Native SDK for Java の使用法

Version 2023.1
2024-01-02

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

Native SDK for Java の使用法
InterSystems IRIS Data Platform　Version 2023.1　2024-01-02

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

Java Native SDK 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。Native SDK クラスおよびメソッドの簡単な説明は、"Java Native SDK のクイック・リファレンス" を参照してください。

InterSystems Native SDK for Java 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。

- ObjectScript メソッドおよび関数の呼び出し ー Java のネイティブ・メソッドを呼び出すのと同じくらい簡単に、Java アプリケーションから任意の埋め込み言語のクラスメソッドを呼び出します。
- 埋め込み言語オブジェクトの操作 ー Java プロキシ・オブジェクトを使用して、埋め込み言語のクラス・インスタンスを制御します。インスタンスがネイティブの Java オブジェクトであるかのように、インスタンス・メソッドを呼び出し、プロパティ値を取得または設定します。
- グローバル配列の操作 ー 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) に直接アクセスします。
- インターシステムズのトランザクションとロックの使用 ー ObjectScript トランザクションおよびロック・メソッドの Native SDK 実装を使用して、インターシステムズのデータベースを操作します。

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

その他の言語用の Native SDK

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

- Native SDK for .NET 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 ー 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

Java 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 Java 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 引数を参照で渡し、%Status コードを確認する方法を示します。

2.1 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：classMethodBoolean()、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、Integer、Short、String、Long、Double、Float、byte[]、Boolean、Time、Date、Timestamp、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（“Java 逆プロキシ・オブジェクトの使用法”を参照)、任意の Java 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 irisjv 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 = "";

comment = "cmBoolean() tests whether two numbers are equal (true=1,false=0): ";
boolean boolVal = irisjv.classMethodBoolean(className,"cmBoolean",7,7);
System.out.println(comment+boolVal);
```

```
comment = "cmBytes creates byte array [72,105,33]. String value of array: ";
byte[] byteVal = irisjv.classMethodBytes(className,"cmBytes",72,105,33);
System.out.println(comment+(new String(byteVal)));

comment = "cmString() concatenates \"Hello\" + arg: ";
String stringVal = irisjv.classMethodString(className,"cmString","World");
System.out.println(comment+stringVal);

comment = "cmLong() returns the sum of two numbers: ";
Long longVal = irisjv.classMethodLong(className,"cmLong",7,8);
System.out.println(comment+longVal);

comment = "cmDouble() multiplies a number by 1.5: ";
Double doubleVal = irisjv.classMethodDouble(className,"cmDouble",10);
System.out.println(comment+doubleVal);

comment = "cmProcedure assigns a value to global node ^cmGlobal: ";
irisjv.classMethodVoid(className,"cmVoid",67);
// Read global array ^cmGlobal and then delete it
System.out.println(comment+irisjv.getInteger("^cmGlobal"));
irisjv.kill("cmGlobal");

comment = "cmList() returns a $LIST containing two values: ";
IRISList listVal = irisjv.classMethodList(className,"cmList","The answer is ",42);
System.out.println(comment+listVal.get(1)+listVal.get(2));
```

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
```

2.2 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。ObjectScript では、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（"サーバ側プログラミングの入門ガイド" の "呼び出しメソッド" を参照）。

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：functionBoolean()、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、Integer、Short、String、Long、Double、Float、byte[]、Boolean、Time、Date、Timestamp、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（"Java 逆プロキシ・オブジェクトの使用法" を参照）、任意の Java 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 Java 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）。irisjv 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): ";
boolean boolVal = irisjv.functionBool("fnBoolean",routineName,7,7);
System.out.println(comment+boolVal);

comment = "fnBytes creates byte array [72,105,33]. String value of the array: ";
byte[] byteVal = new String(irisjv.functionBytes("fnBytes",routineName,72,105,33));
System.out.println(comment+(new String(byteVal)));

comment = "fnString() concatenates \"Hello\" + arg: ";
String stringVal = irisjv.functionString("fnString",routineName,"World");
System.out.println(comment+stringVal);

comment = "fnLong() returns the sum of two numbers: ";
Long longVal = irisjv.functionInt("fnLong",routineName,7,8);
System.out.println(comment+longVal);

comment = "fnDouble() multiplies a number by 1.5: ";
Double doubleVal = irisjv.functionDouble("fnDouble",routineName,5);
System.out.println(comment+doubleVal);
```

