Native SDK for .NET の使用法

Version 2024.1
2024-06-03

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

Native SDK for .NET の使用法
InterSystems IRIS Data Platform　Version 2024.1　2024-06-03

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

図一覧

1

Native SDK for .NET の概要

このドキュメントで取り上げる内容の詳細なリストは、"目次" を参照してください。

InterSystems Native SDK for .NET は、以前は ObjectScript でのみ使用可能であった、以下の強力な InterSystems IRIS® リソースへの軽量インタフェースです。

- ObjectScript クラス・メソッドおよび関数の呼び出し。任意の目的でカスタム ObjectScript クラス・メソッドまたは関数を記述し、ネイティブの .NET メソッドを呼び出すのと同じように、簡単に .NET アプリケーションから呼び出すことができます。
- ゲートウェイ・プロキシ・オブジェクトを介した ObjectScript クラス・インスタンスの操作。ObjectScript アプリケーションで行うのと同じように、簡単にインスタンス・メソッドを呼び出して、プロパティ値を取得または設定できます。
- グローバル (InterSystems 多次元ストレージ・モデルの実装に使用されるツリーベースのスパース配列) への直接アクセス。これらのネイティブ・データ構造は、非常に高速で柔軟性に優れた格納と取得を実現します。InterSystems IRIS では、グローバルを使用してデータをオブジェクトまたはリレーショナル・テーブルとして使用できるようにしますが、Native SDK を使用して独自のデータ構造を実装することもできます。

以下の各章では、Native SDK for .NET の主な機能について説明します。

- .NET Native SDK の概要 ー Native SDK の機能の概要を示し、コードの簡単な例を示します。
- .NET からの ObjectScript メソッドおよび関数の呼び出し ー .NET アプリケーションから ObjectScript クラス・メソッドおよび関数を呼び出す方法について説明します。
- .NET 逆プロキシ・オブジェクトの使用法 ー .NET 逆プロキシ・オブジェクトを使用して ObjectScript オブジェクトを制御する方法を示します。
- グローバル配列の操作 ー 多次元グローバル配列のノードを作成、変更、または削除する方法について説明すると共に、反復処理およびデータ操作のメソッドを示します。
- トランザクションとロック ー .NET Native SDK トランザクションおよび並行処理の制御モデルを操作する方法について説明します。
- .NET Native SDK のクイック・リファレンス ー このドキュメントに記載されている各 Native SDK メソッドについて簡単に説明します。

InterSystems の Java 用の主要 SDK

Native SDK for .NET が含まれるスイートには、オブジェクトおよびリレーショナル・データベース・アクセスのための軽量 .NET SDK も含まれます。詳細は、以下のドキュメントを参照してください。

- InterSystems ソフトウェアでの .NET の使用法 ー リレーショナル・データ・アクセスに ADO.NET Managed Provider、オブジェクト・アクセスに Entity Framework のインターシステムズの実装を使用する方法について説明します。

- XEP による .NET オブジェクトの永続化 – InterSystems XEP SDK を使用して .NET オブジェクトを格納および取得する方法について説明します。

その他の言語用の Native SDK

Native SDK のバージョンは Java、Python、および Node.js でも使用できます。

- Native SDK for Java の使用法
- Native SDK for Python の使用法
- Native SDK for Node.js の使用法

グローバルに関する詳細情報

グローバルを自由自在に使いこなしたい開発者には、以下のドキュメントを強くお勧めします。

- グローバルの使用法 – ObjectScript でグローバルを使用する方法、およびサーバに多次元ストレージを実装する方法の詳細を示します。

2

.NET Native SDK の概要

Native SDK for .NET は、以前は ObjectScript を介してのみ使用可能であった、強力な InterSystems IRIS® リソースへの軽量インタフェースです。Native SDK により、アプリケーションは、InterSystems IRIS Data Platform とのシームレスな統合を利用できます。

- ObjectScript と .NET 間の透過的な双方向通信の実装

  Native SDK、オブジェクト・ゲートウェイ・プロキシ・オブジェクト、および ObjectScript アプリケーションすべてが同じ接続を共有し、同じコンテキストで連携できます。

- ObjectScript クラスの個別のインスタンスの作成および使用

  Native SDK を使用して、InterSystems IRIS で ObjectScript クラスのインスタンスを作成し、実行時にそれらのオブジェクト・ゲートウェイ・プロキシ・オブジェクトを生成できます。.NET アプリケーションは、プロキシを使用して、ネイティブの .NET オブジェクトのように簡単に ObjectScript オブジェクトを操作できます。

- ObjectScript クラス・メソッドおよびユーザ定義関数の呼び出し

  任意の目的でカスタム ObjectScript クラス・メソッドまたは関数を記述し、ネイティブの .NET メソッドと同じように簡単に、Native SDK を使用してアプリケーションからそれらを呼び出すことができます。

- 多次元グローバル配列の操作

  Native SDK は、InterSystems IRIS 多次元ストレージ・モデルを支える高性能ネイティブ・データ構造 (グローバル配列) への直接アクセスを提供します。グローバル配列は、ObjectScript と同様に .NET アプリケーションでも作成、読み取り、変更、および削除できます。

重要　Native SDK for .NET を使用するには、"接続ツール" で説明されている .NET 接続パッケージをダウンロードする必要があります。

以下の例で、これらの機能すべてを .NET アプリケーションに簡単に追加できることを示します。

ObjectScript と .NET 間の透過的な双方向通信の実装

Native SDK for .NET は、InterSystems ADO.NET Managed Provider の拡張機能として実装されます。接続は、Managed Provider を使用する他のアプリケーションと同じように作成します ("InterSystems Managed Provider for .NET の使用法" を参照)。この例では、接続を開いた後、Native SDK IRIS クラスのインスタンスを作成します。

```
//Open a connection to InterSystems IRIS
  IRISConnection conn = new IRISConnection();
  conn.ConnectionString = "Server = localhost; " + "Port = 1972; "
    + "Namespace = USER; " + "Password = SYS; " + "User ID = _SYSTEM;";
  conn.Open();

// Use the connection to create an instance of the Native SDK
  IRIS iris = IRIS.CreateIRIS(conn);
```

また、この接続を InterSystems オブジェクト・ゲートウェイで使用することで、.NET アプリケーションと ObjectScript アプリケーションは同じコンテキストを共有し、同じオブジェクトを操作することができます。

ObjectScript クラスの個別のインスタンスの作成および使用

アプリケーションで ObjectScript クラスのインスタンスを作成し、直ちにそのオブジェクト・ゲートウェイ・プロキシを生成し、このプロキシを使用して ObjectScript インスタンスを操作できます（“.NET 逆プロキシ・オブジェクトの使用法”の章を参照）。

この例では、最初の行が ObjectScript クラス **Demo.dataStore** の %New() メソッドを呼び出して、InterSystems IRIS でインスタンスを作成します。.NET では、この呼び出しによって dataStoreProxy という名前の対応するプロキシ・オブジェクトが返され、これを使用してインスタンス・メソッドが呼び出され、ObjectScript インスタンスのプロパティが取得または設定されます。

```
// use a classmethod call to create an ObjectScript instance and generate a proxy object
  IRISObject dataStoreProxy = (IRISObject)iris.ClassMethodObject("Demo.dataStore","%New");

// use the proxy to call instance methods, get and set properties
  dataStoreProxy.InvokeVoid("ititialize");
  dataStoreProxy.Set("propertyOne","a string property");
  String testString = dataStoreProxy.Get("propertyOne");
  dataStoreProxy.Invoke("updateLog","PropertyOne value changed to "+testString);

// pass the proxy back to ObjectScript method ReadDataStore()
  iris.ClassMethodObject("Demo.useData","ReadDataStore",dataStoreProxy);
```

この例の最後の行は、dataStoreProxy プロキシを ReadDataStore() という名前の ObjectScript メソッドに渡し、それを元の ObjectScript インスタンスへの参照として解釈します。そこからインスタンスをデータベースに保存して、別の ObjectScript アプリケーションに渡したり、.NET アプリケーションに返したりすることができます。

ObjectScript クラス・メソッドおよびユーザ定義関数の呼び出し

ObjectScript クラス・メソッドまたは関数は容易に呼び出すことができます（“ObjectScript メソッドおよび関数の呼び出し”の章を参照）。

```
  String currentNameSpace = iris.ClassMethodString("%SYSTEM.SYS","NameSpace");
```

この例では、単に何らかのシステム情報を取得するためにクラス・メソッドを呼び出していますが、これらの呼び出しの真の実力は、ユーザ記述コードを活用できることです。任意の目的でカスタム ObjectScript クラス・メソッドまたは関数を記述し、ネイティブの .NET メソッドを呼び出すのと同じように、簡単に .NET アプリケーションから呼び出すことができます。

多次元グローバル配列の操作

Native SDK は、グローバル配列の操作に必要なすべてのメソッドを提供しています（“グローバル配列の操作”の章を参照）。グローバルへのアクセスと操作、多次元グローバル配列の検索、およびデータ構造の調査を、ObjectScript と同様に簡単に行うことができます。以下の例は、簡易なグローバル配列を作成、読み取り、変更、および削除する方法を示しています。

```
// Create a global (ObjectScript equivalent: set ^myGlobal("subOne") = 10)
  iris.Set(10, "myGlobal", "subOne");

// Change, read, and delete the global
  iris.increment(2, "myGlobal", "subOne")   // increment value to 12
  Console.Write("New number is " + iris.GetInteger("myGlobal", "subOne"));
  iris.Kill("myGlobal", "subOne");
```

3

.NET からの ObjectScript メソッドおよび関数の呼び出し

この章では、ObjectScript クラス・メソッドとユーザ定義関数を .NET アプリケーションから直接呼び出すことを可能にするクラス **IRIS** のメソッドを説明します。詳細および例は、以下のセクションを参照してください。

- クラス・メソッドの呼び出し ー ObjectScript クラス・メソッドを呼び出す方法を示します。
- 関数の呼び出し ー ユーザ定義の ObjectScript 関数およびプロシージャを呼び出す方法を示します。
- クラス・ライブラリ・メソッドの呼び出し ー 引数の参照渡しを行い、**%Status** コードを確認する方法を示します。

3.1 クラス・メソッドの呼び出し

クラス **IRIS** の以下のメソッドは、ObjectScript クラス・メソッドを呼び出し、メソッド名により指定されるタイプの値を返します：ClassMethodBool()、ClassMethodBytes()、ClassMethodDouble()、classMethodIRISList()、ClassMethodLong()、ClassMethodObject()、ClassMethodString()、および ClassMethodVoid()。ClassMethodStatusCode() を使用して、ObjectScript **%Status** を返すクラス・メソッドからエラー・メッセージを取得します（“%Status エラー・コードの取得” を参照）。

これらのメソッドはすべて、className および methodName の **string** 引数に加え、**0** 個以上のメソッド引数を取ります。これは、**int?**、**short?**、**string**、**long?**、**double?**、**float?**、**byte[]**、**bool?**、**DateTime?**、IRISList?、または IRISObjectのいずれかの型にできます。接続が双方向の場合（“.NET 逆プロキシ・オブジェクトの使用法” を参照）、任意の .NET オブジェクトを引数として使用できます。Native SDK がこれらのデータ型を処理する方法の詳細は、“クラス IRIS のサポートされているデータ型” を参照してください。

すべての引数の数よりも少ない数の引数を渡すか、末尾の引数に対して **null** を渡すことで、引数リストで末尾の引数を省略できます。非 null 引数が null 引数の右側に渡されると、例外がスローされます。

ObjectScript クラス・メソッドの呼び出し

この例のコードでは、サポートされている各データ型のクラス・メソッドを ObjectScript テスト・クラス **User.NativeTest** から呼び出します（この例の直後に表示されています）。変数 iris はクラス **IRIS** の以前に定義されたインスタンスで、現在サーバに接続されていると想定します。

```
String className = "User.NativeTest";
String comment = "";
try{
comment = "cmBoolean() tests whether two numbers are equal (true=1,false=0): ";
bool boolVal = iris.ClassMethodBool(className,"cmBoolean",7,7);
```

```
  Console.WriteLine(comment+boolVal);

  comment = "cmBytes creates byte array [72,105,33]. String value of array: ";
  byte[] byteVal = iris.ClassMethodBytes(className,"cmBytes",72,105,33);
  Console.WriteLine(comment+ (new String(byteVal)));

  comment = "cmString() concatenates \"Hello\" + arg: ";
  string stringVal = iris.ClassMethodString(className,"cmString","World");
  Console.WriteLine(comment+stringVal);

  comment = "cmLong() returns the sum of two numbers: ";
  Long longVal = iris.ClassMethodLong(className,"cmLong",7,8);
  Console.WriteLine(comment+longVal);

  comment = "cmDouble() multiplies a number by 1.5: ";
  Double doubleVal = iris.ClassMethodDouble(className,"cmDouble",10);
  Console.WriteLine(comment+doubleVal);

  comment = "cmProcedure assigns a value to global array ^cmGlobal: ";
  iris.ClassMethodVoid(className,"cmVoid",67);
  // Read global array ^cmGlobal and then delete it
  Console.WriteLine(comment+iris.GetInteger("^cmGlobal"));
  iris.Kill("cmGlobal");

  comment = "cmList() returns a $LIST containing two values: ";
  IRISList listVal = iris.ClassMethodList(className,"cmList","The answer is ",42);
  Console.WriteLine(comment+listVal.Get(1)+listVal.Get(2));
  } catch (Exception e){
    Console.WriteLine("method failed");
  }
```

ObjectScript クラス User.NativeTest

前の例を実行するには、この ObjectScript クラスがコンパイルされ、サーバで使用可能である必要があります。

Class Definition

```
Class User.NativeTest Extends %Persistent
{
ClassMethod cmBoolean(cm1 As %Integer, cm2 As %Integer) As %Boolean
{
    Quit (cm1=cm2)
}
ClassMethod cmBytes(cm1 As %Integer, cm2 As %Integer, cm3 As %Integer) As %Binary
{
    Quit $CHAR(cm1,cm2,cm3)
}
ClassMethod cmString(cm1 As %String) As %String
{
    Quit "Hello "_cm1
}
ClassMethod cmLong(cm1 As %Integer, cm2 As %Integer) As %Integer
{
    Quit cm1+cm2
}
ClassMethod cmDouble(cm1 As %Double) As %Double
{
    Quit cm1 * 1.5
}
ClassMethod cmVoid(cm1 As %Integer)
{
    Set ^cmGlobal=cm1
    Quit ^cmGlobal
}
ClassMethod cmList(cm1 As %String, cm2 As %Integer)
{
    Set list = $LISTBUILD(cm1,cm2)
    Quit list
}
}
```

