサービス」という言葉が入っているとおり、以前の連載「[Webサービスをはじめよう](#)」でご紹介したWebサービスと方向性は同じものと考えてよいでしょう。なぜなら、必要とされる技術的な基盤はSOAP, WSDL, UDDIなど、これまでにご紹介してきたものと同じだからです。そしてこれらの仕様は主にIBMとマイクロソフト社とが協力して策定したものです。

.NET Framework(ドットネット・フレームワーク)とは、こうしたXML Webサービスに対応できるプログラムを開発するためのライブラリ群(開発環境)をいいます。.NET Frameworkでは、それに対応するプログラムをC#をはじめとした各種の言語で記述することができ、それをIL(Intermediate Language)という中間言語にコンパイルします。そしてIL形式のモジュールを実際のプラットフォーム上で実行するのがCLR(Common Language Runtime:共通言語ランタイム)という実行環境です。これも.NET Frameworkに含まれています。

C#やILの実行環境の仕様はECMAという標準化団体に提出され、それぞれECMA-334, ECMA-335 という仕様として、PDF文書として入手することができます。ただしILの実行環境はCLI(Common Language Infrastructure)という名称になっています。

#### 2-2. .NET FrameworkでのC#プログラム開発

C#によるプログラムを開発し、実行するためには.NET Frameworkが必要となります。この連載では、OSとしてWindows 2000 Professionalに「.NET Framework SDK ベータ2 日本語版」をインストールした環境でC#のプログラムを開発する例をご紹介していきます。

「.NET Framework SDK ベータ2 日本語版」をインストールしますと、スタートメニューの「プログラム」の中に「Microsoft .NET Framework SDK」という項目が追加され、その中には画面2-1のように「Documentation」「Samples and QuickStart Tutorials」「Tools」「概要」というメニューが追加されます。このメニューから以下のものが実

DB2の.NETサポートについて  
DB2の.NETサポートはDB2 V8.1のFixpak(フィックスパック)で提供される予定です

→ [C#さん、どうぞよろしく！インデックスへ](#)

### コンテンツ

#### 1. はじめに

#### 2. C#アプリケーションの開発・実行環境

##### 2-1. .NETと.NET Framework

##### 2-2. .NET FrameworkでのC#プログラム開発

#### 3. C#によるDBアプリケーションの開発

##### 3-1.データベースの接続と切断

##### 3-2.SQLを実行する

#### 4. おわりに

### 関連リンク

- [WebSphere](#)
- [SOAP](#)
- [WSDL](#)
- [UDDI](#)
- [Java](#)
- [Web Services for DB2 Universal Database \(DB2 UDB\)](#)
- [Web Services and UDDI](#)
- [Webサービスをはじめよう](#)

行されます。なお「概要」以外は文章がすべて英語です。

#### Documentation

.NETに関する各種のドキュメントがヘルプウィンドウに表示できる

#### Samples and QuickStart Tutorials

C#やVisual Basic .NET(VB.NET)によるサンプルのソースコードを読んだり、それを実行することができる

#### Tools

各種のデバッグ用ツールやWSDLが示すWebサービスを呼び出すプログラムを生成するツールなどのコマンド名と概要を紹介するHTML文書を開く

#### 概要

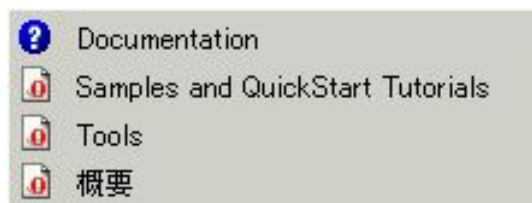
.NET Frameworkの概要を記したHTML文書を開く

#### 執筆者



沖林 正紀

IBMソフトウェア事業部  
データ・マネジメント事業  
業推進技術部のメンバー



画面2-1 .NET Framework SDK ベータ2 日本語版のメニュー

C#のプログラムを開発する場合に、ソースコードの拡張子はcsとします。コンパイルを行うには、「コマンド・プロンプト」を起動し、cscコマンドを実行します。コマンドラインから

```
> csc ソースコードのファイル名称 (例: sample.cs)
```