```
comment = "fnProcedure assigns a value to global array ^fnGlobal: ";
irisjv.procedure("fnProcedure",routineName,88);
// Read global array ^fnGlobal and then delete it
System.out.println(comment+irisjv.getInteger("^fnGlobal")+"\n\n");
irisjv.kill("fnGlobal");

comment = "fnList() returns a $LIST containing two values: ";
IRISList listVal = irisjv.functionList("fnList",routineName,"The answer is ",42);
System.out.println(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
```

2.3 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。

2.3.1 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 jdbc.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
irisjv.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 {
  System.out.print("\n\nCalling %SYS.DatabaseQuery.GetDatabaseFreeSpace()... ");
  status = irisjv.classMethodObject("%SYS.DatabaseQuery","GetDatabaseFreeSpace",dir,freeMB);

  System.out.println("\nFree space in " + dir + " = " + freeMB.value + "MB");
}
catch (RuntimeException e) {
  System.out.print("Call to class method GetDatabaseFreeSpace() returned error:");
  System.out.println(e.getMessage());
}
```

出力 :

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

2.3.2 %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 RuntimeException 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 irisjv 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 a RuntimeException containing the %Status error message:
    irisjv.classMethodStatusCode(className,methodName,pwd);
// This call would fail silently or throw a generic error message:
    Object status = irisjv.classMethodObject(className,methodName,pwd);
    System.out.println("\nPassword validated!");

  } catch (RuntimeException e) {
    System.out.println("Call to "+methodName+"(\""+pwd+"\") returned error:");
    System.out.println(e.getMessage());
  }
```

この例では、意図的に、参照渡し引数を使用しないメソッドを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 (“Java 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。

3

Java 逆プロキシ・オブジェクトの使用法

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

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 Java 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 Java 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。

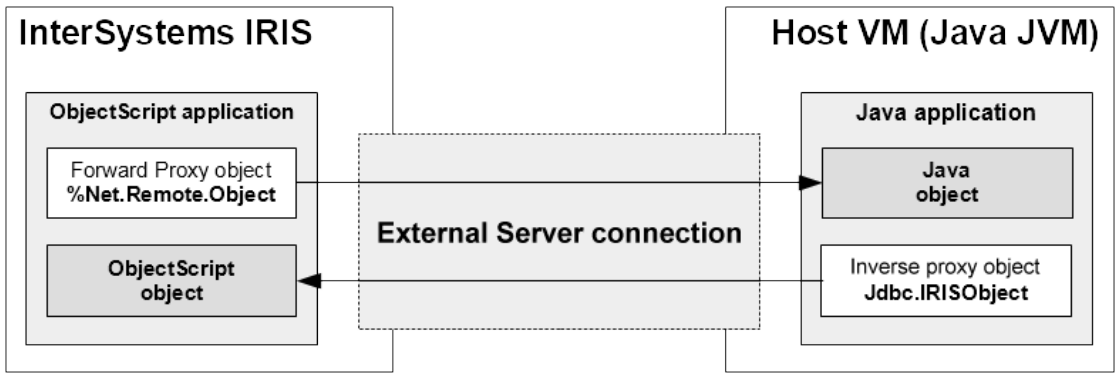
このセクションでは、以下のトピックについて説明します。

- 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。

3.1 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 Java 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 3-1: 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 Java 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)。Java 逆プロキシは反対方向に動作し、Java 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。

3.2 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。

- **jdbc.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。
- **jdbc.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)irisjv.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。
- Java 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 Java 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。

3.3 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書きを行うアクセサ 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)irisjv.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
```