ターミナルからこれらのメソッドを呼び出すことで、それらをテストできます。例を以下に示します。

```
USER>write ##class(User.NativeTest).cmString("World")
Hello World
```

3.2 関数の呼び出し

関数の呼び出しは、メソッド呼び出しと似ていますが、引数の順序が異なります。関数ラベルを最初に指定し、その後に関数が含まれるルーチン名を続けます。これは ObjectScript で使用される順序に対応しており、関数呼び出しは以下の形式になります。

```
set result = $$myFunctionLabel^myRoutineName([arguments])
```

関数がサポートされている理由は古いコード・ベースに必要であるためで（"ObjectScript の使用法" の "呼び出し可能なユーザ定義コードモジュール" を参照）、可能であれば新しいコードでは常にメソッド呼び出しを使用するようにしてください。また、既存のルーチンのメソッド・ラッパを作成するための特別なキーワードも提供されているため、効率性が損なわれることはありません。CodeMode キーワードを `call` に設定してください。

このセクションの Native SDK メソッドは、ユーザ定義の ObjectScript 関数またはプロシージャを呼び出して、メソッド名で示されるタイプの値を返します：FunctionBool()、FunctionBytes()、FunctionDouble()、FunctionIRISList()、FunctionObject()、FunctionLong()、FunctionString()、または Procedure() (戻り値なし)。

これらは、functionLabel および routineName の String 引数に加え、0 個以上の関数の引数を取ります。これは、int?、short?、string、long?、double?、float?、byte[]、bool?、DateTime?、IRISList?、または IRISObjectのいずれかの型にできます。接続が双方向の場合 ("".NET 逆プロキシ・オブジェクトの使用法" を参照)、任意の .NET オブジェクトを引数として使用できます。Native SDK がこれらのデータ型を処理する方法の詳細は、"クラス IRIS のサポートされているデータ型" を参照してください。

すべての引数の数よりも少ない数の引数を渡すか、末尾の引数に対して `null` を渡すことで、引数リストで末尾の引数を省略できます。非 null 引数が null 引数の右側に渡されると、例外がスローされます。

注釈 **組み込みのシステム関数はサポートされません**

これらのメソッドは、ユーザ定義ルーチンで関数を呼び出すように設計されています。ObjectScript システム関数 ($ 文字で開始。"ObjectScript リファレンス" の "ObjectScript 関数" を参照) を .NET コードから直接呼び出すことはできません。ただし、システム関数を呼び出してその結果を返す ObjectScript ラッパ関数を記述することで、間接的にシステム関数を呼び出すことができます。例えば、fnList() 関数 (このセクションの最後にある "ObjectScript Routine NativeRoutine.mac" を参照) は $LISTBUILD を呼び出します。

Native SDK による ObjectScript ルーチンの関数の呼び出し

この例のコードでは、サポートされている各データ型の関数を ObjectScript ルーチン NativeRoutine から呼び出します (この例の直後に表示されているファイル NativeRoutine.mac)。iris はクラス IRIS の既存のインスタンスで、現在サーバに接続されていると想定します。

```
String routineName = "NativeRoutine";
String comment = "";

comment = "fnBoolean() tests whether two numbers are equal (true=1,false=0): ";
Bool boolVal = iris.FunctionBool("fnBoolean",routineName,7,7);
Console.WriteLine(comment+boolVal);

comment = "fnBytes creates byte array [72,105,33]. String value of the array: ";
Byte[] byteVal = new String(iris.FunctionBytes("fnBytes",routineName,72,105,33));
Console.WriteLine(comment+(new String(byteVal)));

comment = "fnString() concatenates \"Hello\" + arg: ";
String stringVal = iris.FunctionString("fnString",routineName,"World");
Console.WriteLine(comment+stringVal);

comment = "fnLong() returns the sum of two numbers: ";
Long longVal = iris.FunctionInt("fnLong",routineName,7,8);
Console.WriteLine(comment+longVal);

comment = "fnDouble() multiplies a number by 1.5: ";
Double doubleVal = iris.FunctionDouble("fnDouble",routineName,5);
Console.WriteLine(comment+doubleVal);
```

```
comment = "fnProcedure assigns a value to global array ^fnGlobal: ";
iris.Procedure("fnProcedure",routineName,88);
// Read global array ^fnGlobal and then delete it
Console.WriteLine(comment+iris.GetInteger("^fnGlobal")+"\n\n");
iris.Kill("fnGlobal");

comment = "fnList() returns a $LIST containing two values: ";
IRISList listVal = iris.FunctionList("fnList",routineName,"The answer is ",42);
Console.WriteLine(comment+listVal.Get(1)+listVal.Get(2));
```

ObjectScript ルーチン NativeRoutine.mac

前の例を実行するには、この ObjectScript ルーチンがコンパイルされ、サーバで使用可能である必要があります。

ObjectScript

```
fnBoolean(fn1,fn2) public {
   quit (fn1=fn2)
}
fnBytes(fn1,fn2,fn3) public {
    quit $CHAR(fn1,fn2,fn3)
}
fnString(fn1) public {
    quit "Hello "_fn1
}
fnLong(fn1,fn2) public {
    quit fn1+fn2
}
fnDouble(fn1) public {
    quit fn1 * 1.5
}
fnProcedure(fn1) public {
    set ^fnGlobal=fn1
    quit
}
fnList(fn1,fn2) public {
    set list = $LISTBUILD(fn1,fn2)
    quit list
}
```

ターミナルからこれらの関数を呼び出すことで、それらをテストできます。例を以下に示します。

```
USER>write $$fnString^NativeRoutine("World")
Hello World
```

3.3 クラス・ライブラリ・メソッドの呼び出し

InterSystems クラス・ライブラリのほとんどのクラスでは、メソッドが %Status 値のみを返す呼び出し規則を使用します。実際の結果は、参照によって渡される引数で返されます。このセクションでは、参照渡しを実行し、%Status 値を読み取る方法について説明します。

- 参照渡し引数の使用 – IRISReference クラスを使用して参照によってオブジェクトを渡す方法を示します。
- %Status エラー・コードの取得 – ClassMethodStatusCode() メソッドを使用して %Status 値をテストして読み取る方法を示します。

3.3.1 参照渡し引数の使用

Native SDK は、メソッドと関数の両方で参照渡しをサポートします。参照によって引数を渡すには、以下のように引数の値をクラス ADO.IRISReference のインスタンスに割り当て、そのインスタンスを引数として渡します。

```
IRISReference valueRef = new IRISReference(""); // set inital value to null string
iris.ClassMethodString("%SomeClass","SomeMethod",valueRef);
String myString = valueRef.value;               // get the method result
```

以下にその実際の例を示します。

参照渡し引数の使用

この例は、%SYS.DatabaseQuery.GetDatabaseFreeSpace() を呼び出して、iristemp データベースで利用可能な空き容量の量 (MB 単位) を取得します。

```
IRISReference freeMB = new IRISReference(0); // set inital value to 0
String dir = "C:/InterSystems/IRIS/mgr/iristemp"; // directory to be tested
Object status = null;

try {
  Console.Write("\n\nCalling %SYS.DatabaseQuery.GetDatabaseFreeSpace()... ");
  status = iris.ClassMethodObject("%SYS.DatabaseQuery","GetDatabaseFreeSpace",dir,freeMB);
  Console.WriteLine("\nFree space in " + dir + " = " + freeMB.value + "MB");
}
catch (IRISException e) {
  Console.Write("Call to class method GetDatabaseFreeSpace() returned error:");
  Console.WriteLine(e.getMessage());
}
```

出力：

```
Calling %SYS.DatabaseQuery.GetDatabaseFreeSpace()...
Free space in C:/InterSystems/IRIS/mgr/iristemp = 8.9MB
```

3.3.2 %Status エラー・コードの取得

クラス・メソッドが戻り値の型として ObjectScript %Status を持つ場合、ClassMethodStatusCode() を使用してエラー・メッセージを取得できます。クラス・メソッドの呼び出しが失敗した場合、結果として生成される IRISException エラーには、%Status エラー・コードとメッセージが含まれます。

以下の例では、ValidatePassword() メソッドは %Status オブジェクトを返します。パスワードが無効である場合 (例えばパスワードが短すぎる場合)、例外がスローされ、%Status メッセージにより失敗した理由が説明されます。変数 iris はクラス IRIS の以前に定義されたインスタンスで、現在サーバに接続されていると想定します。

ClassMethodStatusCode() を使用した ObjectScript %Status 値の取得

この例では、無効なパスワードを %SYSTEM.Security.ValidatePassword() に渡し、エラー・メッセージを取得します。

```
  String className = "%SYSTEM.Security";
  String methodName = "ValidatePassword";
  String pwd = ""; // an invalid password
  try {
// This call will throw an IRISException containing the %Status error message:
    iris.ClassMethodStatusCode(className,methodName,pwd);
// This call would fail silently or throw a generic error message:
    Object status = iris.ClassMethodObject(className,methodName,pwd);
    Console.WriteLine("\nPassword validated!");

  } catch (IRISException e) {
    Console.WriteLine("Call to "+methodName+"(\""+pwd+"\") returned error:");
    Console.WriteLine(e.getMessage());
  }
```

この例では、参照渡し引数を使用しないメソッドを意図的に呼び出していることに注意してください。

より複雑な例を試すには、前の例 ("参照渡し引数の使用") でステータス・コードの取得を試してみることができます。無効なディレクトリを渡すことで強制的に例外を発生させます。

注釈 **インスタンス・メソッドを呼び出す際の IRISObject.InvokeStatusCode() の使用**

ClassMethodStatusCode() メソッドは、クラス・メソッドの呼び出しに使用されます。プロキシ・オブジェクトのインスタンス・メソッドを呼び出す際 (".NET 逆プロキシ・オブジェクトの使用法" を参照)、IRISObject.InvokeStatusCode() メソッドをまったく同じ方法で使用できます。

4

.NET 逆プロキシ・オブジェクトの使用法

.NET Native SDK は .NET 外部サーバ接続を最大限に活用して、InterSystems IRIS と .NET アプリケーション間の完全に透過的な双方向接続を可能にします。

逆プロキシ・オブジェクトは、外部サーバ・ゲートウェイ接続を介して ObjectScript ターゲット・オブジェクトを制御するための .NET オブジェクトです。逆プロキシ・オブジェクトを使用してターゲットのメソッドを呼び出して、ターゲットのプロパティ値を取得または設定し、ネイティブの .NET オブジェクトのように簡単にターゲット・オブジェクトを操作できます。

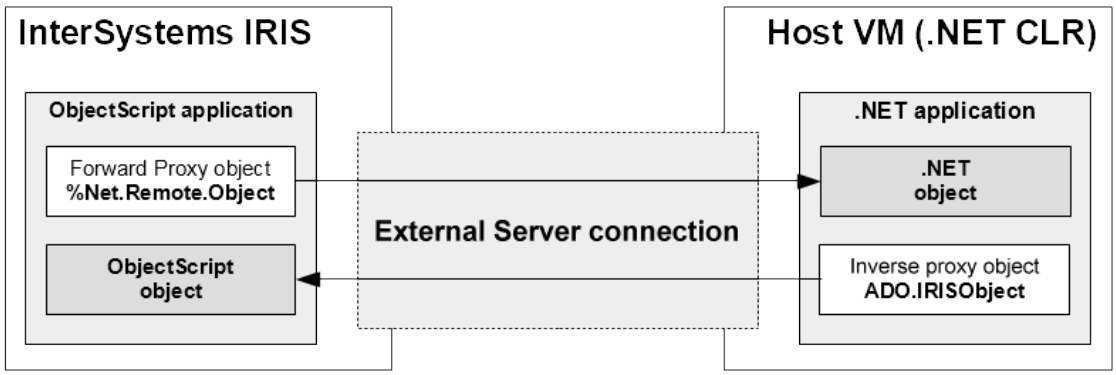
このセクションでは、以下のトピックについて説明します。

- 外部サーバの概要 ― 外部サーバの概要を示します。
- 逆プロキシ・オブジェクトの作成 ― 逆プロキシ・オブジェクトの作成に使用するメソッドについて説明します。
- ターゲット・オブジェクトの制御 ― 逆プロキシ・オブジェクトの使用法を示します。
- IRISObject のサポートされているデータ型 ― 逆プロキシ・メソッドのデータ型固有のバージョンについて説明します。

4.1 外部サーバの概要

外部サーバ接続によって、InterSystems IRIS ターゲット・オブジェクトと .NET オブジェクトは同じ接続を使用して、同じコンテキスト (データベース、セッション、トランザクション) で自由に相互作用することができます。外部サーバのアーキテクチャについては "InterSystems 外部サーバの使用法" で詳しく説明していますが、ここでは、外部サーバ接続を、一方の側のプロキシ・オブジェクトがもう一方の側のターゲット・オブジェクトを制御できる単純なブラック・ボックスとして考えることができます。

図 4-1: 外部サーバ接続



図に示すように、フォワード・プロキシ・オブジェクトは .NET オブジェクトを制御する ObjectScript プロキシです (詳細は、"InterSystems 外部サーバの使用法" の "外部言語の操作" を参照)。.NET 逆プロキシは反対方向に動作し、.NET アプリケーションが InterSystems IRIS ターゲット・オブジェクトを制御できるようにします。

4.2 逆プロキシ・オブジェクトの作成

逆プロキシ・オブジェクトを作成するには、ObjectScript クラス・インスタンスの OREF を取得して (通常は、クラスの %New() メソッドを呼び出して)、**IRISObject** にキャストします (詳細は、"ObjectScript メソッドおよび関数の呼び出し" を参照)。以下のメソッドを使用して、逆プロキシ・オブジェクトを生成できます。

- **ADO.IRIS.**ClassMethodObject() は ObjectScript クラス・メソッドを呼び出して、結果を **object** のインスタンスとして返します。
- **ADO.IRIS.**FunctionObject() は ObjectScript 関数を呼び出して、結果を **object** のインスタンスとして返します。

%New() メソッドが新しいターゲット・インスタンスを正常に作成すると、そのインスタンスの逆プロキシ・オブジェクトが生成されます。例えば、以下の呼び出しによって、ObjectScript クラス **Demo.Test** のインスタンスを制御する、test という逆プロキシ・オブジェクトが作成されます。

```
IRISObject test = (IRISObject)iris.ClassMethodObject("Demo.Test","%New");
```

- ClassMethodObject() は、**Demo.Test** という ObjectScript クラスの %New() メソッドを呼び出して、そのクラスの新しいターゲット・インスタンスを作成します。
- %New() への呼び出しでクラスの有効なインスタンスが返された場合、新しいインスタンスの逆プロキシが生成され、classMethodObject() はそれを **Object** として返します。
- .NET では、**Object** は **IRISObject** にキャストされ、逆プロキシ変数 test が作成されます。