と入力して実行することで、指定したC#のソースコードをコンパイルできます。コンパイルの結果は、拡張子がexeのファイルになります。つまり実行形式ですから、すぐに実行することができます。たとえば、sample.csというソースコードをコンパイルするとsample.exeというファイルが生成されますので、コマンドラインからsampleと入力すると、コンパイルされたプログラムを実行することができます。

### 3. C#によるDBアプリケーションの開発

それでは、ここからはC#によるDBアプリケーションの作成方法をご紹介します。今回は最初ということで、C#(シー・シャープ)言語によるデータベースの接続と切断、そしてSQLを実行する方法について取り上げます。ここで登場するのは、System.Data.OleDb名前空間(Namespace)に属する、名前がOleDbから始まる各クラスです。データベースにアクセスするためのクラス群は、ADO.NETと呼ばれるクラス群に含まれています。

今回のサンプルコード: [sample01.zip](#)

#### 3-1. データベースの接続と切断

それでは、ここからいよいよC#が登場します。まずはデータベースの接続からはじめます。C#の場合、System.Data.OleDb名前空間に属するOleDbConnectionというクラスのインスタンスがデータベースの接続を表しています。つまりこのクラスのインスタンスを生成することが即ちデータベースを接続することになります。データベースの接続に先立って、接続先を表す文字列が必要になります。OLE DBとしての接続を司るIBM OLE DB Provider for DB2(以下OLE DBプロバイダ)はIBMDADB2という名称で表されますので、文字列全体では、以下のようになります。なお、udbuser, passwordはあらかじめ設定されているデータベースSAMPLEの接続に必要なユーザ名とパスワードです。

```
Provider=IBMDADB2;DSN=(データソース名);Server=(サーバ名);UID=(ユーザ名);PWD=(パスワード)
例) Provider=IBMDADB2;DSN=SAMPLE;Server=localhost;UID=udbuser;PWD=password
```

C#でデータベースの接続を表すOleDbConnectionクラスのインスタンスはこの文字列を引数とするコンストラクタを用いて生成します。なお、C#での文字列はstringオブジェクト(System.Stringクラスのインスタンス)となります。

```
using System.Data.OleDb;
// ..... (省略) .....
string connStr =
    "Provider=IBMDADB2;DSN=SAMPLE;Server=localhost;UID=udbuser;PWD=password";
OleDbConnection conn = new OleDbConnection( connStr );
```

そして、実際にデータベースを接続するには、Open()メソッドを実行します。

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

また、データベースを切断するときはClose()メソッドを実行します。

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

データベースを接続するときにデータソース名が間違っているなど、正常にデータベースに接続できない場合も考えられますので、実際のプログラムでは、その際に発生する例外OleDbExceptionを捕らえて、例外が発生した場合の処理を記述しておいたほうが良いでしょう。また、正常に接続できた後でも、データベースにアクセスしている間に何らかの例外が発生する可能性がありますので、例外処理が終了した後にデータベースを切断できるようにClose()メソッドはfinally節の中で記述しておきます。

```
using System.Data.OleDb;
// ..... (省略) .....
string connStr =
    "Provider=IBMDADB2;DSN=SAMPLE;Server=localhost;UID=udbuser;PWD=password";
OleDbConnection conn = new OleDbConnection( connStr );
try {
    conn.Open();           // データベースを接続する
    //
    // データベースにアクセスする処理
    //
} catch( OleDbException ex ) {
    // 例外が発生した場合の処理(例)
    Console.WriteLine( ex.ToString() ); // エラーメッセージをコマンド・プロンプトに表示
} finally {
    conn.Close();         // データベースを切断する
}
// ..... (省略) .....
```

### 3-2. SQLを実行する

前項(3-1.)でご紹介しているように、データベースの接続と切断は、始めにOLE DBプロバイダ 'IBMDADB2' と接続先を設定したあとはSystem.Data.OleDb.OleDbConnectionクラスのOpen()メソッドとClose()メソッドを使用することで実現することができます。