上の例では、IRIS.classMethodObject() への呼び出しによって逆プロキシ・オブジェクト test が作成され、以下のメソッドを使用して 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ています。

- IRISObject.invokeVoid() はターゲット・インスタンス・メソッド initialize() を起動します。このメソッドは、文字列引数を受け入れますが、値は返しません。
- IRISObject.invoke() は、ターゲット・インスタンス・メソッド add() 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。
- IRISObject.set() は、新しい値をターゲット・プロパティ name に割り当てます。
- IRISObject.get() は、ターゲット・プロパティ name 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。

3.4 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() メソッドおよび 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 Java 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、getBoolean()、getBytes()、getDouble()、getIRISList()、getLong()、getObject()、getString()、および invokeVoid() 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、invokeBoolean()、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。

サポートされる標準のデータ型のメソッドのほかに、IRISObject は invokeStatusCode() も提供します。これは、ObjectScript %Status 戻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 メソッドすべては、methodName の String 引数に加え、0 個以上のメソッド引数を取ります。これは以下のいずれかの型にできます : Integer、Short、String、Long、Double、Float、byte[]、Boolean、Time、Date、Timestamp、IRISList、または IRISObject。接続が双方向の場合、任意の Java 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。

4

グローバル配列の操作

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、Java 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 **JDBC 接続の作成**

この章の例では、irisjv 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 JDBC 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
      String connStr = "jdbc:IRIS://127.0.0.1:1972/USER";
      String user = "_SYSTEM";
      String pwd = "SYS";
      IRISConnection conn = (IRISConnection)
java.sql.DriverManager.getConnection(connStr,user,pwd);
      IRIS irisjv = 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。JDBC 接続の作成方法の一般的な情報は、"InterSystems ソフトウェアでの Java の使用法" の "JDBC 接続の確立" を参照してください。。

4.1 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、irisjv 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

```
package nativedemo;
import com.intersystems.jdbc.*;
```

Java

```
public class NativeDemo {
  public static void main(String[] args) throws Exception {
    try {

//Open a connection to the server and create an IRIS object
      String connStr = "jdbc:IRIS://127.0.0.1:1972/USER";
      String user = "_SYSTEM";
      String password = "SYS";
      IRISConnection conn = (IRISConnection)
java.sql.DriverManager.getConnection(connStr,user,password);
      IRIS irisjv = IRIS.createIRIS(conn);

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

// Read the values from the database and print them
      String subfoo = irisjv.getString("root", "foo", "SubFoo");
      String underbar = irisjv.getString("root", "bar", "lowbar", "UnderBar");
      System.out.println("Created two values: \n"
         + " root(\"foo\",\"SubFoo\")=" + subfoo + "\n"
        + " root(\"bar\",\"lowbar\",\"UnderBar\")=" + underbar);

//Delete the global array and terminate
      irisjv.kill("root"); // delete global array root
      irisjv.close();
      conn.close();
    }
    catch (Exception e) {
      System.out.println(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、irisjv 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。

4.1.1 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。添え字とは異なり、グローバル名には文字、数字、およびピリオドのみを含めることができます ("グローバル命名規則" を参照)。

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。

4.1.2 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、″グローバルの使用法″ の “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。

4.2 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。

4.2.1 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。

```
irisjv.set("first", "myGlobal", "A");    // create node myGlobal("A") = "first"
irisjv.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。number 引数は、Double、Integer、Long、または Short のいずれかになります。

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++) {
  irisjv.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。

4.2.2 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：Boolean、byte[]、Double、Float、Integer、Long、Short、String、Date、Time、Timestamp に加え、Object、IRISList、java.io.InputStream と java.io.Reader のサブクラス、および java.io.Serializable を実装するオブジェクト。

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：getBoolean()、getShort()、getInteger()、getLong()、getDouble()、getFloat()
- 文字列およびバイナリ・データ型：getBytes()、getString()
- オブジェクトおよび $list データ型：getObject()、getIRISList()
- 一時データ型：getDate()、getTime()、getTimestamp()
- その他のデータ型：getInputStream()、getReader()

