



第3回 C#にふれる(3) - オブジェクトからのXML文書の生成とXML文書からのデータの抽出

沖林 正紀

1. はじめに

前回はデータベースのインスタンスのような役割を果たすDataSetクラスとそれに関連するクラス群をご紹介しましたが、今回はC#であらかじめ用意されているXMLに関連したクラスやメソッドを用いてXML文書を生成したり解析したりする方法をご紹介します。

XML文書はデータフォーマットのひとつとしてさまざまな分野に広まっており、リレーショナル・データベースのデータとXMLのフォーマットとの仲介役がアプリケーションに求められるケースがあります。また、Webサービス(C#ではXML Webサービス)においてもSOAP, WSDL, UDDIなどでXML文書が用いられているように、XMLは単に「文書」としてだけでなく、通信プロトコルとしての役割も担うようになってきています。

ここでは特にデータベースに関連した処理として、データベースのデータからXML文書を生成したり、逆にXML文書を解析して抽出したデータをデータベースに挿入する方法についてご紹介いたします。前回ご紹介したDataSetオブジェクトも再登場します。C#の開発環境は引き続き.NET Framework SDK ベータ2 日本語版を前提としています。

⇒ 今回のサンプルコード: [sample03.zip](#)

2. XMLに関連したクラス

まず、C#においてXMLに関連するクラスについて簡単にご紹介しましょう。

C#のネームスペース(Namespace)のうち、XMLに直接関連するのは以下のものです。

a) <code>System.Xml</code>	XML文書の生成と解析、書き込みと読み込み
b) <code>System.Xml.Xsl</code>	XSLT(XSL Transformations)によるXML文書の変換
c) <code>System.Xml.Schema</code>	XML Schema文書に関連するクラス
d) <code>System.Xml.Serialization</code>	オブジェクトとXML文書とを相互に変換
e) <code>System.Xml.XPath</code>	XPath式により指定されたノードの特定とノード間の移動

今回登場するクラスは、このうちのa),b)のネームスペースに属するものです。

a) <code>System.Xml</code> ネームスペースのクラス(抜粋)	
<code>XmlDocument</code>	XML文書そのもの
<code>XmlDataDocument</code>	DataSetオブジェクトとXMLDocumentクラスとの橋渡し
<code>XmlElement</code>	XMLの要素(~)
<code>XmlDeclaration</code>	XML宣言 (例: <code><?xml version="1.0" encoding="UTF-8"?></code>)
<code>XmlProcessingInstruction</code>	XML処理命令 (例: <code><?xml-stylesheet href="template.xsl" type="text/xsl"?></code>)
<code>XmlNodeList</code>	XMLのノードの集合とノード間の移動
<code>XmlTextWriter</code>	XML文書の書き込み(出力)
<code>XmlTextReader</code>	XML文書の読み込み(入力)
b) <code>System.Xml.Xsl</code> ネームスペースのクラス(抜粋)	
<code>XsltTransform</code>	XSLT文書の内容によりXML文書を別の文書に変換

そして前回登場したDataSetクラスにはXML文書とXMLスキーマ定義(XML Schema Definition)に関するメソッドが用意されています。

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](#)
△

コンテンツ

- はじめに
 - XMLに関連したクラス
 - XML文書の構造
 - XML文書の生成
 - DataSetオブジェクトからのXML文書の生成
 - XmlDataDocumentオブジェクトからのXML文書の生成
 - XmlDocumentオブジェクトによるXML文書の生成
 - XML文書からのデータの抽出
 - XML文書からのDataSetオブジェクトの生成
 - XML文書からのXmlDocumentオブジェクトの生成とデータの抽出
 - 2つの表の関係とXML文書の構造
 - 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](#)