次はSQLを実行する方法についてご紹介します。C#でSQLを実行するには、照会(SELECT)、照会以外(INSERT, UPDATE, DELETE)、結果が1行のみ(count(\*)関数など)のそれぞれによって3種類のメソッドを使い分けます。

C#でSQLを実行するには、その準備として、まずOleDbCommandオブジェクトを生成しておきます。コンストラクタの引数はSQLを値とする文字列(stringオブジェクト)とOleDbConnectionオブジェクトです。

```
// ..... (省略) .....
OleDbConnection conn = ....; // 前項(3-1.)参照
try {
    conn.Open();
    string sql = "select * from department order by deptno"; // 実行するSQL文(例)
    OleDbCommand command = new OleDbCommand( sql, conn );
    // SQLを実行する (この後ご紹介します)
} catch( OleDbException ex ) {
    // データベースアクセス時の例外処理
} finally {
    conn.Close();
}
// ..... (省略) .....
```

OleDbCommandオブジェクトが生成できたら、先ほどご紹介した3種類のメソッドを使ってSQLを実行します。それらのメソッド名称と戻り値は以下のとおりです。それぞれのメソッドに引数はありません(正確にはExecuteReaderメソッドは引数を持つものもありますが、簡単のためにここでは省略させていただいています)。

表3-1 SQLの処理とOleDbCommandクラスのメソッド名称や戻り値との対応

SQLの処理	メソッド名称	戻り値
照会 (SELECT)	ExecuteReader	OleDbDataReader
照会以外 (INSERT,UPDATE,DELETE)	ExecuteNonQuery	int (System.Int32)
実行結果が一行のみ (count(*)関数など)	ExecuteScalar	Object

#### a) 照会(SELECT)を実行する

まずはSELECTによる照会からです。OleDbCommandオブジェクトのExecuteReaderメソッドを実行するとOleDbDataReaderオブジェクトが戻り値となります。これは照会結果を1行ずつ読み取るためのオブジェクトで、さらにその中の各列のデータを直接取得することができます。

```
// ..... (省略) .....
OleDbConnection conn = ....; // 前項(3-1.)参照
// ..... (省略) .....
string sql = "select * from department order by deptno"; // 実行するSQL文(例)
OleDbCommand command = new OleDbCommand( sql, conn );
OleDbDataReader reader = command.ExecuteReader();
// ..... (省略) .....
```

戻り値であるOleDbDataReaderオブジェクトから照会(SELECT文)の実行結果を取得するには、Read()メソッドでカーソルを次の行に進められるかどうかを確認し、それが可能であれば、カーソルを進めた行の各列のデータを取得します。各行の値を取得するには、主にi) 列名を指定する方法とii) 0から始まる番号を指定する方法とがあります。

#### i) 列名を指定して各列のデータを取得する方法

この方法は戻り値がObjectとなりますので、必要に応じてそれをキャストする必要があります。もしあらかじめ取得するデータの型が決められている場合はb)の方法で取得するようにして処理の手間を増やさないようにすると良いでしょう。

```
// ..... (省略) .....
OleDbDataReader reader = command.ExecuteReader();
while( reader.Read() ) {
    Console.WriteLine( "-----+-----" );
    Console.WriteLine( "DEPTNO = {0}", reader[ "DEPTNO" ] );
    Console.WriteLine( "DEPTNAME = {0}", reader[ "DEPTNAME" ] );
}
// ..... (省略) .....
```

#### ii) 0から始まる番号を指定して各列のデータを取得する方法

この方法は戻り値のデータ型に応じて、データを取得するためのメソッドの名称が異なります。以下は文字列(char, varchar)を取得している例ですが、これ以外にも表3-2のようなメソッドがあります。少々読みにくいかもしれませんが、Console.WriteLineメソッドの引数の文字列にある{0}はどの引数(ここではreader.GetString(0)など)をコマンド・プロンプトに表示するのかを決めているもので、GetString()メソッドの引数とはまったく別のものです。

```
// ..... (省略) .....
OleDbDataReader reader = command.ExecuteReader();
while( reader.Read() ) {
    Console.WriteLine( "-----+-----" );
    Console.WriteLine( "DEPTNO = {0}", reader.GetString( 0 ) );
    Console.WriteLine( "DEPTNAME = {0}", reader.GetString( 1 ) );
}
// ..... (省略) .....
```