変数 test は、**Demo.Test** の新しいターゲット・インスタンスの .NET 逆プロキシ・オブジェクトです。以下のセクションでは、test を使用して **Demo.Test** ターゲット・インスタンスのメソッドとプロパティにアクセスします。

4.3 ターゲット・オブジェクトの制御

逆プロキシ・オブジェクトは **IRISObject** のインスタンスです。ターゲット・インスタンス・メソッドを呼び出すメソッド Invoke() と InvokeVoid()、およびターゲットのプロパティの読み取りと書き込みを行うアクセサ Get() と Set() を提供します。このセクションの例では、逆プロキシを使用して、ObjectScript クラス **Demo.Test** のターゲット・インスタンスを制御します。このクラスには、メソッド initialize() と add()、およびプロパティ name の宣言が含まれています。

ObjectScript クラス Demo.Test のメソッドとプロパティの宣言

```
Class Demo.Test Extends %Persistent
   Method initialize(initialVal As %String)
   Method add(val1 As %Integer, val2 As %Integer) As %Integer
   Property name As %String
```

以下の例では、最初の行で **Demo.Test** の新規インスタンスの test という名前の逆プロキシ・オブジェクトを作成します (前のセクションの説明を参照)。残りのコードでは、test を使用して **Demo.Test** ターゲット・インスタンスを制御します。

逆プロキシ・オブジェクトによる Demo.Test のインスタンスの制御

```
// Create an instance of Demo.Test and return a proxy object for it
  IRISObject test = (IRISObject)iris.ClassMethodObject("Demo.Test","%New");

// instance method test.initialize() is called with one argument, returning nothing.
  test.InvokeVoid("initialize", "Test One");

// instance method test.add() is called with two arguments, returning an int value.
  int sum = test.Invoke("add",2,3);  // adds 2 plus 3, returning 5

// The value of property test.name is set and then returned.
  test.Set("name", "Einstein, Albert");  // sets the property to "Einstein, Albert"
  String name = test.Get("name");   // returns the new property value
```

この例では、以下の IRISObject メソッドを使用して、Demo.Test インスタンスのメソッドとプロパティにアクセスしました。

- ClassMethodObject() は、Demo.Test クラス・メソッド %New() を呼び出します。このメソッドは Demo.Test のインスタンスを作成し、test という名前の IRISObject プロキシを返します (“リバース・プロキシ・オブジェクトの作成” で前述しています)。
- InvokeVoid() は、initialize() インスタンス・メソッドを呼び出します。このメソッドは、内部変数を初期化しますが、値は返しません。
- Invoke() は、add() インスタンス・メソッドを起動します。このメソッドは、2 つの整数の引数を受け入れて、合計を整数として返します。
- Set() は、name プロパティを新しい値に設定します。
- Get() は、name プロパティの値を返します。

また、これらのメソッドのデータ型固有のバージョンもありますが、これらについては以下のセクションで説明します。

4.4 IRISObject のサポートされているデータ型

前のセクションの例では、汎用の Set()、Get()、および Invoke() メソッドを使用しましたが、IRISObject クラスでサポートされるデータ型用にデータ型固有のメソッドも提供します。

IRISObject set() および get() メソッド

IRISObject.Set() メソッドは、IRIS.Set() でサポートされるすべてのデータ型を含む、任意の .NET オブジェクトをプロパティ値として受け入れます (“クラス IRIS のサポートされているデータ型” を参照)。

汎用の Get() メソッドに加えて、IRISObject は GetBool()、GetBytes()、GetDouble()、GetIRISList()、GetLong()、GetObject()、および GetString() のデータ型固有のメソッドを提供します。

IRISObject invoke() メソッド

IRISObject invoke メソッドは、IRIS クラス・メソッド呼び出しと同じセットのデータ型をサポートしています (“クラス・メソッドの呼び出し” を参照)。

汎用の Invoke() メソッドに加えて、IRISObject は、InvokeBool()、InvokeBytes()、InvokeDouble()、InvokeIRISList()、InvokeLong()、InvokeObject()、InvokeString()、および InvokeVoid() のデータ型固有のメソッドを提供します。また、InvokeStatusCode() も提供します。これは、ObjectScript %Status 返り値の内容を取得します (“%Status エラー・コードの取得” を参照)。

Invoke メソッドはすべて、methodName の String 引数に加え、0 個以上のメソッド引数を取ります。これは、int?、short?、string、long?、double?、float?、byte[]、bool?、DateTime?、IRISList?、または IRISObjectのいずれかの型にできます。接続が双方向の場合、任意の .NET オブジェクトを引数として使用できます。

すべての引数の数よりも少ない数の引数を渡すか、末尾の引数に対して `null` を渡すことで、引数リストで末尾の引数を省略できます。非 null 引数が null 引数の右側に渡されると、例外がスローされます。

5
グローバル配列の操作

.NET Native SDK は、InterSystems IRIS のインスタンスからグローバル配列を操作するメカニズムを提供します。この章では、以下の項目について説明します。

- グローバル配列の概要 – グローバル配列の概念を示し、.NET Native SDK がどのように使用されるかを簡単に説明します。
- ノードの作成、アクセス、および削除 – グローバル配列のノードを作成、変更、または削除する方法、およびノード値を取得する方法を示します。
- グローバル配列のノードの検索 – グローバル配列のノードへの高速アクセスを可能にする反復メソッドについて説明します。
- クラス IRIS のサポートされているデータ型 – ノード値を特定のデータ型として取得する方法の詳細を説明します。

注釈 **.NET 接続の作成**

この章の例では、iris という名前の **IRIS** オブジェクトが既に存在し、サーバに接続されていると想定します。以下のコードは、標準の .NET 接続を確立し、**IRIS** のインスタンスを作成します。

```
//Open a connection to the server and create an IRIS object
        IRISConnection conn = new IRISConnection();
        conn.ConnectionString = "Server = localhost; "
        + "Port = 1972; " + "Namespace = USER; "
        + "Password = SYS; " + "User ID = _SYSTEM;";
        conn.Open();
        IRIS iris = IRIS.CreateIRIS(conn);
```

IRIS のインスタンスの作成方法の詳細は、CreateIRIS() のクイック・リファレンスのエントリを参照してください。.NET 接続の作成方法の一般的な情報は、"InterSystems ソフトウェアでの .NET の使用法" の "接続の作成" を参照してください。.

5.1 グローバル配列の概要

グローバル配列は、すべてのスパース配列のように、ツリー構造です (シーケンシャル・リストではありません)。グローバル配列の背後にある基本概念は、ファイル構造に例えて示すことができます。ツリーの各ディレクトリは、ルート・ディレクトリ識別子とそれに続く一連のサブディレクトリ識別子で構成されるパスによって一意に識別され、ディレクトリにはデータが含まれることも、含まれないこともあります。

グローバル配列の仕組みも同じです。ツリーの各ノードは、グローバル名識別子と一連の添え字識別子で構成されるノード・アドレスによって一意に識別され、ノードには値が含まれることも、含まれないこともあります。例えば、以下は 6 つのノードで構成されるグローバル配列で、そのうち 2 つのノードには値が含まれます。

```
root -->|--> foo --> SubFoo="A"
        |--> bar --> lowbar --> UnderBar=123
```

値はその他のノード・アドレス (root または root->bar など) に格納できますが、それらのノード・アドレスが値なしの場合、リソースは無駄になりません。InterSystems ObjectScript グローバルの表記では、値を持つ 2 つのノードは次のようになります。

```
root("foo","SubFoo")
root("bar","lowbar","UnderBar")
```

グローバル名 (root) の後に、括弧で囲まれたコンマ区切り添え字リストが続きます。この両者で、ノードのパス全体を指定します。

このグローバル配列は、Native SDK Set() メソッドへの 2 つの呼び出しで作成されます。

```
irisObject.Set("A", "root", "foo", "SubFoo");
irisObject.Set(123, "root", "bar", "lowbar", "UnderBar");
```

グローバル配列 root は、最初の呼び出しが値 "A" をノード root("foo","SubFoo") に割り当てるときに作成されます。ノードは任意の順序で、任意の添え字セットを使用して作成できます。これら 2 つの呼び出しの順序を逆にした場合でも、同じグローバル配列が作成されます。値なしノードは自動的に作成され、不要になると自動的に削除されます。詳細は、この章で後述する“ノードの作成、アクセス、および削除”を参照してください。

この配列を作成する Native SDK コードを、以下に例示します。IRISConnection オブジェクトは、サーバへの接続を確立します。この接続は、iris という名前のクラス IRIS のインスタンスによって使用されます。Native SDK メソッドを使用してグローバル配列を作成し、生成された永続値をデータベースから読み取った後、グローバル配列を削除します。

NativeDemo プログラム

Native SDK for .NET は、InterSystems.Data.IrisClient.dll ライブラリに含まれます。詳細は、"InterSystems ソフトウェアでの .NET の使用法" の "Introduction" を参照してください。

```
using System;
using InterSystems.Data.IRISClient;
using InterSystems.Data.IRISClient.ADO;

namespace NativeSpace {
  class NativeDemo {
    static void Main(string[] args) {
      try {

//Open a connection to the server and create an IRIS object
        IRISConnection conn = new IRISConnection();
        conn.ConnectionString = "Server = localhost; "
        + "Port = 1972; " + "Namespace = USER; "
        + "Password = SYS; " + "User ID = _SYSTEM;";
        conn.Open();
        IRIS iris = IRIS.CreateIRIS(conn);

//Create a global array in the USER namespace on the server
        iris.Set("A", "root", "foo", "SubFoo");
        iris.Set(123, "root", "bar", "lowbar", "UnderBar");

// Read the values from the database and print them
        string subfoo = iris.GetString("root", "foo", "SubFoo");
        string underbar = iris.GetString("root", "bar", "lowbar", "UnderBar");
        Console.WriteLine("Created two values: \n"
          + " root(\"foo\",\"SubFoo\")=" + subfoo + "\n"
          + " root(\"bar\",\"lowbar\",\"UnderBar\")=" + underbar);

//Delete the global array and terminate
        iris.Kill("root"); // delete global array root
        iris.Close();
        conn.Close();
      }
      catch (Exception e) {
```

```
            Console.WriteLine(e.Message);
          }
        }// end Main()
      } // end class NativeDemo
    }
```

NativeDemo は、以下の行を出力します。

```
Created two values:
   root("foo","SubFoo")=A
   root("bar","lowbar","UnderBar")=123
```

この例では、conn という名前の IRISConnection オブジェクトが、USER ネームスペースに関連付けられているデータベースへの接続を提供します。Native SDK メソッドは以下のアクションを実行します。

- IRIS.CreateIRIS() は、conn を介してデータベースにアクセスする、iris という名前の IRIS の新しいインスタンスを作成します。
- IRIS.Set() は、新しい永続ノードをデータベースに作成します。
- IRIS.GetString() は、データベースに対してクエリを実行して、指定されたノードの値を返します。
- IRIS.Kill() は、指定されたノードとそのサブノードすべてをデータベースから削除します。

次の章では、これらのすべてのメソッドについて詳細に説明し、例を示します。

5.1.1 Native SDK 用語の用語集

ここに示す概念の概要については、前のセクションを参照してください。この用語集の例は、以下に示すグローバル配列構造を指しています。Legs グローバル配列には、10 個のノードと 3 つのノード・レベルがあります。10 個のノードのうち 7 つには、値が含まれます。

```
Legs                      // root node, valueless, 3 child nodes
  fish = 0                // level 1 node, value=0
  mammal                  // level 1 node, valueless
    human = 2             // level 2 node, value=2
    dog = 4               // level 2 node, value=4
  bug                     // level 1 node, valueless, 3 child nodes
    insect = 6            // level 2 node, value=6
    spider = 8            // level 2 node, value=8
    millipede = Diplopoda // level 2 node, value="Diplopoda", 1 child node
      centipede = 100     // level 3 node, value=100
```

子ノード

指定された親ノードの直下にあるノードです。子ノードのアドレスは、親添え字リストの末尾に 1 つの添え字を追加して指定します。例えば、親ノード Legs("mammal") には子ノード Legs("mammal","human") および Legs("mammal","dog") があります。

グローバル名

ルート・ノードの識別子は、グローバル配列全体の名前でもあります。例えば、ルート・ノード識別子 Legs は、グローバル配列 Legs のグローバル名です。

ノード

グローバル配列の要素で、グローバル名と任意の数の添え字識別子で構成されるネームスペースによって一意に識別されます。ノードは、データを含むか、子ノードを持つか、またはこれらの両方を持つ必要があります。

ノード・レベル

ノード・アドレス内の添え字の数。'レベル 2 ノード' は、'2 つの添え字を持つノード' のもう 1 つの表現方法です。例えば、Legs("mammal","dog") は、レベル 2 ノードです。ルート・ノード Legs の 2 レベル下で、Legs("mammal") の 1 レベル下です。

ノード・アドレス

グローバル名とすべての添え字を含む、ノードの完全なネームスペースです。例えば、ノード・アドレスLegs(″fish″)は、ルート・ノード識別子 `Legs` と 1 つの添え字 `"fish"` を含むリストで構成されます。コンテキストに応じて、Legs (添え字リストなし) はルート・ノード・アドレスまたはグローバル配列全体を参照することができます。

ルート・ノード

グローバル配列ツリーの基点にある添え字なしノードです。ルート・ノードの識別子は、その添え字なしのグローバル名です。

サブノード

特定のノードのすべての下位ノードは、そのノードのサブノードと呼ばれます。例えば、ノード Legs(″bug″) には 2 つのレベルの 4 つの異なるサブノードがあります。9 つの添え字付きノードはすべて、ルート・ノード Legs のサブノードです。

添え字/添え字リスト

ルート・ノードの下にあるノードはすべて、グローバル名および 1 つまたは複数の添え字識別子のリストを指定して処理されます(グローバル名に添え字リストを加えたものが、ノード・アドレスです)。

ターゲット・アドレス

多くの Native SDK メソッドは有効なノード・アドレスを指定する必要がありますが、ノード・アドレスは必ずしも既存のノードを指す必要はありません。例えば、Set() メソッドは value 引数とターゲット・アドレスを取り、そのアドレスに値を格納します。ターゲット・アドレスにノードが存在しない場合は、新しいノードが作成されます。

値

ノードには、サポートされている任意の型の値を含めることができます。子ノードを持たないノードには値を含める必要があり、子ノードを持つノードは値なしにすることができます。

値なしノード

ノードは、データを含むか、子ノードを持つか、またはこれらの両方を持つ必要があります。子ノードを持っているがデータを含まないノードは、値なしノードと呼ばれます。値なしノードは、下位レベルのノードへのポインタとしてのみ存在します。