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
推進技術部のメンバー

```
• DataSetクラスが持つXML文書に関するメソッド
DataSet ds = new DataSet( "instance" );
// ..... (省略)DataSetオブジェクトにデータベースのデータを展開する(前回の3-2.参照)

// ----- XML文書に関するメソッド(使用例) -----
// DataSetオブジェクトが持つデータをXML文書の文字列として生成する
string xml = ds.GetXml( );
// 上記のXML文書をdocument.xmlというファイルに出力する
ds.WriteXml( "document.xml" );
// XML文書document.xmlを読み込み、内容を解析してDataSetオブジェクトを生成する
ds.ReadXml( "document.xml" );

// ----- XMLスキーマ定義に関するメソッド(使用例) -----
// GetXml()メソッドで生成するXML文書のXMLスキーマ定義を生成する
string xsd = ds.GetXmlSchema( );
// 上記のXMLスキーマ定義をschema.xsdというファイルに出力する
ds.WriteXmlSchema( "schema.xsd" );
// XML文書schema.xsdをXMLスキーマ定義として読み込む
ds.ReadXmlSchema( "schema.xsd" );
```

3. XML文書の構造

それでは、XML文書の生成や解析の処理方法をご紹介します前に、XML文書の構造について確認しておきましょう。

XML文書には、その先頭に、以下のようなXML宣言というものがが必要です。

```
<?xml version="1.0" encoding="UTF-8"?>
```

それぞれの属性の値は、以下のようなものを指定します。詳細はXMLの仕様をご参照ください。

- i) version XML仕様のバージョン番号(2002年2月現在は1.0のみ)
- ii) encoding 文字コードのエンコード形式(デフォルトはUTF-8、シフトJISはShift-JIS)

そして、XML宣言の直後にはルートタグと呼ばれる最初の開始タグ(形式:<tag>)を記述し、最後には同じ名前の終了タグ(形式:</tag>)を記述します。

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
:
<!-- XML文書の内容 -->
:
</root>
```

XMLのタグの形式には、開始タグ、終了タグ、単独タグ(形式:<tag />)があります。開始タグを記述した場合には、同じタグ名称の終了タグが必要(<tag> ~ </tag>)となります。開始タグと終了タグとの間にはテキストが記述されたり、別のタグが記述されたりします。単独タグは開始タグと終了タグの性質を併せ持ったものといえます。そして開始タグと単独タグには複数の属性を記述することが可能です。

- 開始タグの終了タグを対で記述する場合(例)
<product family="DB2">DB2 ユニバーサル・データベース</product>
- 単独タグを記述する場合(例)
<lineup name="Informix" />

これらのタグは、XMLスキーマ定義で制限している場合を除いて、同じ名称のものを複数回使用することもできます。また、タグとテキストとを入れ子構造にすることも可能です。

・同じ名称のタグを複数回記述し、タグとテキストを入れ子構造とする場合(例)

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <product family="DB2">DB2 ユニバーサル・データベース</product>
  <product family="BI">DB2 OLAP Server</product>
  <product family="CM">Content Manager</product>
  <product family="Informixデータベース・サーバー">IBM Red Brick Warehouse</product>
</root>
```

XMLにおけるタグの名称は、XML仕様に定められた文字を使用する限りは自由に決めることができますが、事前にXML文書の構造やタグ名称がXMLスキーマ定義によって決められている場合は、それに従っていないタグを使用することはできません。このようにXMLスキーマ定義に沿った構造とタグが用いられているXML文書を受当なXML (valid XML)もしくは検証済みXMLといい、XMLスキーマ定義が存在せず、タグやテキストの入れ子構造が整っているだけのXML文書を整形XML(well-formed XML)といいます。

4.XML文書の生成

C#におけるXML文書に関連するクラスとXML文書の構造について確認したところで、XML文書の生成する方法をご紹介します。ここではデータベースのデータを照会し、その結果を取得してXML文書を生成するという一連の流れを取り上げます。

4-1. DataSetオブジェクトからのXML文書の生成

DataSetクラスは前回ご紹介したように、データベースではインスタンスに相当するといえるクラスで、内部には表に相当するDataTableオブジェクトの集合や関係の定義を表すDataRelationオブジェクトの集合を持っています。このクラスには、2.でもご紹介しているように、XML文書に関連するメソッドも持っており、DataSetオブジェクトが持つデータからXML文書を生成したり、逆にXML文書の内容からDataSetオブジェクトを生成することもできます。