表3-2 各列の主なデータ型とOleDbDataReaderクラスのメソッドとの対応

SQLデータ型

OleDbDataReaderクラスのメソッド 戻り値(C#オブジェクト)

SMALLINT	GetInt16	Int16
INTEGER	GetInt32	Int32
BIGINT	GetInt64	Int64
REAL	GetFloat	Single
FLOAT/DOUBLE	GetDouble	Double
DEC (p, s)	GetDecimal	Decimal
BLOB(N)	GetBytes	Byte[]
CHAR(N), VARCHAR(N), LONG VARCHAR, CLOB(N)	GetString	String
GRAPHIC(N), VARGRAPHIC(N), LONG GRAPHIC	GetString	String
DATE	GetDateTime	DateTime
TIME	GetTimeSpan	TimeSpan
TIMESTAMP	GetDateTime	DateTime

データベースから照会されたデータをすべて読み終えて処理を終了する場合は、OleDbDataReaderオブジェクトのClose()メソッドを実行してリソースを開放しておきましょう。

```
// ..... (省略) .....
OleDbDataReader reader = command.ExecuteReader();
// ..... (省略) .....
reader.Close(); // OleDbDataReaderオブジェクトをクローズ
// ..... (省略) .....
```

b) 照会以外 (INSERT, UPDATE, DELETE) を実行する

a) の照会 (SELECT 文) 以外を実行する場合には、ExecuteNonQuery() メソッドを実行します。なぜこのように SELECT とそれ以外 (INSERT, UPDATE, DELETE) でメソッドが分けられているかというと、これらを実行する場合にはトランザクション処理についても考慮に入れなければならないからです。それでは、まずトランザクションの開始と終了の方法からご紹介します。

i) トランザクションの開始

DB2 UDB に接続している場合のトランザクションは OleDbTransaction クラスで表されます。トランザクションを開始するには、以下のようにデータベースの接続を表す OleDbConnection オブジェクトの BeginTransaction() メソッドで OleDbTransaction オブジェクトを取得し、それを OleDbCommand オブジェクトの Transaction プロパティに設定 (代入) します。



```

using System.Data;           // IsolationLevelクラス
using System.Data.OleDb;    // OleDb~クラス
// ..... (省略) .....
string sql = "insert ~ ";   // 実行したいSQL文(SELECT以外)
OleDbConnection conn = ....; // 前項(3-1.)参照
// ..... (省略) .....
OleDbCommand command = new OleDbCommand( sql, conn );
OleDbTransaction tran = conn.BeginTransaction( IsolationLevel.ReadCommitted );
command.Transaction = tran; // トランザクションの開始
// ..... (省略) .....

```

## ii) トランザクション分離レベルの設定

トランザクションを開始する際、OleDbConnectionのBeginTransactionメソッドの引数に(System.Data.)IsolationLevelクラスの定義済みのフィールド(定数)を用いることでトランザクション分離レベルを設定することができます。このうち、DB2 UDBがサポートするトランザクション分離レベル(詳細は[Web de DB2 第8回 5.参考](#)を参照)に対応するものは以下のとおりです。

System.Data.IsolationLevelクラスの定数 DB2 UDBのトランザクション分離レベル	
Serializable	反復可能読み取り (RR:Repeatable Read)
RepeatableRead	読み取り固定 (RS:Read Stability)
ReadCommitted	カーソル固定 (CS:Cursor Stability)
ReadUnCommitted	非コミット読み取り (UR:UnCommitted Read)