5.1.2 グローバル命名規則

グローバル名と添え字は、次の規則に従います。

- ノード・アドレスの長さ (グローバル名とすべての添え字の長さの合計) は最大 511 文字です(入力した一部の文字は、この制限のために、複数のエンコードされた文字としてカウントされることがあります。詳細は、“グローバル参照の最大長” を参照してください)。
- グローバル名には文字、数字、およびピリオド (`'.'`) を使用でき、最大 31 文字の有効文字を使用できます。文字で始まる必要があり、ピリオドで終了することはできません。
- 添え字は文字列または数値にすることができます。文字列の添え字は大文字と小文字が区別され、すべての文字 (制御文字と非表示文字を含む) を使用できます。長さの制限は、ノード・アドレス全体の最大長が 511 文字という制限のみです。

5.2 ノードの作成、アクセス、および削除

Native SDK には、データベース内で変更を実行することのできるメソッドが 3 つ用意されています。Set() および Increment() はノード値を作成または変更することができ、Kill() は 1 つのノードまたは一連のノードを削除することができます。ノード値はタイプ固有のゲッター・メソッド (GetInteger() や GetString()) などで取得します。

- ノードの作成とノード値の設定 – Set() と Increment() の使用方法について説明します。
- ノード値の取得 – サポートされている各データ型のゲッター・メソッドのリストを示します。
- ノードの削除 – Kill() の使用方法について説明します。

5.2.1 ノードの作成とノード値の設定

Set() メソッドと Increment() メソッドを使用して、指定した値を持つ永続ノードを作成したり、既存のノードの値を変更したりできます。

IRIS.Set() は、任意のサポートされているデータ型の value 引数を取り、指定されたアドレスにその値を格納します。ターゲット・アドレスにノードが存在しない場合は、新しいノードが作成されます。

ノード値の設定および変更

以下の例では、Set() の最初の呼び出しによって、新しいノードがサブノード・アドレス myGlobal("A") に作成され、このノードの値が文字列 `"first"` に設定されます。2 回目の呼び出しによって、サブノードの値が変更され、整数 `1` に置き換えられます。

```
iris.Set("first", "myGlobal", "A");     // create node myGlobal("A") = "first"
iris.Set(1, "myGlobal", "A");   // change value of myGlobal("A") to 1.
```

Set() は、サポートされている任意のデータ型の値を作成および変更できます。既存の値を読み取るには、次のセクションで説明するように、データ型ごとに異なるゲッター・メソッドを使用する必要があります。

IRIS.Increment() は number 引数を取り、その数だけノードの値をインクリメントし、そのインクリメントした値を返します。

ターゲット・アドレスにノードがない場合、このメソッドはノードを作成し、値として number 引数を割り当てます。このメソッドはスレッドセーフなアトミック処理を使用して、ノードの値を変更します。したがって、ノードは決してロックされません。

ノード値のインクリメント

以下の例では、Increment() の最初の呼び出しによって、新しいサブノード myGlobal("B") が値 `-2` で作成されます。次の 2 回の呼び出しによって、それぞれ `-2` だけインクリメントされ、最終値は `-6` になります。

```
for (int loop = 0; loop < 3; loop++) {
  iris.Increment(-2,"myGlobal", "B");
}
```

注釈 **グローバル命名規則**

Set() または Increment() の 2 番目の引数は、グローバル配列名です。グローバル配列名には、文字、数字、およびピリオドを使用できます。名前は文字で始まる必要があり、ピリオドで終了することはできません。グローバル名の後の引数は添え字で、これには数値または文字列を指定できます (大文字と小文字が区別され、英数字に限定されません)。詳細は、“グローバル命名規則” を参照してください。

5.2.2 ノード値の取得

Set() メソッドはサポートされているすべてのデータ型で使用できますが、各データ型には別個のゲッターが必要です。ノード値は、以下のいずれかのデータ型になります：int?、short?、string、long?、double?、float?、byte[]、bool?、DateTime?、IRISList?、および System.IO.MemoryStream を実装するオブジェクトのインスタンス (byte[] として格納および取得)。NULL 値は "" に変換されます。

以下のメソッドを使用して、これらのデータ型からノード値を取得します。

- 数値データ型：GetBool()、GetInt16()、GetInt32()、GetInt64()、GetSingle()、GetDouble()
- 文字列データ型：GetString()、GetBytes()、GetIRISList()
- その他のデータ型：GetDateTime()、GetObject()

データ型の詳細は、この章で後述する“クラス IRIS のサポートされているデータ型”を参照してください。

5.2.3 ノードの削除

IRIS.Kill() は、指定したノードとそのすべてのサブノードを削除します。ルート・ノードを削除した場合、または値を持つすべてのノードを削除した場合、グローバル配列全体が削除されます。

以下の例では、グローバル配列 myGlobal には最初、以下のノードが含まれています。

```
myGlobal = <valueless node>
  myGlobal("A") = 0
    myGlobal("A",1) = 0
    myGlobal("A",2) = 0
  myGlobal("B") = <valueless node>
    myGlobal("B",1) = 0
```

この例では、myGlobal(″A″) と myGlobal(″B″,1) の 2 つのサブノードで Kill() を呼び出して、グローバル配列全体を削除します。

ノードまたはノード・グループの削除

最初の呼び出しによって、ノード myGlobal(″A″) とその両方のサブノードが削除されます。

```
iris.Kill("myGlobal", "A");
// also kills child nodes myGlobal("A",1) and myGlobal("A",2)
```

2 番目の呼び出しによって、値を持つ最後の残りのサブノード myGlobal(″B″,1) が削除されます。

```
iris.Kill("myGlobal","B",1);
```

残りのどのノードにも値がない場合は、グローバル配列全体が削除されます。

- 親ノード myGlobal(″B″) は、値がなく、サブノードもなくなったため、削除されます。
- その結果、ルート・ノード myGlobal は値がなく、サブノードもなくなったため、グローバル配列全体がデータベースから削除されます。

5.3 グローバル配列のノードの検索

Native SDK は、1 つのグローバル配列の一部または全部に対して反復処理する方法を提供します。以下のトピックでは、さまざまな反復メソッドについて説明します。

- 一連の子ノードにわたる反復 – 指定した親ノードの下のすべての子ノードにわたって反復する方法について説明します。
- 条件付きループでの反復処理 – 反復処理を詳細に制御するメソッドについて説明します。
- 子ノードおよびノード値のテスト – ノード・レベルに関係なくすべてのサブノードを検索して、値を持つノードを識別する方法について説明します。

5.3.1 一連の子ノードにわたる反復

子ノードは、同じ親ノードの直下にある一連のノードです。親の添え字リストに 1 つの添え字を追加することで、子ノード・アドレスを定義できます。例えば、以下のグローバル配列には、親ノード heroes("dogs") の下に 4 つの子ノードがあります。

heroes グローバル配列

このグローバル配列では、いくつかのヒーロー犬 (および無謀な少年と先導する羊) の名前を添え字として使用します。値は生まれた年です。

```
heroes                                          // root node,    valueless, 2 child nodes
   heroes("dogs")                               // level 1 node, valueless, 4 child nodes
      heroes("dogs","Balto") = 1919             // level 2 node, value=1919
      heroes("dogs","Hachiko") = 1923           // level 2 node, value=1923
      heroes("dogs","Lassie") = 1940            // level 2 node, value=1940, 1 child node
         heroes("dogs","Lassie","Timmy") = 1954 // level 3 node, value=1954
      heroes("dogs","Whitefang") = 1906         // level 2 node, value=1906
   heroes("sheep")                              // level 2 node, valueless, 1 child node
      heroes("sheep","Dolly") = 1996            // level 2 node, value=1996
```

以下のメソッドを使用して反復子を作成し、反復処理の方向を定義し、検索の開始位置を設定します。

- IRIS.GetIRISIterator() は、指定されたターゲット・ノードの子ノードの IRISIterator のインスタンスを返します。
- IRIS.GetIRISReverseIterator() は、指定されたターゲット・ノードの子ノードの、逆方向の反復処理に設定された IRISIterator のインスタンスを返します。
- IRISIterator.StartFrom() は、反復子の開始位置を指定の添え字に設定します。添え字は任意の開始ポイントであり、既存のノードを処理する必要はありません。

逆順序での子ノード値の読み取り

以下のコードは、heroes("dogs") の子ノードを逆方向の照合順序で反復処理します (添え字 V から開始します)。

```
// Iterate in reverse, seeking nodes lower than heroes('dogs','V') in collation order
    IRISIterator iterDogs = iris.GetIRISReverseIterator("heroes","dogs");
    iterDogs.StartFrom("V");
    String output = "\nDog birth years: ";
    foreach (int BirthYear in iterDogs) {
        output += BirthYear + " ";
    };
    Console.WriteLine(output);
```

このコードによって、以下のような出力が生成されます。

```
Dog birth years: 1940 1923 1919
```

この例は、以下を実行します。

- GetIRISReverseIterator() は、heroes("dogs") の子ノードを逆方向の照合順序で検索する、反復子 iterDogs を返します。

- StartFrom() は、添え字 V を指定します。すなわち、検索範囲には、照合順序が V よりも下位の添え字を持つ heroes("dogs") のすべての子コードが含まれます。反復子は、まず添え字 Lassie を検索し、続いて Hachiko および Balto を検索します。

heroes("dogs") の以下の 2 つのサブノードは無視されます。

- 子ノード heroes("dogs","Whitefang") は検索範囲外にあるため、見つかりません (Whitefang は照合順序が V より上位です)。
- レベル 3 ノード heroes("dogs","Lassie","Timmy") は、dogs ではなく Lassie の子であるため、見つかりません。

複数のノード・レベルにわたって反復処理する方法については、この章の最後のセクション (“子ノードおよびノード値のテスト”) を参照してください。

注釈 **照合順序**

ノードが取得される順序は、添え字の照合順序によって決まります。これは、反復子の機能ではありません。ノードが作成されると、ストレージ定義で指定された照合順に、自動的にノードが格納されます。この例では、heroes("dogs") の子ノードは、作成された順序に関係なく、表示されている順序 (Balto、Hachiko、Lassie、Whitefang) で格納されます。詳細は、"グローバルの使用法" の “グローバル・ノードの照合” を参照してください。

5.3.2 条件付きループでの反復処理

前のセクションでは、一連の子ノードに対して 1 つのパスを作成する簡単な方法を示しましたが、簡易な foreach ループで提供できる以上の制御が必要になる場合があります。このセクションでは、反復子をより詳細に制御し、データに容易にアクセスできるメソッドとプロパティをいくつか示します。

- IRISIterator.MoveNext() は、System.Collections.IEnumerator を実装して、反復子がいつ次のノードに移動するかを正確に制御できるようにします。次のノードが見つかった場合は true を返し、現在の反復処理にこれ以上ノードがない場合は false を返します。
- IRISIterator.Reset() をループ終了後に呼び出して、反復子を開始位置にリセットし、再び使用できるようにすることができます。
- IRISIterator.Current は、現在の反復子位置にあるノードの値を含むオブジェクトを取得します。これは、foreach ループで現在のループ変数に割り当てられている値と同じです。
- IRISIterator.CurrentSubscript は、現在の反復子位置にあるノードの最下位の添え字を含むオブジェクトを取得します。例えば、反復子がノード myGlobal(23,"somenode") を指す場合、返されるオブジェクトには値 "somenode" が含まれます。

前の例と同じように、この例でも heroes グローバル配列を使用し、heroes("dogs") の下の子ノードを反復処理します。ただし、この例では、同じ反復子を使用して、子ノードで何度かパスを実行し、特定の条件が満たされたらすぐにループを終了します。

リストの項目に一致する値の検索

この例では、特定のノード値が見つかるか、ノードがなくなるまで、heroes("dogs") の下の子ノードをスキャンします。配列 targetDates は、メインの `foreach` ループで使用される targetYear 値のリストを指定します。メイン・ループ内で、`do while` ループは、各子ノードを検索して、その値を現在の targetYear と比較します。

```
IRISIterator iterDogs = iris.GetIRISIterator("heroes","dogs");
bool seek;
int[] targetDates = {1906, 1940, 2001};
foreach (int targetYear in targetDates) {
  do {
    seek = iterDogs.MoveNext();
    if (!seek) {
      Console.WriteLine("Could not find a dog born in " + targetYear);
    }
    else if ((int)iterDogs.Current == targetYear) {
      Console.WriteLine(iterDogs.CurrentSubscript + " was born in " + iterDogs.Current);
      seek = false;
    }
  } while (seek);
  iterDogs.Reset();
} // end foreach
```

このコードによって、以下のような出力が生成されます。

```
Whitefang was born in 1906
Lassie was born in 1940
Could not find a dog born in 2001
```

この例は、以下を実行します。

- GetIRISIterator() は、heroes("dogs") の子ノードを照合順に検索する、反復子 iterDogs を返します (前のセクションの "一連の子ノードにわたる反復" に示されています)。iterDogs はリセットされ、`foreach` ループの各パスで再び使用されます。
- MoveNext() は、次の子ノードを検索するために、`do while` ループの各パスで呼び出されます。ノードが見つかった場合、seek は `true` に設定され、これ以上子コードがない場合は `false` に設定されます。seek が `false` の場合、現在の targetYear 値が見つからなかったことを示すメッセージを出力した後、`do while` ループは終了します。
- 子ノードが見つかるたびに、iterDogs の Current および CurrentSubscript プロパティが設定されます。Current には現在のノード値が含まれ、CurrentSubscript には現在の添え字が含まれます。
- Current は、targetYear と比較されます。一致がある場合、メッセージに添え字とノード値の両方が表示され、seek を `false` に設定して、`do while` ループは終了します。
- Reset() は、各 `do while` パスの最後に呼び出されます。これにより、反復子 iterDogs は元の開始条件に戻され、次のパスで再び使用できるようになります。

5.3.3 子ノードおよびノード値のテスト

前の例では、検索の範囲は heroes("dogs") の子ノードに制限されています。グローバル配列 heroes の以下の 2 つの値は異なる親の下にあるため、反復子はこれらの値を見つけられません。

- レベル 3 ノード heroes("dogs","Lassie","Timmy") は、`dogs` ではなく `Lassie` の子であるため、見つかりません。
- レベル 2 ノード heroes("sheep","Dolly") は、`dogs` ではなく `sheep` の子であるため、見つかりません。

グローバル配列全体を検索するには、子ノードを持つすべてのノードを見つけて、それぞれに反復子を作成する必要があります。IsDefined() メソッドは、必要な情報を提供します。