早速ですが、どんなXML文書が生成できるのかを確かめてみましょう。

前回ご紹介したようにDataSetオブジェクトにDB2 UDBのサンプルデータベースSAMPLEのDEPARTMENT表の内容を展開し、その状態を表したXML文書を生成してみます。

```
using System;          // Console
using System.Data;     // DataSet
// ..... (省略) .....
DataSet ds = new DataSet( "table" );
// ..... (省略)DataSetオブジェクトにデータベースのデータを展開する(前回の3-2.参照)
string xml = ds.GetXml( );
Console.WriteLine( xml );
// ..... (省略) .....
```

DEPARTMENT表のデータをXML文書に展開した結果は以下のようになります。この表のデータの内容はSQL概説で確認していただくことができます。なお、LOCATION列のデータはNULLであるため、タグが生成されていません。

```
<table>
  <department>
    <DEPTNO>A00</DEPTNO>
    <DEPTNAME>SPIFFY COMPUTER SERVICE DIV.</DEPTNAME>
    <MGRNO>000010</MGRNO>
    <ADMRDEPT>A00</ADMRDEPT>
  </department>

  <!-- (省略) -->

  <department>
    <DEPTNO>E21</DEPTNO>
    <DEPTNAME>SOFTWARE SUPPORT</DEPTNAME>
    <MGRNO>000100</MGRNO>
    <ADMRDEPT>E01</ADMRDEPT>
  </department>
</table>
```

このXML文書のタグやテキストの入れ子構造は外側から順に次のようになっています。

- i) DataSetオブジェクトと同じ名称のタグ(<table> ~ </table>)
- ii) 表と同じ名称のタグ(<department> ~ </department>)
- iii) 各列名のタグ(<DEPTNO> ~ </DEPTNO>など)
- iv) 各列のデータ(DEPTNO列であれば'A00' ~ 'E21')

i)はここでは<table> ~ </table>となっていますが、これはDataSetクラスのコンストラクタやDataSetNameプロパティで定義した名称がそのままタグの名称にもなります。もしこれらで名称を定義しなかった場合には<NewDataSet> ~ </NewDataSet>となります。

また、このXML文書には3.で紹介しているXML宣言(<?xml version=~ ?>)がありませんが、これはこのXML文書を更に加工して別のタグを追加する場合に備えてのことです。XML宣言も加わったXML文書としたい場合はWriteXml()メソッドを用いて、このXML文書をファイルなどに出力します。このメソッドの引数にファイル名を指定するとエンコードはUTF-8となります。シフトJISなど別の文字コードでエンコードしたい場合には、メソッドの引数にエンコード方法を指定したXmlTextWriterオブジェクトを指定してください。以下はその例です

```
using System.Data; // DataSet
using System.Text; // Encoding
using System.Xml; // XmlTextWriter
// ..... (省略) .....
DataSet ds = new DataSet( );
// ..... (省略)DataSetオブジェクトにデータベースのデータを展開する(前回の3-2.参照)
// XML文書をdepartment.xmlというファイルに出力する(引数はファイル名、エンコード)
XmlTextWriter xmlOut
    = new XmlTextWriter( "department.xml", Encoding.Default );
xmlOut.Formatting = Formatting.Indented; // XML文書の入れ子構造をインデントする
xmlOut.WriteStartDocument(); // XML文書最初の部分をファイルに書き込む
ds.WriteXml( xmlOut ); // XML文書の本体をファイルに書き込む
xmlOut.Flush();
xmlOut.Close();
// ..... (省略) .....
```

ここではDataSetクラスがもつメソッドを用いてXML文書の生成を行いました。この方法では、DataSetオブジェクトにあらかじめ展開されているデータをXML文書として出力するのみでした。これはとても単純な方法で、生成されるXML文書は既に完成しているものとして扱うか、その前後にタグを追加することはできても、XML文書の途中でタグやテキストを追加することはできません。それを行うには次項で紹介するXmlDataDocumentクラスを用いるのが良いでしょう。そうすれば、XMLのタグやテキストを追加することができるようになります。

4-2. XmlDataDocumentオブジェクトからのXML文書の生成

前節4-1.で用いたDataSetクラスのメソッドによるXML文書の生成では、単純にDataSetオブジェクトが持つデータの内容をXML文書として出力するのみでした。ですからXML文書の途中でデータを追加したい場合には対応できませんでした。ここではそれを可能にする方法として、XmlDataDocumentオブジェクトを用いてXML文書を生成する方法をご紹介します。