## iii) トランザクションの終了(コミットやロールバック)

SQLを実行した後、その処理が正常に完了したらトランザクションをコミット(Commit)し、何らかの例外が起きた場合はロールバック(Rollback)しなくてはなりません。これらはOleDbTransactionオブジェクトのCommit()メソッドやRollback()メソッドで行います。特に、処理中に例外が発生した場合にロールバックするためには、try ~ catchを用いて以下のように記述するのが良いでしょう。

```

// ..... (省略) .....
OleDbConnection conn = ....; // 前項(3-1.)参照
try {
    conn.Open();
    string sql = "insert ~"; // 実行するSQL文(例)
    OleDbCommand command = new OleDbCommand( sql, conn );
    OleDbTransaction tran = conn.BeginTransaction( IsolationLevel.ReadCommitted );
    command.Transaction = tran; // トランザクションの開始
    // SQLを実行する (この後ご紹介します)
    tran.Commit(); // 処理が正常に完了したらコミット
} catch( OleDbException ex ) {
    tran.Rollback(); // 処理中に何らかの例外が発生したらロールバック
    // データベースアクセス時の例外処理
} finally {
    conn.Close();
}
// ..... (省略) .....

```

#### iv) SQLの実行

トランザクションの開始と終了の方法に続いてSQLを実行する方法をご紹介します。前出のとおりOleDbCommandクラスのExecuteNonQuery()メソッドによりSQLを実行します。このメソッドの戻り値は実際に操作したデータの件数です。つまり、DB2のコマンド行プロセッサなどでSQLを実行した場合に「5件更新されました」という結果が表示されるときに5が戻り値となります。ですので、トランザクションの開始・SQLの実行・トランザクションの終了の一連の流れは以下のようになります。

```
// ..... (省略) .....
OleDbConnection conn = ....; // 前項(3-1.)参照
try {
    conn.Open();
    string sql = "insert ~"; // 実行するSQL文(例)
    OleDbCommand command = new OleDbCommand( sql, conn );
    OleDbTransaction tran = conn.BeginTransaction( IsolationLevel.ReadCommitted );
    command.Transaction = tran; // トランザクションの開始

    int row = command.ExecuteNonQuery(); // SQLを実行する
    Console.WriteLine( "{0} 件のデータが操作されました", row ); // 操作された件数

    tran.Commit(); // 処理が正常に完了したらコミット
} catch( OleDbException ex ) {
    tran.Rollback(); // 処理中に何らかの例外が発生したらロールバック
    // データベースアクセス時の例外処理
    Console.WriteLine( "処理中に例外が発生しました" );
    Console.WriteLine( ex.ToString() );
} finally {
    conn.Close();
}
// ..... (省略) .....
```

#### c) 実行結果が一行のみの場合

count(\*)関数などのように、SQLを実行した結果が1行のみになる場合には、ExecuteScalar()メソッドを実行します。このメソッドは実行結果の先頭行のみを取得することができるメソッドです。仮に先頭以外の行が存在しているとしてもそれらは無視されます。

実行の方法はメソッドの名称が変わる以外はa)の場合と同じなのですが、実行結果をOleDbDataReaderからではなく、直接Objectとして取得するため、これを実際には処理の必要に応じた適切な型にキャストしてから使用することになります。なお、C#におけるSQLデータ型とOLE DBデータ型との対応は表3-2をご参照ください。

```
// ..... (省略) .....
OleDbConnection conn = ....; // 前項(3-1.)参照
// ..... (省略) .....
string sql = "select count(*) from department order by deptno"; // 実行するSQL文(例)
OleDbCommand command = new OleDbCommand( sql, conn );
int count = (int)command.ExecuteScalar(); // SQLの実行結果を1行だけ取り出す
Console.WriteLine( "結果 = {0}", count ); // 行数をコマンド・プロンプトに表示
// ..... (省略) .....
```

## 4.おわりに

.NETやC#の概要とOleDb~クラスによるデータベースの接続と切断、そしてSQLの実行と駆け足で眺めてみました

いかがでしたでしょうか。もしかしたら「クラス」「オブジェクト」という概念に、すぐにはなじめない部分もあるかもしれませんが、これから少しずつプログラムを作成していきながら習得していきましょう。どうぞよろしくお願いいたします。