- IRIS.IsDefined() ー これを使用して、ノードに値、サブノード、またはその両方があるかどうかを確認できます。以下の値を返します。

- `0` ー 指定したノードは存在しません。
- `1` ー ノードは存在し、値があります。
- `10` ー ノードに値はありませんが、子ノードがあります。
- `11` ー ノードには値と子ノードの両方があります。

返り値を使用して、いくつかの有用なブーリアン値を確認できます。

```
bool exists = (iris.IsDefined(root,subscripts) > 0);      // value is 1, 10, or 11
bool hasValue = (iris.IsDefined(root,subscripts)%10 > 0); // value is 1 or 11
bool hasChild = (iris.IsDefined(root,subscripts) > 9);    // value is 10 or 11
```

以下の例は、2 つのメソッドで構成されます。

- TestNode() は、heroes グローバル配列のノードごとに呼び出されます。現在のノードで IsDefined() を呼び出して、ノードに子ノードがあるかどうかを示すブーリアン値を返します。現在の添え字が `Timmy` または `Dolly` であるかも確認し、そうである場合はメッセージを出力します。
- FindAllHeroes() は、TestNode() の返り値を使用して、グローバル配列全体を操作します。まず、ルート・ノード heroes の子ノードを反復処理します。TestNode() によって現在のノードに子ノードがあると示された場合、FindAllHeroes() は新しい反復子を作成して、下位レベルの子ノードをテストします。

メソッド FindAllHeroes()

この例では、既知の構造を処理して、入れ子になった単純な呼び出しを使用してさまざまなレベルを検索します。構造に任意の数のレベルがあるようなあまり一般的でない場合には、再帰アルゴリズムを使用できます。

```
public void FindAllHeroes() {
  string root = "heroes";

// Iterate over child nodes of root node heroes
  IRISIterator iterRoot = iris.GetIRISIterator(root);
  foreach (object node in iterRoot) {
    object sub1 = iterRoot.CurrentSubscript;
    bool hasChild1 = TestNode(iterRoot,sub1);

    // Process current child of heroes(sub1)
    if (hasChild1) {
      IRISIterator iterOne = iris.GetIRISIterator(root,sub1);
      foreach (object node in iterOne) {
        object sub2 = iterOne.CurrentSubscript;
        bool hasChild2 = TestNode(iterOne,sub1,sub2);

      // Process current child of heroes(sub1,sub2)
        if (hasChild2) {
          IRISIterator iterTwo = iris.GetIRISIterator(root,sub1,sub2);
          foreach (object node in iterTwo) {
            object sub3 = iterTwo.CurrentSubscript;
            TestNode(iterTwo,sub1,sub2,sub3); //no child nodes below level 3
          }
        } //end hasChild2
      }
    } //end hasChild1
  } // end main loop
} // end FindAllHeroes()
```

メソッド TestNode()

```
public bool TestNode(IRISIterator iter, string root, params object[] subscripts) {
// Test for values and child nodes
  int state = iris.IsDefined(root,subscripts);
  bool hasValue = (state%10 > 0); // has value if state is 1 or 11
  bool hasChild = (state > 9);    // has child if state is 10 or 11

// Look for lost heroes


//    string[] lost = {"Timmy","Dolly"};
```

```
    var lost = new List<string> {"Timmy","Dolly"};

    if (hasValue) { // ignore valueless nodes
      string name = (string)iter.CurrentSubscript;
      int year = (int)iter.Current;
//      foreach (string hero in lost) {
//        if (hero == name) {
      if (lost.Contains(name))
        Console.WriteLine("Hey, we found " + name + " (born in " + year + ")!!!");
//        }
//      }
    }

    return hasChild;
  }
```

5.4 クラス IRIS のサポートされているデータ型

わかりやすくするために、この章の以前のセクションの例では、常に Integer または String ノード値を使用していましたが、IRIS クラスでは以下のサポートされているデータ型に対してデータ型固有のメソッドも提供しています。

IRIS.set()

IRIS.Set() メソッドは、データ型 bool、byte[]、Single、Double、DateTime、Int16、Int32、Int64、String、IRISList、および System.IO.MemoryStream を実装するオブジェクトのインスタンスをサポートします。null 値は "" として格納されます。

クラス IRIS の数値向けのゲッター

以下の IRIS メソッドは、ノード値が数値であると想定し、適切な .NET 変数への変換を試みます：GetBool()、GetSingle()、GetDouble()、GetInt16()、GetInt32()、または GetInt64()。整数のノード値を指定すると、すべての数値メソッドは意味のある値を返します。整数のゲッターは、Single または Double の値を確実には取得できず、不正確な値または意味のない値を返す可能性があります。

クラス IRIS の String、byte[]、および IRISList 向けのゲッター

InterSystems IRIS データベースでは、String、byte[]、および IRISList のオブジェクトは、すべて文字列として格納され、元のデータ型についての情報は維持されません。IRIS GetString()、GetBytes()、および GetIRISList() のメソッドは、文字列データを取得して、それを希望の形式に強制しようとします。

文字列ゲッターはノード値が数値ではないと想定し、それを適切に変換しようと試みます。ターゲット・ノードに値がない、またはターゲット・ノードが存在しない場合は、null を返します。これらのメソッドはタイプ・チェックを実行しないため、ノード値が間違ったデータ型であっても通常は例外をスローしません。

クラス IRIS の .NET クラス向けのゲッター

IRIS クラスは、以下に示す一部の .NET クラス向けのゲッターもサポートします。

- GetDateTime() ― System.DateTime のインスタンスを取得します。
- GetObject() ― ターゲット・ノードの値を object として返します。
- GetBytes() ― System.IO.MemoryStream を実装するオブジェクトを取得するために使用できます。MemoryStream のインスタンスがノード値として設定されるときに byte[] として格納されます。GetBytes() を使用して値を取得し、その後返される byte [] 値で MemoryStream を初期化してください。

重要

ゲッター・メソッドは互換性のないデータ型をチェックしない

これらのメソッドは処理速度のために最適化されており、タイプ・チェックを実行しません。いずれかのメソッドが間違ったデータ型の値をフェッチしようとした場合に例外をスローする処理に、アプリケーションが依存しないようにしてください。例外がスローされる場合もありますが、メソッドが通知なしで失敗し、不正確な値または意味のない値が返される可能性の方が高くなります。

6

トランザクションとロック

Native SDK for .NET は、InterSystems IRIS トランザクション・モデルを使用するトランザクション・メソッドとロック・メソッドを提供します。これについて、以下のセクションで説明します。

- トランザクションの制御 ートランザクションの開始、入れ子、ロールバック、およびコミットの方法について説明します。
- 並行処理の制御 ー さまざまなロック・メソッドの使用法について説明します。

重要

Native SDK トランザクション・モデルと ADO.NET トランザクション・モデルを混在させない

Native SDK トランザクション・モデルを ADO.NET/SQL トランザクション・モデルと混在させないでください。

- トランザクション内で Native SDK コマンドのみを使用する場合は、常に Native SDK トランザクション・メソッドを使用する必要があります。
- トランザクション内で Native SDK コマンドと ADO.NET/SQL コマンドの組み合わせを使用する場合は、自動コミットをオフにした後、常に Native SDK トランザクション・メソッドを使用する必要があります。
- トランザクション内で ADO.NET/SQL コマンドのみを使用する場合は、常に SQL トランザクション・メソッドを使用することも、自動コミットをオフにした後、常に Native SDK トランザクション・メソッドを使用することもできます。
- 同じアプリケーションで両方のモデルを使用できますが、トランザクションが一方のモデルでまだ実行されている間に、トランザクションをもう一方のモデルで開始しないように注意してください。

6.1 トランザクションの制御

ここで説明したメソッドは、標準の ADO.NET/SQL トランザクション・モデルに代わるメソッドです。トランザクションおよび並行処理の制御用の Native SDK モデルは ObjectScript メソッドに基づいており、.NET モデルと互換性はありません。トランザクションに Native SDK メソッド呼び出しが含まれる場合は、Native SDK モデルを使用する必要があります。

ObjectScript トランザクション・モデルの詳細は、"ObjectScript の使用法" の "トランザクション処理" を参照してください。

Native SDK for .NET には、トランザクションを制御するために以下のメソッドが用意されています。

- **IRIS.**TCommit() ー 1 レベルのトランザクションをコミットします。
- **IRIS.**TStart() ー トランザクションを開始します (このトランザクションは入れ子になっている場合あり)。

- IRIS.GetTLevel() ー 現在のトランザクション・レベルを示す int 値 (トランザクション内でない場合は 0) を返します。
- IRIS..TRollback() ー セッション内の開いているトランザクションすべてをロールバックします。
- IRIS..TRollbackOne() ー 現在のレベルのトランザクションのみをロールバックします。入れ子になったトランザクションである場合、それより上のレベルのトランザクションはロールバックされません。

以下の例では、3 レベルの入れ子のトランザクションを開始し、各トランザクション・レベルに異なるノードの値を設定します。3 つのノードはすべて出力され、それらに値があることが証明されます。その後、この例では、2 番目と 3 番目のレベルがロールバックされ、最初のレベルがコミットされます。3 つのノードはすべて再び出力され、依然として値があるのは最初のノードのみであることが証明されます。

トランザクションの制御 : 3 レベルの入れ子トランザクションの使用法

```
        String globalName = "myGlobal";
        iris.TStart();

        // GetTLevel() is 1: create myGlobal(1) = "firstValue"
        iris.Set("firstValue", globalName, iris.GetTLevel());

        iris.TStart();
        // GetTLevel() is 2: create myGlobal(2) = "secondValue"
        iris.Set("secondValue", globalName, iris.GetTLevel());

        iris.TStart();
        // GetTLevel() is 3: create myGlobal(3) = "thirdValue"
        iris.Set("thirdValue", globalName, iris.GetTLevel());

        Console.WriteLine("Node values before rollback and commit:");
        for (int ii=1;ii<4;ii++) {
          Console.Write(globalName + "(" + ii + ") = ");
          if (iris.isDefined(globalName,ii) > 1) Console.WriteLine(iris.GetString(globalName,ii));
          else Console.WriteLine("<valueless>");
        }
      // prints: Node values before rollback and commit:
      //         myGlobal(1) = firstValue
      //         myGlobal(2) = secondValue
      //         myGlobal(3) = thirdValue

        iris.TRollbackOne();
        iris.TRollbackOne();  // roll back 2 levels to GetTLevel 1
        iris.TCommit();  // GetTLevel() after commit will be 0
        Console.WriteLine("Node values after the transaction is committed:");
        for (int ii=1;ii<4;ii++) {
          System.out.print(globalName + "(" + ii + ") = ");
          if (iris.isDefined(globalName,ii) > 1) Console.WriteLine(iris.GetString(globalName,ii));
          else Console.WriteLine("<valueless>");
        }
      // prints: Node values after the transaction is committed:
      //         myGlobal(1) = firstValue
      //         myGlobal(2) = <valueless>
      //         myGlobal(3) = <valueless>
```

6.2 並行処理の制御

並行処理の制御は、InterSystems IRIS などのマルチプロセス・システムのきわめて重要な機能です。この機能により、データの特定の要素をロックして、複数のプロセスが同じ要素を同時に変更することによって起きるデータ破損を防ぐことができます。Native SDK トランザクション・モデルは、ObjectScript コマンドに対応する一連のロック・メソッドを提供します。これらのメソッドは、ADO.NET/SQL トランザクション・モデルで使用することはできません (詳細は、この章の冒頭にある警告を参照)。

クラス IRIS の以下のメソッドを使用して、ロックを取得および解放します。どちらのメソッドも、ロックが共有であるか排他であるかを指定する lockMode 引数を取ります。

```
Lock(string lockMode, int timeout, string globalName, Object[] subscripts)
Unlock(string lockMode, string globalName, Object[] subscripts)
```

- **IRIS.**Lock() ー lockMode、timeout、globalName、および subscripts 引数を取り、ノードをロックします。lockMode 引数は、前に保持されたロックを解放するかどうかを指定します。このメソッドは、ロックを取得できない場合、事前に定義した間隔が経過するとタイムアウトします。
- **IRIS.**Unlock() ー lockMode、globalName、および subscripts 引数を取り、ノードのロックを解放します。

以下の引数値を使用できます。

- lockMode ー 共有ロックを表す **S** という文字、エスカレート・ロックを表す **E** という文字、または共有およびエスカレート・ロックを表す **SE** という文字の組み合わせ。既定値は空の文字列 (排他の非エスカレート) です。
- timeout ー ロックの取得を試行するときに待機する秒数。

注釈　管理ポータルを使用して、ロックを検証できます。**System Operation > Locks** に移動して、システムでロックされている項目のリストを表示します。

現在保持されているロックをすべて解放するには、2 つの方法があります。

- **IRIS.**ReleaseAllLocks() ー この接続で現在保持されているロックをすべて解放します。
- 接続オブジェクトの Close() メソッドが呼び出されると、すべてのロックおよびその他の接続リソースが解放されます。

Tipヒント　並行処理の制御に関する詳細な説明は、このドキュメントの対象ではありません。この内容に関する詳細は、以下のドキュメントと技術文書を参照してください。

- ″ObjectScript の使用法″ の “トランザクション処理” および “ロック管理”
- ″サーバ側プログラミングの入門ガイド″ の “ロックと並行処理の制御”
- ″ObjectScript リファレンス″ の “LOCK”

7

Native SDK for .NET のクイック・リファレンス

これは、InterSystems.Data.IRISClient.ADO 内の以下の拡張クラスで構成される InterSystems IRIS Native SDK for .NET のクイック・リファレンスです。

- クラス IRIS は、Native SDK の主要な機能を提供します。
- クラス IRISIterator は、グローバル配列を操作するためのメソッドを提供します。
- クラス IRISList は、インターシステムズの $LIST シリアライゼーションをサポートします。
- クラス IRISObject は、ゲートウェイ逆プロキシ・オブジェクトを操作するためのメソッドを提供します。

これらのクラスはすべて、InterSystems ADO.NET Managed Provider (InterSystems.Data.IRISClient.ADO) の一部です。これらのクラスは標準の .NET 接続を介してデータベースにアクセスし、特別な設定やインストール手順なしに使用できます。

注釈　このリファレンスは、このドキュメントの読者の利便性を目的としたものであり、Native SDK の最終的なリファレンスではありません。最も包括的な最新情報については、オンライン・クラス・ドキュメントを参照してください。

以下のセクションにリストされているすべてのメソッドは、いずれかの種類のエラーが発生すると例外をスローします。添え字引数 `Object[] args` は常に `params object[]` として定義されます。

7.1 クラス IRIS

クラス IRIS は InterSystems.Data.IRISClient.ADO (InterSystems ADO.NET Managed Provider) のメンバです。

IRIS にはパブリック・コンストラクタはありません。IRIS のインスタンスは、静的メソッド IRIS.CreateIRIS() を呼び出すことによって作成されます。