XmlDataDocumentクラスはXMLのタグやテキストをオブジェクトとして扱うDOM(Document Object Model)に対応したXmlDocumentクラスのサブクラスで、DataSetオブジェクトがもつデータをその関係も含めてDOMオブジェクトとして扱うためのクラスです。XML文書の途中でタグやテキストを追加する場合は、それらをオブジェクトとして生成し、XmlDataDocumentオブジェクトのメソッドにより追加することになります。

まず、XML宣言を追加した後にXML文書をファイルに出力する方法を以下に示します。この例では出力先にファイル名を直接指定していますので、UTF-8によるエンコードとなります。その他のエンコードで出力したい場合には、前節4-1.で登場したXmlTextWriterクラスなどエンコードを指定できるクラスを用いて出力してください。

```
using System.Data; // DataSet
using System.Xml; // XmlDataDocument, XmlDeclaration
// ..... (省略) .....
DataSet ds = new DataSet( );
// ..... (省略)DataSetオブジェクトにデータベースのデータを展開する(前回の3-2.参照)
ds.EnforceConstraints = false; // データ構造の変更を許可する(制約をはずす)
XmlDataDocument xmlDoc = new XmlDataDocument( ds );
// XML宣言(<?xml version=" ~ ?>)の生成(エンコードをシフトJISとする場合)
XmlDeclaration dec = xmlDoc.CreateXmlDeclaration( "1.0", "Shift-JIS", null );
xmlDoc.PrependChild( dec ); // XML文書の先頭にXML宣言を追加
xmlDoc.Save( "dataset.xml" ); // XML文書をファイルに出力する(文字コードUTF-8)
// ..... (省略) .....
```

上記の例ではXML宣言をXmlDeclarationオブジェクトとして生成し、それをXmlDataDocumentクラスのPrependChild()メソッドによりXML文書の先頭に追加するという処理を行っています。これ以外にもタグやテキストのオブジェクトが追加できます。追加できるオブジェクトのうち主なものを以下に示します。

```
using System.Data;          // DataSet
using System.Xml;          // XmlProcessingInstruction, XmlElement, XmlComment, XmlText etc.
// ..... (省略) .....
DataSet ds = new DataSet( );
// ..... (省略)DataSetオブジェクトにデータベースのデータを展開する(前回の3-2.参照)
ds.EnforceConstraints = false;          // データ構造の変更を許可する
XmlDataDocument xmlDoc = new XmlDataDocument( ds );
// (XSL文書により画面表示用に変換する場合)xml-stYLESHEET命令
XmlProcessingInstruction pi = xmlDoc.CreateProcessingInstruction(
    "xml-stYLESHEET", "href=¥"template.xsl¥" type=¥"text/xsl¥" );
// XMLのタグ (例: ~ )
XmlElement tag = xmlDoc.CreateElement( "tag" );
// XMLのコメント (例: <!-- コメント --> )
XmlComment comment = xmlDoc.CreateComment( "コメント" );
// XMLのテキスト (文字列)
XmlText text = xmlDoc.CreateTextNode( "文字列" );
// ..... (省略) .....
```

これらのオブジェクトを追加するメソッドのうち、主なものを以下に示します。

XML処理命令はXML宣言のすぐあとに追加する必要がありますので、InsertAfter()メソッドにより、位置を指定した上で追加しています。そしてタグとテキストの両方を追加する場合は、先にAppendChild()メソッドでタグにテキストを追加しておき、そのあとそれをXML文書に追加してください。

```
using System.Xml;          // XmlNodeList
// ..... (省略) .....
// XML処理命令の追加 (XML宣言の次に追加)
xmlDoc.InsertAfter( pi, dec );
// タグにテキストを追加 (例: 文字列)
tag.AppendChild( text );
XmlNodeList list = xmlDoc.GetElementsByTagName( "DEPTNO" );
// <DEPTNO>タグの前に<tag>文字列</tag>を追加する場合
xmlDoc.InsertBefore( tag, list[0] );
// <DEPTNO>タグの後に<tag>文字列</tag>を追加する場合
xmlDoc.InsertAfter( tag, list[0] );
// ..... (省略) .....
```