7.1.1 IRIS メソッドの詳細

ClassMethodBool()

ADO.IRIS.ClassMethodBool() は ObjectScript クラス・メソッドを呼び出して、0 個以上の引数を渡し、bool? のインスタンスを返します。

```
bool ClassMethodBool(string className, string methodName, params object[] args)
```

パラメータ：

- `className` ー 呼び出されるメソッドが属するクラスの完全修飾名。
- `methodName` ー クラス・メソッドの名前。
- `args` ー サポートされる型の 0 個以上の引数。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

ClassMethodBytes()

ADO.IRIS.ClassMethodBytes() は ObjectScript クラス・メソッドを呼び出して、0 個以上の引数を渡し、byte[] のインスタンスを返します。

```
byte[] ClassMethodBytes(string className, string methodName, params object[] args)
```

パラメータ：

- `className` ー 呼び出されるメソッドが属するクラスの完全修飾名。
- `methodName` ー クラス・メソッドの名前。
- `args` ー サポートされる型の 0 個以上の引数。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

ClassMethodDouble()

ADO.IRIS.ClassMethodDouble() は ObjectScript クラス・メソッドを呼び出して、0 個以上の引数を渡し、double? のインスタンスを返します。

```
double ClassMethodDouble(string className, string methodName, params object[] args)
```

パラメータ：

- `className` ー 呼び出されるメソッドが属するクラスの完全修飾名。
- `methodName` ー クラス・メソッドの名前。
- `args` ー サポートされる型の 0 個以上の引数。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

ClassMethodIRISList()

ADO.IRIS.ClassMethodIRISList() は ObjectScript クラス・メソッドを呼び出して、0 個以上の引数を渡し、IRISList のインスタンスを返します。

```
IRISList ClassMethodIRISList(string className, string methodName, params object[] args)
```

このメソッドは `newList=(IRISList) ClassMethodBytes(className, methodName, args)` と同等です。

パラメータ：

- `className` ー 呼び出されるメソッドが属するクラスの完全修飾名。
- `methodName` ー クラス・メソッドの名前。
- `args` ー サポートされる型の 0 個以上の引数。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

ClassMethodLong()

ADO.IRIS.ClassMethodLong() は ObjectScript クラス・メソッドを呼び出して、0 個以上の引数を渡し、long? のインスタンスを返します。

```
long ClassMethodLong(string className, string methodName, params object[] args)
```

パラメータ：

- `className` ― 呼び出されるメソッドが属するクラスの完全修飾名。
- `methodName` ― クラス・メソッドの名前。
- `args` ― サポートされる型の 0 個以上の引数。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

ClassMethodObject()

ADO.IRIS.ClassMethodObject() は ObjectScript クラス・メソッドを呼び出して、0 個以上の引数を渡し、object のインスタンスを返します。返されるオブジェクトが有効な OREF である場合 (例えば、%New() が呼び出された場合)、ClassMethodObject() は参照オブジェクトの逆プロキシ・オブジェクト (IRISObject のインスタンス) を生成して返します。詳細と例は、“.NET 逆プロキシ・オブジェクトの使用法”を参照してください。

```
object ClassMethodObject(string className, string methodName, params object[] args)
```

パラメータ：

- `className` ― 呼び出されるメソッドが属するクラスの完全修飾名。
- `methodName` ― クラス・メソッドの名前。
- `args` ― サポートされる型の 0 個以上の引数。

ClassMethodStatusCode()

ADO.IRIS.ClassMethodStatusCode() は、ObjectScript $Status オブジェクトを返すクラス・メソッドが、指定された引数で呼び出された場合にエラーをスローするかどうかをテストします。呼び出しが失敗した場合、ObjectScript $Status エラー・ステータス番号とメッセージを含む IRISException エラーがスローされます。

```
void ClassMethodStatusCode(string className, string methodName, params object[] args)
```

パラメータ：

- `className` ― 呼び出されるメソッドが属するクラスの完全修飾名。
- `methodName` ― クラス・メソッドの名前。
- `args` ― サポートされる型の 0 個以上の引数。

これは、ClassMethod[type] 呼び出しを使用している場合に例外を捕捉する間接的な方法です。呼び出しがエラーなしで実行された場合、このメソッドは何もせずに戻ります。つまり、指定した引数で安全に呼び出しを行うことができます。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

ClassMethodString()

ADO.IRIS.ClassMethodString() は ObjectScript クラス・メソッドを呼び出して、0 個以上の引数を渡し、string のインスタンスを返します。

```
string ClassMethodString(string className, string methodName, params object[] args)
```

パラメータ：

- `className` ― 呼び出されるメソッドが属するクラスの完全修飾名。
- `methodName` ― クラス・メソッドの名前。
- `args` ― サポートされる型の 0 個以上の引数。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

ClassMethodVoid()

ADO.IRIS.ClassMethodVoid() は、返り値のない ObjectScript クラス・メソッドを呼び出し、0 個以上の引数を渡します。

```
void ClassMethodVoid(string className, string methodName, params object[] args)
```

パラメータ：

- `className` ― 呼び出されるメソッドが属するクラスの完全修飾名。
- `methodName` ― クラス・メソッドの名前。
- `args` ― サポートされる型の 0 個以上の引数。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

Close()

ADO.IRIS.Close() は IRIS オブジェクトを閉じます。

```
void Close()
```

CreateIRIS() [static]

ADO.IRIS.CreateIRIS() は、指定された IRISConnection を使用する ADO.IRIS のインスタンスを返します。

```
static IRIS CreateIRIS(IRISADOConnection conn)
```

パラメータ：

- `conn` ― IRISConnection のインスタンス。

詳細と例は、“グローバル配列の概要”を参照してください。

FunctionBool()

ADO.IRIS.FunctionBool() は ObjectScript 関数を呼び出して、0 個以上の引数を渡し、bool? のインスタンスを返します。

```
bool FunctionBool(string functionName, string routineName, params object[] args)
```

パラメータ：

- `functionName` ― 呼び出す関数の名前。

- routineName — 関数を含むルーチンの名前。
- args — サポートされる型の 0 個以上の引数。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

FunctionBytes()

ADO.IRIS.FunctionBytes() は ObjectScript 関数を呼び出して、0 個以上の引数を渡し、byte[] のインスタンスを返します。

```
byte[] FunctionBytes(string functionName, string routineName, params object[] args)
```

パラメータ：

- functionName — 呼び出す関数の名前。
- routineName — 関数を含むルーチンの名前。
- args — サポートされる型の 0 個以上の引数。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

FunctionDouble()

ADO.IRIS.FunctionDouble() は ObjectScript 関数を呼び出して、0 個以上の引数を渡し、double? のインスタンスを返します。

```
double FunctionDouble(string functionName, string routineName, params object[] args)
```

パラメータ：

- functionName — 呼び出す関数の名前。
- routineName — 関数を含むルーチンの名前。
- args — サポートされる型の 0 個以上の引数。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

FunctionInt()

ADO.IRIS.FunctionInt() は ObjectScript 関数を呼び出して、0 個以上の引数を渡し、long? のインスタンスを返します。

```
long FunctionInt(string functionName, string routineName, params object[] args)
```

パラメータ：

- functionName — 呼び出す関数の名前。
- routineName — 関数を含むルーチンの名前。
- args — サポートされる型の 0 個以上の引数。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

FunctionIRISList()

ADO.IRIS.FunctionIRISList() は ObjectScript 関数を呼び出して、0 個以上の引数を渡し、IRISList のインスタンスを返します。

```
IRISList FunctionIRISList(string functionName, string routineName, params object[] args)
```

この関数は `newList=(IRISList) FunctionBytes(functionName, routineName, args)` と同等です。

パラメータ：

- `functionName` ― 呼び出す関数の名前。
- `routineName` ― 関数を含むルーチンの名前。
- `args` ― サポートされる型の 0 個以上の引数。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

FunctionLong()

ADO.IRIS.FunctionLong() は ObjectScript 関数を呼び出して、0 個以上の引数を渡し、Long のインスタンスを返します。

```
long FunctionLong(string functionName, string routineName, params object[] args)
```

パラメータ：

- `functionName` ― 呼び出す関数の名前。
- `routineName` ― 関数を含むルーチンの名前。
- `args` ― サポートされる型の 0 個以上の引数。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

FunctionObject()

ADO.IRIS.FunctionObject() は ObjectScript 関数を呼び出して、0 個以上の引数を渡し、object のインスタンスを返します。返されるオブジェクトが有効な OREF である場合、FunctionObject() は参照オブジェクトの逆プロキシ・オブジェクト (IRISObject のインスタンス) を生成して返します。詳細と例は、“.NET 逆プロキシ・オブジェクトの使用法”を参照してください。

```
object FunctionObject(string functionName, string routineName, params object[] args)
```

パラメータ：

- `functionName` ― 呼び出す関数の名前。
- `routineName` ― 関数を含むルーチンの名前。
- `args` ― サポートされる型の 0 個以上の引数。

関連情報については、“クラス IRIS のサポートされているデータ型”を参照してください。

FunctionString()

ADO.IRIS.FunctionString() は ObjectScript 関数を呼び出して、0 個以上の引数を渡し、string のインスタンスを返します。

```
string FunctionString(string functionName, string routineName, params object[] args)
```

パラメータ：

- `functionName` — 呼び出す関数の名前。
- `routineName` — 関数を含むルーチンの名前。
- `args` — サポートされる型の 0 個以上の引数。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

GetAPIVersion() [static]

ADO.IRIS.GetAPIVersion() は、Native SDK バージョン文字列を返します。

```
static String GetAPIVersion()
```

GetBool()

ADO.IRIS.GetBool() は、グローバルの値を bool? として取得します（ノードが存在しない場合は `null` を返します）。ノード値が空の文字列である場合は `false` を返します。

```
bool GetBool(string globalName, params object[] subscripts)
```

パラメータ：

- `globalName` — グローバル名。
- `subscripts` — ターゲット・ノードを指定する 0 個以上の添え字。

関連情報については、“クラス IRIS のサポートされているデータ型”を参照してください。

GetBytes()

ADO.IRIS.GetBytes() は、グローバルの値を byte[] として取得します（ノードが存在しない場合は `null` を返します）。

```
byte[] GetBytes(string globalName, params object[] subscripts)
```

パラメータ：

- `globalName` — グローバル名。
- `subscripts` — ターゲット・ノードを指定する 0 個以上の添え字。

関連情報については、“クラス IRIS のサポートされているデータ型”を参照してください。

GetDateTime()

ADO.IRIS.GetDateTime() は、グローバルの値を DateTime? として取得します（ノードが存在しない場合は `null` を返します）。

```
DateTime GetDateTime(string globalName, params object[] subscripts)
```

パラメータ：

- `globalName` – グローバル名。
- `subscripts` – ターゲット・ノードを指定する 0 個以上の添え字。

関連情報については、“クラス IRIS のサポートされているデータ型”を参照してください。

GetDouble()

ADO.IRIS.GetDouble() は、グローバルの値を double? として取得します (ノードが存在しない場合は `null` を返します)。ノード値が空の文字列である場合は `0.0` を返します。

```
Double GetDouble(string globalName, params object[] subscripts)
```

パラメータ :

- `globalName` – グローバル名。
- `subscripts` – ターゲット・ノードを指定する 0 個以上の添え字。

関連情報については、“クラス IRIS のサポートされているデータ型”を参照してください。

GetInt16()

ADO.IRIS.GetInt16() は、グローバルの値を Int16? として取得します (ノードが存在しない場合は `null` を返します)。ノード値が空の文字列である場合は 0 を返します。

```
Int16 GetInt16(string globalName, params object[] subscripts)
```

パラメータ :

- `globalName` – グローバル名。
- `subscripts` – ターゲット・ノードを指定する 0 個以上の添え字。

関連情報については、“クラス IRIS のサポートされているデータ型”を参照してください。

GetInt32()

ADO.IRIS.GetInt32() は、グローバルの値を Int32? として取得します (ノードが存在しない場合は `null` を返します)。ノード値が空の文字列である場合は 0 を返します。

```
Int32 GetInt32(string globalName, params object[] subscripts)
```

パラメータ :

- `globalName` – グローバル名。
- `subscripts` – ターゲット・ノードを指定する 0 個以上の添え字。

関連情報については、“クラス IRIS のサポートされているデータ型”を参照してください。

GetInt64()

ADO.IRIS.GetInt64() は、グローバルの値を Int64? として取得します (ノードが存在しない場合は `null` を返します)。ノード値が空の文字列である場合は 0 を返します。

```
Int64 GetInt64(string globalName, params object[] subscripts)
```

パラメータ :

- `globalName` — グローバル名。
- `subscripts` — ターゲット・ノードを指定する 0 個以上の添え字。

関連情報については、“クラス IRIS のサポートされているデータ型”を参照してください。

GetIRISIterator()

ADO.IRIS.GetIRISIterator() は、検索方向が FORWARD に設定された、指定されたノードの IRISIterator オブジェクトを返します (“クラス IRISIterator”を参照)。詳細および例は、“一連の子ノードにわたる反復”を参照してください。

```
IRISIterator GetIRISIterator(string globalName, params object[] subscripts)
```

パラメータ：

- `globalName` — グローバル名。
- `subscripts` — ターゲット・ノードを指定する 0 個以上の添え字。

関連情報については、“クラス IRIS のサポートされているデータ型”を参照してください。

GetIRISList()

ADO.IRIS.GetIRISList() は、グローバルの値を IRISList として取得します (ノードが存在しない場合は `null` を返します)。

```
IRISList GetIRISList(String globalName, params object[] subscripts)
```

これは `newList=(IRISList) GetBytes(globalName, subscripts)` を呼び出すことと同等です。

パラメータ：

- `globalName` — グローバル名。
- `subscripts` — ターゲット・ノードを指定する 0 個以上の添え字。

関連情報については、“クラス IRIS のサポートされているデータ型”を参照してください。

GetIRISReverseIterator()

ADO.IRIS.GetIRISReverseIterator() は、検索方向が BACKWARD に設定された、指定されたノードの IRISIterator オブジェクトを返します (“クラス IRISIterator”を参照)。詳細および例は、“一連の子ノードにわたる反復”を参照してください。

```
IRISIterator GetIRISReverseIterator(string globalName, params object[] subscripts)
```

パラメータ：

- `globalName` — グローバル名。
- `subscripts` — ターゲット・ノードを指定する 0 個以上の添え字。

関連情報については、“クラス IRIS のサポートされているデータ型”を参照してください。

GetObject()

ADO.IRIS.GetObject() は、グローバルの値を Object として取得します (ノードが存在しない場合は null を返します)。詳細と例は、“.NET 逆プロキシ・オブジェクトの使用法”を参照してください。

```
object GetObject(string globalName, params object[] subscripts)
```

パラメータ :

- globalName – グローバル名。
- subscripts – ターゲット・ノードを指定する 0 個以上の添え字。

関連情報については、“クラス IRIS のサポートされているデータ型”を参照してください。

GetServerVersion()

ADO.IRIS.GetServerVersion() は、現在の接続のサーバ・バージョン文字列を返します。これは、ObjectScript で $system.Version.GetVersion() を呼び出すことと同じです。

```
String GetServerVersion()
```

GetSingle()

ADO.IRIS.GetSingle() は、グローバルの値を Single? (Nullable<float>) として取得します。または、ノードが存在しない場合は null を返します。ノード値が空の文字列である場合は 0.0 を返します。

```
Single GetSingle(string globalName, params object[] subscripts)
```

パラメータ :

- globalName – グローバル名。
- subscripts – ターゲット・ノードを指定する 0 個以上の添え字。

関連情報については、“クラス IRIS のサポートされているデータ型”を参照してください。

GetString()

ADO.IRIS.GetString() は、グローバルの値を string として取得します (ノードが存在しない場合は null を返します)。

空の文字列および null 値では変換が必要となります。.NET の空の文字列 "" は、ObjectScript では null 文字 $CHAR(0) に変換されます。.NET の null は、ObjectScript では空の文字列に変換されます。この変換は、.NET によるこれらの値の処理方法と一致しています。

```
string GetString(string globalName, params object[] subscripts)
```

パラメータ :

- globalName – グローバル名。
- subscripts – ターゲット・ノードを指定する 0 個以上の添え字。

関連情報については、“クラス IRIS のサポートされているデータ型”を参照してください。

GetTLevel()

ADO.IRIS.GetTLevel() は、入れ子になった現在の Native SDK トランザクションのレベルを取得します。開いているトランザクションが 1 つのみである場合は 1 を返します。開いているトランザクションがない場合は 0 を返します。これは、$TLEVEL 特殊変数の値のフェッチと同じです。

```
int GetTLevel()
```

このメソッドは Native SDK トランザクション・モデルを使用し、ADO.NET/SQL トランザクション・メソッドとは互換性がありません。この 2 つのトランザクション・モデルを混在させないでください。詳細と例は、“トランザクションとロック”を参照してください。

Increment()

ADO.IRIS.Increment() は、渡された値を使用して、指定されたグローバルをインクリメントします。指定されたアドレスにノードがない場合は、`value` を値として新しいノードが作成されます。null 値は 0 として解釈されます。グローバル・ノードの新しい値を返します。詳細と例は、“ノードの作成、アクセス、および削除”を参照してください。

```
long Increment(long? value, string globalName, params object[] subscripts)
```

パラメータ：

- `value` ― このノードを設定する long 値 (`null` 値の場合、グローバルは `0` に設定されます)。
- `globalName` ― グローバル名。
- `subscripts` ― ターゲット・ノードを指定する 0 個以上の添え字。

IsDefined()

ADO.IRIS.IsDefined() は、指定したノードが存在するかどうか、および値が含まれるかどうかを示す値を返します。詳細と例は、“子ノードおよびノード値のテスト”を参照してください。

```
int IsDefined(string globalName, params object[] subscripts)
```

パラメータ：

- `globalName` ― グローバル名。
- `subscripts` ― ターゲット・ノードを指定する 0 個以上の添え字。

返り値：

- `0` ― 指定したノードは存在しません。
- `1` ― ノードは存在し、値があります。
- `10` ― ノードに値はありませんが、サブノードがあります。
- `11` ― ノードには値とサブノードの両方があります。

Kill()

ADO.IRIS.Kill() は、下位ノードを含め、グローバル・ノードを削除します。詳細と例は、“ノードの作成、アクセス、および削除”を参照してください。

```
void Kill(string globalName, params object[] subscripts)
```

パラメータ：

- globalName – グローバル名。
- subscripts – ターゲット・ノードを指定する 0 個以上の添え字。

Lock()

ADO.IRIS.Lock() は、Native SDK トランザクションのグローバルをロックし、成功すると true を返します。このメソッドは増分ロックを実行し、ObjectScript でも提供されるロック前の暗黙的なアンロック機能は実行しません。

```
bool Lock(string lockMode, int timeout, string globalName, params object[] subscripts)
```

パラメータ：

- lockMode – 共有ロックの場合は文字 S、エスカレート・ロックの場合は E、両方の場合は SE。既定値は空の文字列 (排他の非エスカレート) です。
- timeout – ロックの取得を試行するときに待機する秒数。
- globalName – グローバル名。
- subscripts – ターゲット・ノードを指定する 0 個以上の添え字。

このメソッドは Native SDK トランザクション・モデルを使用し、ADO.NET/SQL トランザクション・メソッドとは互換性がありません。この 2 つのトランザクション・モデルを混在させないでください。詳細と例は、“トランザクションとロック”を参照してください。

Procedure()

ADO.IRIS.Procedure() はプロシージャを呼び出し、0 個以上の引数を渡します。値は返しません。

```
void Procedure(string procedureName, string routineName, params object[] args)
```

パラメータ：

- procedureName – 呼び出すプロシージャの名前。
- routineName – プロシージャを含むルーチンの名前。
- args – サポートされる型の 0 個以上の引数。

詳細と例は、“ObjectScript メソッドおよび関数の呼び出し”を参照してください。

ReleaseAllLocks()

ADO.IRIS.ReleaseAllLocks() は、セッションに関連付けられたすべてのロックを解放する Native SDK トランザクション・メソッドです。

```
void ReleaseAllLocks()
```

このメソッドは Native SDK トランザクション・モデルを使用し、ADO.NET/SQL トランザクション・メソッドとは互換性がありません。この 2 つのトランザクション・モデルを混在させないでください。詳細と例は、“トランザクションとロック”を参照してください。

Set()

ADO.IRIS.Set() は、現在のノードをサポートされているデータ型の値に設定します (値が null の場合は "")。指定されたノード・アドレスにノードがない場合は、指定した値で新しいノードが作成されます。詳細は、“ノードの作成、アクセス、および削除”を参照してください。

```
void Set(bool? value, string globalName, params object[] subscripts)
void Set(byte[] value, string globalName, params object[] subscripts)
void Set(DateTime? value, string globalName, params object[] subscripts)
void Set(Double? value, string globalName, params object[] subscripts)
void Set(Int16? value, string globalName, params object[] subscripts)
void Set(Int32? value, string globalName, params object[] subscripts)
void Set(Int64? value, string globalName, params object[] subscripts)
void Set(IRISList value, String globalName, params object[] subscripts)
void Set(object value, string globalName, params object[] subscripts)
void Set(Single? value, string globalName, params object[] subscripts)
void Set(string value, string globalName, params object[] subscripts)
void Set(System.IO.MemoryStream value, string globalName, params object[] subscripts)
```

パラメータ：

- value — サポートされているデータ型の値 (null 値の場合、グローバルは "" に設定されます)。
- globalName — グローバル名。
- subscripts — ターゲット・ノードを指定する 0 個以上の添え字。

関連情報については、“クラス IRIS のサポートされているデータ型”を参照してください。

特定のデータ型に関するメモ

以下のデータ型には追加機能があります。

- string — 空の文字列および null 値では変換が必要となります。.NET の空の文字列 "" は、ObjectScript では null 文字 $CHAR(0) に変換されます。.NET の null は、ObjectScript では空の文字列に変換されます。この変換は、.NET によるこれらの値の処理方法と一致しています。
- System.IO.MemoryStream — MemoryStream を実装するオブジェクトのインスタンスが、グローバル値として設定されるときに byte[] として格納されます。.NET の null は、ObjectScript では空の文字列に変換されます。getBytes() を使用して値を取得し、その後返される byte [] 値で MemoryStream を初期化してください。

TCommit()

ADO.IRIS.TCommit() は現在の Native SDK トランザクションをコミットします。

```
void TCommit()
```

このメソッドは Native SDK トランザクション・モデルを使用し、ADO.NET/SQL トランザクション・メソッドとは互換性がありません。この 2 つのトランザクション・モデルを混在させないでください。詳細と例は、“トランザクションとロック”を参照してください。

TRollback()

ADO.IRIS.TRollback() は、セッション内の開いている Native SDK トランザクションすべてをロールバックします。

```
void TRollback()
```

このメソッドは Native SDK トランザクション・モデルを使用し、ADO.NET/SQL トランザクション・メソッドとは互換性がありません。この 2 つのトランザクション・モデルを混在させないでください。詳細と例は、“トランザクションとロック”を参照してください。

TRollbackOne()

ADO.IRIS.TRollbackOne() は、現在のレベルの Native SDK トランザクションのみをロールバックします。入れ子になったトランザクションである場合、それより上のレベルのトランザクションはロールバックされません。

```
void TRollbackOne()
```

このメソッドは Native SDK トランザクション・モデルを使用し、ADO.NET/SQL トランザクション・メソッドとは互換性がありません。この 2 つのトランザクション・モデルを混在させないでください。詳細と例は、“トランザクションとロック”を参照してください。

TStart()

ADO.IRIS.TStart() は Native SDK トランザクションを開始します (または開きます)。

```
void TStart()
```

このメソッドは Native SDK トランザクション・モデルを使用し、ADO.NET/SQL トランザクション・メソッドとは互換性がありません。この 2 つのトランザクション・モデルを混在させないでください。詳細と例は、“トランザクションとロック”を参照してください。

Unlock()

ADO.IRIS.Unlock() は Native SDK トランザクション内のグローバルをアンロックします。このメソッドは増分アンロックを実行し、ObjectScript でも提供されるロック前の暗黙的なアンロック機能は実行しません。

```
void Unlock(string lockMode, string globalName, params object[] subscripts)
```

パラメータ：

- `lockMode` ― 共有ロックの場合は文字 S、エスカレート・ロックの場合は E、両方の場合は SE。既定値は空の文字列 (排他の非エスカレート) です。
- `globalName` ― グローバル名。
- `subscripts` ― ターゲット・ノードを指定する 0 個以上の添え字。

このメソッドは Native SDK トランザクション・モデルを使用し、ADO.NET/SQL トランザクション・メソッドとは互換性がありません。この 2 つのトランザクション・モデルを混在させないでください。詳細と例は、“トランザクションとロック”を参照してください。

7.2 クラス IRISIterator

クラス IRISIterator は InterSystems.Data.IRISClient.ADO (InterSystems ADO.NET Managed Provider) のメンバです。

IRISIterator のインスタンスは、以下のいずれかの IRIS メソッドを呼び出すことで作成されます。

- ADO.IRIS.GetIRISIterator() ― 順方向の反復処理に設定された IRISIterator インスタンスを返します。
- ADO.IRIS.GetIRISReverseIterator() ― 逆方向の反復処理に設定された IRISIterator インスタンスを返します。

詳細および例は、“一連の子ノードにわたる反復”を参照してください。

このクラスは必須の IEnumerator メソッド System.Collections.IEnumerator.GetEnumerator() を実装します。これは、`foreach` ループで反復子が使用される場合に呼び出されます。

以下のセクションにリストされているすべてのメソッドは、いずれかの種類のエラーが発生すると例外をスローします。

7.2.1 IRISIterator メソッドとプロパティの詳細

プロパティ Current

ADO.IRISIterator.Current は、現在の反復子位置にあるノードの値を取得します。`foreach` ループでは、この値は現在のループ変数にも割り当てられます。詳細と例は、“条件付きループでの反復処理” を参照してください。

```
object Current [get]
```

プロパティ CurrentSubscript

ADO.IRISIterator.CurrentSubscript は、現在の反復子位置にあるノードの最下位の添え字を取得します。例えば、反復子がノード ^myGlobal(23,"somenode") を指す場合、返り値は `"somenode"` になります。詳細と例は、“条件付きループでの反復処理” を参照してください。

```
object CurrentSubscript [get]
```

MoveNext()

ADO.IRISIterator.MoveNext() は、System.Collections.IEnumerator を実装します。次の値が取得された場合は `true` を返し、これ以上値がない場合は `false` を返します。詳細と例は、“条件付きループでの反復処理” を参照してください。

```
bool MoveNext()
```

Reset()

ADO.IRISIterator.Reset() を `foreach` ループが完了した後に呼び出して、反復子を開始位置にリセットし、再び使用できるようにすることができます。詳細と例は、“条件付きループでの反復処理” を参照してください。

```
void Reset()
```

StartFrom()

ADO.IRISIterator.StartFrom() は、反復子の開始位置を指定の添え字に設定します。添え字は任意の開始ポイントであり、既存のノードを指定する必要はありません。詳細および例は、“一連の子ノードにわたる反復” を参照してください。

```
void  StartFrom(Object subscript)
```

7.3 クラス IRISList

クラス IRISList は InterSystems.Data.IRISClient.ADO (InterSystems ADO.NET Managed Provider) のメンバです。このクラスは、インターシステムズの $LIST シリアライゼーションのための .NET インタフェースを実装します。IRISList コンストラクタ (以下のセクションで説明) に加えて、ADO.IRIS のメソッド classMethodIRISList()、functionIRISList()、getIRISList() によっても返されます。

7.3.1 IRISList コンストラクタ

ADO.IRISList.IRISList() コンストラクタのシグニチャを以下に示します。

```
public IRISList()
public IRISList(IRISList list)
public IRISList(byte[] buffer, int length)
```

パラメータ：

- list – コピーする IRISList のインスタンス
- buffer – 割り当てるバッファ
- length – 割り当てる初期バッファ・サイズ

IRISList インスタンスは以下の方法で作成できます。

空の IRISList を作成する

```
IRISList list = new IRISList();
```

別の IRISList のコピーを作成する

引数 list によって指定された IRISList インスタンスのコピーを作成します。

```
IRISList listcopy = new IRISList(myOtherList)
```

バイト配列から IRISList インスタンスを作成する

IRIS.GetBytes() から返されたものなど、$LIST 形式のバイト配列からインスタンスを作成します。コンストラクタは、サイズ length の buffer を取ります。

```
byte[] listBuffer = myIris.GetBytes("myGlobal",1);
IRISList listFromByte = new IRISList(listBuffer, listBuffer.length);
```

返されるリストは、リストへの変更がバッファに表示されてサイズ変更が必要になるまで、このバッファ (コピーではなく) を使用します。

7.3.2 IRISList メソッドの詳細

Add()

ADO.IRISList.Add() は、任意のサポートされる型の Object を IRISList の末尾に追加します。value が IRISList である場合、単一の要素として追加されます (IRISList インスタンスは連結されません)。

```
void Add(Object value)
```

パラメータ：

- value – Object の値です。Int16、Int32、Int64、bool、Single、Double、string、byte[]、IRISList の型がサポートされます。