4-3. XmlDocumentオブジェクトによるXML文書の生成

前節4-2.ではDataSetオブジェクトのデータを利用してXML文書を作成するためにXmlDataDocumentオブジェクトを用いましたが、今度はそれを行わずに、SQLにより照会したデータを取得して、そのデータから、はじめは空のXML文書に直接タグを追加していき、XML文書を生成してみます。生成するのは4-1.に登場したXML文書と同じもので、使用するクラスは前節4-2.に登場したXmlDataDocumentクラスのスーパークラスであるXmlDocumentクラスです。

まず、空のXML文書(XmlDocumentオブジェクト)に対して、前節4-2.と同じようにPrependChild()メソッドによりXML宣言を最初に追加しておきます。XSL文書による変換処理が必要な場合はXML処理命令xml-stYLESHEETも追加しておきます。

```

using System.Xml; // XmlDocument, XmlDeclaration, XmlProcessingInstruction, XmlElement
// ..... (省略) .....
XmlDocument xmlDoc = new XmlDocument( );
// XML宣言(<?xml version=" ~ ?>)の生成(エンコードをシフトJISとする場合)
XmlDeclaration dec = xmlDoc.CreateXmlDeclaration( "1.0", "Shift-JIS", null );
xmlDoc.PrependChild( dec ); // XML文書の先頭にXML宣言を追加
// (XSL文書により画面表示用に変換する場合)xml-stYLESHEET命令
XmlProcessingInstruction pi
    = xmlDoc.CreateProcessingInstruction(
        "xml-stYLESHEET", "href=¥"template.xsl¥" type=¥"text/xsl¥" );
xmlDoc.AppendChild( pi ); // XML文書の先頭にXML宣言を追加
// ..... (省略) .....

```

そして、ルートタグを定義して、データベースのデータをXML文書に追加するための準備を整えます。ここまではXML文書を作成するために最低限しておかなくてはならないことです。このタグはすぐにはXmlDocumentオブジェクトには追加せず、これに他のタグを入れ子にして追加していきます。

```
XmlElement root = xmlDoc.CreateElement( "table" );
```

次に入れ子にするデータを定義していくのですが、これはDEPARTMENT表の各列のデータです。第1回でご紹介したように、SQLを実行してデータベースを照会し、その結果をXMLのタグとテキストとして定義していきます。なおこの方法では、データがNULLであるLOCATION列のタグ(<LOCATION></LOCATION>)も生成されます。

```

// ..... (省略) .....
OleDbConnection conn = .....; // データベースを接続する(省略: 第1回の3-2.を参照)
try {
    OleDbCommand command = new OleDbCommand( "select * from department" , conn );
    OleDbDataReader reader = command.ExecuteReader( ); // SQLを実行する
    while( reader.Read() ) {
        XmlElement row = xmlDoc.CreateElement( "department" ); // タグを生成
        for( int i = 0; i < reader.FieldCount; i++ ) {
            // 照会したデータをXMLのテキストとする
            XmlText text = xmlDoc.CreateTextNode( reader.GetValue( i ).ToString() );
            // 列名と同じ名称のタグを生成する
            XmlElement tag = xmlDoc.CreateElement( reader.GetName( i ) );
            // テキストを列名と同じ名称のタグの間に追加する(例: A00)
            tag.AppendChild( text );
            // ~ タグの間に各列のデータを追加する
            row.AppendChild( tag );
        }
        // ~ タグ全体のデータ(1行分)をルートタグに追加する
        root.AppendChild( row );
    }
    reader.Close(); // OleDbDataReaderのクローズ
    // ルートタグ以下のデータ(全データ)をXML文書に追加する
    xmlDoc.AppendChild( root );
    xmlDoc.Save( Console.Out ); // コマンド・プロンプト画面にXML文書を表示する
} catch( Exception ex ) {
    // 例外が発生した場合の処理
} finally {
    conn.Close(); // データベースのクローズ
}
// ..... (省略) .....

```

上記の処理では、タグを追加する順番が最も内側にあたる各列のデータから始まっています。そして1行分のデータ(<department> ~ </department>)、最後にすべてのデータをルートタグに追加しています。

5.XML文書からのデータの抽出

4.ではXML文書を作成する方法をご紹介してきました。この場合はXmlElement, XmlElement, XmlTextなどのオブジェクトを生成し、どんどん追加していくことで最終的にXML文書を作成することができました。

今度は既に生成されたXML文書から必要なデータを抽出して、それをデータベースに追加するまでの一連の流れをご紹介してい

きます。C#では4.で登場したクラスにXML文書からデータを抽出するためのメソッドもあらかじめ用意されています。そのため、XML文書からデータを抽出する場合は、まずXML文書を読み込んでオブジェクトを生成し、そのオブジェクトが持つメソッドを利用してデータを抽出することになります。以下、DataSet, XmlDocument, XmlDataDocumentの各クラスにおける方法をご紹介します。いきたいと思います。

5-1. XML文書からのDataSetオブジェクトの生成

DataSetクラスでは、XML文書の内容を読み込んで内部にもつデータを生成することができます。もっともシンプルな方法を以下に示します。XML文書はUTF-8でエンコードされているものとします。また、XML文書の構造は4-1.のXML文書と同じとします。

```
using System.DataSet; // DataSet
// ..... (省略) .....
DataSet ds = new DataSet();
ds.ReadXml( "sample.xml" );
```

XML文書がシフトJISでエンコードされていることがわかっている場合は、以下のようにXmlTextReaderクラスを使用して読み込みます。

```
using System.Data; // DataSet
using System.IO; // XmlTextReader, StreamReader, FileStream
using System.Text; // Encoding
using System.Xml; // XmlTextReader
// ..... (省略) .....
DataSet ds = new DataSet();
XmlTextReader xmlIn = new XmlTextReader( new StreamReader(
    new FileStream( "sample.xml" ), Encoding.GetEncoding( "Shift-JIS" ) ) );
ds.ReadXml( xmlIn );
xmlIn.Close();
// ..... (省略) .....
```

このように、XML文書の構造が4-1.に登場したXML文書のようにになっている場合には、ReadXml()メソッドにより読み込むだけでDataSetオブジェクトが生成されます。そのあとは前回ご紹介した方法で、その内部にもつデータを取り扱うことが可能になります。

5-2. XML文書からのXmlDocumentオブジェクトの生成とデータの抽出

XmlDocumentクラスにも、XML文書を読み込むためのメソッドが用意されています。。こちらをもっともシンプルな方法を以下に示します。この場合もXML文書はUTF-8でエンコードされているものとします。

```
using System.Xml; // XmlDocument
// ..... (省略) .....
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.Load( "sample.xml" );
```

このクラスの場合は、以下のようにInnerXmlプロパティにXMLそのものの文字列を代入して読み込むことも可能です。

```
XmlDocument xmlDoc = new XmlDocument();
string xml = "<?xml version='1.0'?">";
xmlDoc.InnerXml = xml;
```

さて、このようにしてXML文書を読み込んでXmlDocumentオブジェクトを生成することはできたのですが、アプリケーションとしては、そのあとの段階の「どのようにして必要なデータを取得するか」の方が重要です。次にその方法をご紹介します。XmlDocumentオブジェクトが既に生成できているとすれば、あとはどの位置にあるデータが必要かということ指定すればデータを取得することができます。ただし、3.でもご紹介したように、同じ名称のタグを複数回指定することも可能なのですから、単にタグの名称を指定しただけでは、データの位置を特定できません。こういった場合にはXmlNodeListオブジェクトによりデータの位置を特定します。このオブジェクトはタグの位置情報の集合を持ちますので、あとは何番目のタグかを指定すれば、位置を特定できることになります。たとえば、<department>タグの位置情報を取得するためにはXmlDocumentクラスのGetElementsByTagName()メソッドを用います。