IRISList 要素の追加

Add() は、IRISList インスタンスを常に単一のオブジェクトとして追加します。ただし、IRISList.ToArray() を使用して、IRISList を配列に変換し、その後各要素上で別個に Add() を呼び出すことができます。

AddRange()

ADO.IRISList.AddRange() は、コレクションの各要素を、コレクションの反復子から返されたとおりにリストの末尾に追加します。要素がサポートされている型の 1 つではない場合、例外がスローされます。

```
void AddRange(System.Collections.IList list)
```

パラメータ：

- `list` — Int16、Int32、Int64、bool、Single、Double、string、byte[]、IRISList の型の要素のみを含むコレクションです。

関連情報については、“クラス IRIS のサポートされているデータ型” を参照してください。

Clear()

ADO.IRISList.Clear() は、リストからすべての要素を削除することでリストをリセットします。

```
void Clear()
```

Count()

ADO.IRISList.Count() はリストを反復処理して、見つかった要素の数を返します。

```
int Count()
```

DateToHorolog() [static]

ADO.IRISList.DateToHorolog() は、DateTime 値の date プロパティを、$Horolog 文字列の day フィールドを表す int に変換します。"HorologToDate()" も参照してください。

```
static int DateToHorolog(DateTime value)
```

パラメータ：

- `value` — 変換する DateTime の値。

DateToPosix() [static]

ADO.IRISList.DateToPosix() は、Time オブジェクトを $Horolog 文字列の time フィールドに変換します。

```
static long DateToPosix(DateTime Value)
```

パラメータ：

- `Value` — 変換する DateTime の値。

Equals()

ADO.IRISList.Equals() は、指定された list を IRISList のこのインスタンスと比較し、それらが同一である場合に true を返します。同じであるためには、両方のリストに、同じ数の要素が同じ順序で、シリアライズされた同一の値で含まれている必要があります。

```
override bool Equals(Object list)
```

パラメータ：

- `list` — 比較する IRISList のインスタンス。

Get()

ADO.IRISList.Get() は、index にある要素を Object として返します。インデックスが 1 未満であるか、リストの末尾を越えている場合、IndexOutOfRangeException をスローします。

```
Object Get(int index)
```

パラメータ：

- index — 取得するリスト要素を指定する整数。

GetList()

ADO.IRISList.GetList() は、index にある要素を IRISList として取得します。インデックスが 1 未満であるか、リストの末尾を越えている場合、IndexOutOfRangeException をスローします。

```
IRISList GetList(int index)
```

パラメータ：

- index — 返すリスト要素を指定する整数。

インデックスが 1 未満であるか、リストの末尾を越えている場合、IndexOutOfRangeException をスローします。

HorologToDate() [static]

ADO.IRISList.HorologToDate() は、$Horolog 文字列の day フィールドを DateTime 値に変換します。"DateToHorolog()" も参照してください。

```
static DateTime HorologToDate(int HorologValue)
```

パラメータ：

- HorologValue — $Horolog 文字列の day フィールドを表す int。

HorologToTime() [static]

ADO.IRISList.HorologToTime() は、$Horolog 文字列の time フィールドを TimeSpan 値に変換します。"TimeToHorolog()" も参照してください。

```
static TimeSpan HorologToTime(int HorologValue)
```

パラメータ：

- HorologValue — $Horolog 文字列の time フィールドを表す int。date フィールドはゼロに設定される (エポックに設定される) ため、0L から 86399999L の範囲外のミリ秒値を持つ TimeSpan は、ラウンドトリップしません。

PosixToDate() [static]

ADO.IRISList.PosixToDate() は、%Library.PosixTime の値を DateTime に変換します。"DateToPosix()" も参照してください。

```
static DateTime PosixToDate(long PosixValue)
```

パラメータ：

- PosixValue — %PosixTime の値 (64 ビットの符号付き整数)。

Remove()

ADO.IRISList.Remove() は、index にある要素をリストから削除します。要素が存在していて削除された場合は true を返し、そうでない場合は false を返します。このメソッドは、通常、リストを再割り当てする必要があるため、高いコストがかかる可能性があります。

```
bool Remove(int index)
```

パラメータ：

- index — 削除するリスト要素を指定する整数。

Set()

ADO.IRISList.Set() は、index にあるリスト要素を value で置き換えます。value が配列である場合、各配列要素が index からリストに挿入されます。index の後にある既存のリスト要素は、新しい値の場所を確保するために移動されます。index がリストの末尾より後にある場合、value は index に格納され、リストはその位置まで NULL で埋められます。

```
void Set(int index, Object value)
void Set(int index, object[] value)
```

index が 1 未満の場合は、IndexOutOfRangeException をスローします。すべての要素は、Int16、Int32、Int64、bool、Single、Double、string、byte[]、IRISList の型である必要があります。

パラメータ：

- index — 設定または置換するリスト要素を指定する整数。
- value — index に挿入する Object 値または object[] 配列。

Size()

ADO.IRISList.Size() は、この IRISList のシリアライズされた値のバイト長を返します。

```
int Size()
```

SubList()

ADO.IRISList.SubList() は、閉範囲 [from, to] 内の要素を含む新しい IRISList を返します。from が Count() より大きいか、to が from.より小さい場合、IndexOutOfRangeException をスローします。

```
IRISList SubList(int from, int to)
```

パラメータ：

- from — 新しいリストに追加する最初の要素のインデックス。
- to — 新しいリストに追加する最後の要素のインデックス。

TimeToHorolog() [static]

ADO.IRISList.TimeToHorolog() は、DateTime の値を $Horolog の値の time フィールドに変換します。"HorologToTime()" も参照してください。

```
static int TimeToHorolog(DateTime value)
```

パラメータ：

- value — 変換する DateTime の値。

ToArray()

ADO.IRISList.ToArray() は、このリスト内のすべての要素を含む特定の型の配列を返します。リストが空の場合、長さがゼロの配列が返されます。

重要：このメソッドは、すべての要素を強制的に同じ型にできる場合にのみ使用してください。

```
Object[] ToArray()
```

ToList()

ADO.IRISList.ToList() は、この IRISList 内のすべての要素を含む特定の型の List を返します。リストが空の場合、長さがゼロの List が返されます。

```
List<Object> ToList()
```

ToString()

ADO.IRISList.ToString() は、リストの表示可能な表現を返します。

```
override string ToString()
```

一部の要素タイプの表現方法は、ObjectScript における表現方法と異なります。

- 空のリスト (ObjectScript の `""`) は、`"$lb()"` と表示されます。
- 空の要素 (`$lb()=$c(1)`) は `"null"` と表示されます。
- 文字列は引用符で囲まれません。
- 有効桁数 16 桁の倍精度形式

7.4 クラス IRISObject

クラス IRISObject は InterSystems.Data.IRISClient.ADO (InterSystems ADO.NET Managed Provider) のメンバです。ゲートウェイ逆プロキシ・オブジェクトを操作するためのメソッドを提供します (詳細は、".NET 逆プロキシ・オブジェクトの使用法" を参照)。

IRISObject にはパブリック・コンストラクタはありません。IRISObject のインスタンスは、以下のいずれかの IRIS メソッドを呼び出すことで作成できます。

- ADO.IRIS.ClassMethodObject()
- ADO.IRIS.FunctionObject()

呼び出されたメソッドまたは関数が、有効な OREF であるオブジェクトを返した場合、参照オブジェクトの逆プロキシ・オブジェクト (IRISObject のインスタンス) が生成されて返されます。例えば、ClassMethodObject() は、%New() によって作成されたオブジェクトのプロキシ・オブジェクトを返します。

詳細と例は、".NET 逆プロキシ・オブジェクトの使用法" を参照してください。

7.4.1 IRISObject メソッドの詳細

Dispose()

ADO.IRISObject.Dispose() は、IRISObject のこのインスタンスを解放します。

```
void Dispose()
```

Get()

ADO.IRISObject.Get() は、プロキシ・オブジェクトのプロパティ値を object のインスタンスとして返します。

```
object Get(string propertyName)
```

パラメータ：

- propertyName — 返されるプロパティの名前。

GetBool()

ADO.IRISObject.GetBool() は、プロキシ・オブジェクトのプロパティ値を bool のインスタンスとして返します。

```
bool GetBool(string propertyName)
```

パラメータ：

- propertyName — 返されるプロパティの名前。

関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

GetBytes()

ADO.IRISObject.GetBytes() は、プロキシ・オブジェクトのプロパティ値を byte[] のインスタンスとして返します。

```
byte[] GetBytes(string propertyName)
```

パラメータ：

- propertyName — 返されるプロパティの名前。

関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

GetDouble()

ADO.IRISObject.GetDouble() は、プロキシ・オブジェクトのプロパティ値を double のインスタンスとして返します。

```
double GetDouble(string propertyName)
```

パラメータ：

- propertyName — 返されるプロパティの名前。

関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

GetIRISList()

ADO.IRISObject.GetIRISList() は、プロキシ・オブジェクトのプロパティ値を IRISList のインスタンスとして返します。

```
IRISList  getIRISList(String propertyName)
```

パラメータ：

- propertyName — 返されるプロパティの名前。

関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

GetLong()

ADO.IRISObject.GetLong() は、プロキシ・オブジェクトのプロパティ値を long のインスタンスとして返します。

```
Long  getLong(String propertyName)
```

パラメータ：

- `propertyName` ― 返されるプロパティの名前。

関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

GetObject()

ADO.IRISObject.GetObject() は、プロキシ・オブジェクトのプロパティ値を object のインスタンスとして返します。

```
Object  getObject(String propertyName)
```

パラメータ：

- `propertyName` ― 返されるプロパティの名前。

関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

GetString()

ADO.IRISObject.GetString() は、プロキシ・オブジェクトのプロパティ値を string のインスタンスとして返します。

```
String  getString(String propertyName)
```

パラメータ：

- `propertyName` ― 返されるプロパティの名前。

関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

Invoke()

ADO.IRISObject.Invoke() は、オブジェクトのインスタンス・メソッドを呼び出して、値を object として返します。

```
object Invoke(string methodName, params object[] args)
```

パラメータ：

- `methodName` ― 呼び出されるインスタンス・メソッドの名前。
- `args` ― サポートされる型の 0 個以上の引数。

引数は、int?、short?、string、long?、double?、float?、byte[]、bool?、DateTime?、IRISList?、または IRISObject のいずれかの型にできます。接続が双方向の場合、任意の .NET オブジェクトを引数として使用できます。

有効な引数の型と関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

InvokeBoolean()

ADO.IRISObject.InvokeBool() は、オブジェクトのインスタンス・メソッドを呼び出して、値を bool として返します。

```
Boolean  invokeBoolean(String methodName, Object... args)
```

パラメータ：

- `methodName` — 呼び出されるインスタンス・メソッドの名前。
- `args` — サポートされる型の 0 個以上の引数。

有効な引数の型と関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

InvokeBytes()

ADO.IRISObject.InvokeBytes() は、オブジェクトのインスタンス・メソッドを呼び出して、値を byte[] として返します。

```
byte[]  invokeBytes(String methodName, Object... args)
```

パラメータ：

- `methodName` — 呼び出されるインスタンス・メソッドの名前。
- `args` — サポートされる型の 0 個以上の引数。

有効な引数の型と関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

InvokeDouble()

ADO.IRISObject.InvokeDouble() は、オブジェクトのインスタンス・メソッドを呼び出して、値を double として返します。

```
Double  invokeDouble(String methodName, Object... args)
```

パラメータ：

- `methodName` — 呼び出されるインスタンス・メソッドの名前。
- `args` — サポートされる型の 0 個以上の引数。

有効な引数の型と関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

InvokeIRISList()

ADO.IRISObject.InvokeIRISList() は、オブジェクトのインスタンス・メソッドを呼び出して、値を IRISList として返します。

```
IRISList  invokeIRISList(String methodName, Object... args)
```

パラメータ：

- `methodName` — 呼び出されるインスタンス・メソッドの名前。
- `args` — サポートされる型の 0 個以上の引数。

有効な引数の型と関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

InvokeLong()

ADO.IRISObject.InvokeLong() は、オブジェクトのインスタンス・メソッドを呼び出して、値を long として返します。

```
Long  invokeLong(String methodName, Object... args)
```

パラメータ：

- `methodName` — 呼び出されるインスタンス・メソッドの名前。

- args – サポートされる型の 0 個以上の引数。

有効な引数の型と関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

InvokeObject()

ADO.IRISObject.InvokeObject() は、オブジェクトのインスタンス・メソッドを呼び出して、値を object として返します。

```
Object  invokeObject(String methodName, Object... args)
```

パラメータ：

- methodName – 呼び出されるインスタンス・メソッドの名前。
- args – サポートされる型の 0 個以上の引数。

有効な引数の型と関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

InvokeStatusCode()

ADO.IRISObject.InvokeStatusCode() は、ObjectScript $Status オブジェクトを返す呼び出しメソッドが、指定された引数で呼び出された場合にエラーをスローするかどうかをテストします。呼び出しが失敗した場合、ObjectScript $Status エラー番号とメッセージを含む RuntimeException エラーがスローされます。詳細および例については、“%Status エラー・コードの取得”を参照してください。

```
void  invokeStatusCode(String methodName, Object... args)
```

パラメータ：

- methodName – 呼び出されるインスタンス・メソッドの名前。
- args – サポートされる型の 0 個以上の引数。

有効な引数の型と関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

InvokeString()

ADO.IRISObject.InvokeString() は、オブジェクトのインスタンス・メソッドを呼び出して、値を string として返します。

```
String  invokeString(String methodName, Object... args)
```

パラメータ：

- methodName – 呼び出されるインスタンス・メソッドの名前。
- args – サポートされる型の 0 個以上の引数。

有効な引数の型と関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

InvokeVoid()

ADO.IRISObject.InvokeVoid() は、オブジェクトのインスタンス・メソッドを呼び出しますが、値は返しません。

```
void InvokeVoid(string methodName, params object[] args)
```

パラメータ：

- methodName – 呼び出されるインスタンス・メソッドの名前。

- `args` — サポートされる型の 0 個以上の引数。

有効な引数の型と関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

iris [attribute]

jdbc.IRISObject.iris は、このオブジェクトに関連付けられた IRIS インスタンスへのアクセスを提供するパブリック・フィールドです。

```
public IRIS iris
```

関連情報については、“IRISObject のサポートされているデータ型”を参照してください。

Set()

ADO.IRISObject. Set() は、プロキシ・オブジェクトのプロパティを設定します。

```
void Set(string propertyName, Object value)
```

パラメータ：

- `propertyName` — value が割り当てられるプロパティの名前。
- `value` — 割り当てるプロパティ値。