以下は、4-1.で登場しているXML文書の内容を元にDEPARTMENT表にデータを追加するためのSQL文(INSERT)を生成する処理を表していますが、この中では各種のプロパティを用いてデータの位置を特定する方法が取り入れられています。

```
using System;           // String(string)
using System.Xml;       // XmlNodeList, XmlNode
// ..... (省略) .....
XmlNodeList tags = xmlDoc.GetElementsByTagName( "department" );
foreach( XmlNode tag in tags ) {
    string comma = "", columns = "", values = "";
    // タグ以下のタグ(各列の列名と同じ名称)の位置情報を取得する
    XmlNodeList columnTags = tag.ChildNodes;
    foreach( XmlNode columnTag in columnTags ) {
        columns += comma;    values += comma;
        // 各列の列名と同じ名称のタグに挟まれたテキスト(例: A00)
        // を取得して、INSERT文の一部とする
        columns += columnTag.Name;
        values += " '" + columnTag.FirstChild.Value + "'";    // ここで取り扱うデータは文字列のみ
        comma = ", ";
    }
    // SQL文を生成する
    string sql = "insert into department ( " + columns + " ) values ( " + values + " )";
    // SQLを実行する(第1回の3-2.参照)
// ..... (省略) .....
```

XmlNodeListクラスのChildNodesプロパティは<department>タグに対する<DEPTNO>、<DEPTNAME>などのタグのように、更に内側にあるタグのある場所をあらわしています。これを子ノードと呼んでいます。ちなみに<DEPTNO>A00</DEPTNO>となっている場合、'A00'という文字列の場所は<DEPTNO>からみた子ノードとなります。このような文字列の子ノードを表す場合には、ChildNodesのうち一番最初の子ノードを表すFirstChildプロパティを用いるのが良いでしょう。

6. 2つの表の関係とXML文書の構造

ここまで、主に1つの表からXML文書を生成する方法やそれを解析する方法についてご紹介してきましたが、ある表が別の表との関係を定義されている場合にはどのようなXML文書が生成されるのでしょうか。

前回、DataRelationクラスで2つの表の関係を定義する方法をご紹介しましたが、このクラスはXML文書の生成でも果たす役割があります。それはある表のデータに別の表のデータを割り込ませることです。これはどのようなことを表すのかをご紹介します。

前回と同様にサンプルデータベースSAMPLEのDEPARTMENT表とEMPLOYEE表のデータをDataSetオブジェクトに展開し、DataRelationオブジェクトでEMPLOYEE表のWORKDEPT列(所属部署番号)とDEPARTMENT表のDEPTNO列(部署番号)との関係を定義しておきます。

```
// ..... (省略) ..... 以下は前回と同じ
using System.Data;
using System.Data.OleDb;
// ..... (省略) .....
OleDbConnection conn = .....; // (省略) 第1回の3-2.を参照

string sql_d = "select * from department"; // DEPARTMENT表のデータを照会するSQL文
string sql_e = "select * from employee";   // EMPLOYEE表のデータを照会するSQL文
OleDbDataAdapter adapter_d = new OleDbDataAdapter( sql_d, conn );
OleDbDataAdapter adapter_e = new OleDbDataAdapter( sql_e, conn );

DataSet ds = new DataSet();
adapter_d.Fill( ds, "department" ); // DEPARTMENT表のデータをDataSetオブジェクトに展開
adapter_e.Fill( ds, "employee" );   // EMPLOYEE表のデータをDataSetオブジェクトに展開

DataTable department = ds.Tables[ "department" ];
DataTable employee = ds.Tables[ "employee" ];
ds.Relations.Add( new DataRelation( "manager",
                                   department.Columns[ "deptno" ], // 親表
                                   employee.Columns[ "workdept" ] ) ); // 従属表
// ..... (省略) .....
```

この状態のDataSetオブジェクトからGetXml()メソッドでXML文書を生成すると、以下のように、単にDEPARTMENT表のデータとEMPLOYEE表のデータが並ぶだけとなります。

```
<NewDataSet>
  <department>
    <DEPTNO>A00</DEPTNO>
    <DEPTNAME>SPIFFY COMPUTER SERVICE DIV.</DEPTNAME>
    <MGRNO>000010</MGRNO>
    <ADMRDEPT>A00</ADMRDEPT>
  </department>
  <!-- (省略) -->
  <employee>
    <EMPNO>000010</EMPNO>
    <FIRSTNAME>CHRISTINE</FIRSTNAME>
    <MIDINIT>I</MIDINIT>
    <LASTNAME>HAAS</LASTNAME>
    <WORKDEPT>A00</WORKDEPT>
    <PHONENO>3978</PHONENO>
    <HIREDATE>1965-01-01T00:00:00.0000000+09:00</HIREDATE>
    <JOB>PRES    </JOB>
    <EDLEVEL>18</EDLEVEL>
    <SEX>F</SEX>
    <BIRTHDATE>1933-08-24T00:00:00.0000000+09:00</BIRTHDATE>
    <SALARY>52750</SALARY>
    <BONUS>1000</BONUS>
    <COMM>4220</COMM>
  </employee>
  <!-- (省略) -->
</NewDataSet>
```

ここでDataRelationオブジェクトのNestedプロパティをtrueとしてから、同様にXml文書を生成しますと、以下のようにDEPARTMENT表のデータにEMPLOYEE表のデータが割り込んだ形になります。

```
// 関係を入れ子構造で表す場合はNestedプロパティをtrueに設定する
ds.Relations[ "manager" ].Nested = true;
```

```
<NewDataSet>
  <department>
    <DEPTNO>A00</DEPTNO>
    <DEPTNAME>SPIFFY COMPUTER SERVICE DIV.</DEPTNAME>
    <MGRNO>000010</MGRNO>
    <ADMRDEPT>A00</ADMRDEPT>
    <employee>
      <EMPNO>000010</EMPNO>
      <FIRSTNAME>CHRISTINE</FIRSTNAME>
      <MIDINIT>I</MIDINIT>
      <LASTNAME>HAAS</LASTNAME>
      <WORKDEPT>A00</WORKDEPT>
      <PHONENO>3978</PHONENO>
      <HIREDATE>1965-01-01T00:00:00.0000000+09:00</HIREDATE>
      <JOB>PRES    </JOB>
      <EDLEVEL>18</EDLEVEL>
      <SEX>F</SEX>
      <BIRTHDATE>1933-08-24T00:00:00.0000000+09:00</BIRTHDATE>
      <SALARY>52750</SALARY>
      <BONUS>1000</BONUS>
      <COMM>4220</COMM>
    </employee>
  </employee>
  <!-- (省略) -->
</employee>
```

```
<!-- (省略) -->
</department>
<!-- (省略) -->
</NewDataSet>
```

上記のXML文書の中で緑色で示した<HIREDATE> ~ </HIREDATE>の間と<BIRTHDATE> ~ </BIRTHDATE>の間にある文字列は日付を表しています。データベースではそれぞれの列はDATE型になっていますが、XMLスキーマ定義のデータ型(仕様はXML Schema Part 2: Datatypesの「3.2.7 datetime」をご参照ください)として記述する場合にはこのようになります。これは'T'を境に左側が日付、右側が時刻とUTC(協定世界時)からの時差を表しています。

たとえば'1965-01-01T00:00:00.0000000+09:00'であれば、'1965-01-01'が日付、'00:00:00.0000000'が時刻、'+09:00'がUTCからの時差となります。このデータは以下のようにいくつかの書式に変換することができます。

```
using System;
using System.Xml;
// ..... (省略) .....
string hiredate = "1965-01-01T00:00:00.0000000+09:00";
DateTime dt = XmlConvert.ToDateTime( hiredate );
Console.WriteLine( dt.ToShortDateString( ) ); // 実行結果は'1965/01/01'
Console.WriteLine( dt.ToLongDateString( ) ); // 実行結果は'1965年1月1日'
Console.WriteLine( dt.ToString( "yyyy-MM-dd" ) ); // 実行結果は'1965-01-01'
// ..... (省略) .....
```

7. おわりに

これまで3回にわたって、駆け足ではありましたがC#にふれてみました。次回からは別の形態のアプリケーションを作成していきます。まずはGUIアプリケーションからです。これまでに作成した処理に加えて、アプリケーションのウィンドウを作成したり、ボタン、ラベル、テキストボックスなどを配置する処理を新たに追加します。DataSetオブジェクトのデータをそっくり表示するためのコンポーネントもありますので、それを使ってみたいと思います。

[↑ 上に戻る](#)