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。

4.2.3 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。

```
irisjv.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。

```
irisjv.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。

4.3 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。
- すべてのレベルのサブノードの検索 – サブノードの存在をテストしてノード・レベルに関係なくすべてのサブノードにわたって反復する方法について説明します。

4.3.1 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サブノードです。現在のターゲット・ノードの子は、ターゲット・アドレスに添え字を 1 つのみ追加してアドレス指定できます。同じ親の下のすべての子ノードは相互に兄弟ノードです。例えば、以下のグローバル配列には、親ノード ^myNames("people") の下に 6 つの兄弟ノードがあります。

```
^myNames                                (valueless root node)
   ^myNames("people")                   (valueless level 1 node)
      ^myNames("people","Anna") = 2     (first level 2 child node)
      ^myNames("people","Julia") = 4
      ^myNames("people","Misha") = 5
      ^myNames("people","Ruri") = 3
      ^myNames("people","Vlad") = 1
      ^myNames("people","Zorro") = -1   (this node will be deleted in example)
```

注釈 **照合順序**

反復子は、照合順 (この場合は、Anna、Julia、Misha、Ruri、Vlad、Zorro というアルファベット順) にノードを返します。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、InterSystems IRIS によって自動的に、ストレージ定義で指定された照合順にノードが格納されます。この例のノードは、作成された順序に関係なく、示された順序で格納されます。

ここでは、以下のメソッドを示します。

- 反復子を作成して一連の子ノードを走査するために使用されるメソッド
  - jdbc.IRIS.getIRISIterator() は、指定されたノードから開始して、グローバルの IRISIterator のインスタンスを返します。
  - IRISIterator.next() は、照合順で次の兄弟ノードの添え字を返します。
  - IRISIterator.hasNext() は、照合順に別の兄弟ノードがある場合に true を返します。
- 現在のノードに作用するメソッド
  - IRISIterator.getValue() は、現在のノード値を返します。
  - IRISIterator.getSubscriptValue() は、現在の添え字 (最後に成功した next() の呼び出しと同じ値) を返します。
  - IRISIterator.remove() は、現在のノードとそのサブノードすべてを削除します。

以下の例は、^myNames("people") の下の各子ノードを反復処理します。値が 0 以上の場合は添え字とノード値を出力し、値が負の場合はノードを削除します。

^myNames("people") の下のすべての兄弟ノードの検索

```
// Read child nodes in collation order while iter.hasNext() is true
  System.out.print("Iterate from first node:");
  try {
    IRISIterator iter = irisjv.getIRISIterator("myNames","people");
    while (iter.hasNext()) {
      iter.next();
      if ((Long)iter.getValue()>=0) {
        System.out.print(" \"" + iter.getSubscriptValue() + "\"=" + iter.getValue()); }
      else {
        iter.remove();
      }
    };
  } catch  (Exception e) {
    System.out.println( e.getMessage());
  }
```

- getIRISIterator() の呼び出しによって、^myNames("people") の直接の子の反復子インスタンス iter が作成されます。
- `while` ループが繰り返されるたびに、以下のアクションが実行されます。
  - next() は、照合順で次に有効なノードの添え字を特定し、そのノードに反復子を配置します(最初の反復では、添え字は `"Anna"` で、ノード値は `2` です)。
  - getValue() から返されたノード値が負である場合は、remove() が呼び出されてノードが削除されます(すべてのサブノードも含みます。これは、現在のノードに対する kill() の呼び出しと同じです)。

    それ以外の場合は、getSubscriptValue() と getValue() を使用して、現在のノードの添え字と値が出力されます。
- この順序に子ノードがそれ以上ないことを示す `false` を hasNext() が返すと、`while` ループが終了します。

このコードは、以下の行を出力します (要素 `"Zorro"` は、値が負であるため出力されていません)。

```
Iterate from first node: "Anna"=2 "Julia"=4 "Misha"=5 "Ruri"=3 "Vlad"=1
```

この例は、非常に単純であり、いくつかの状況で失敗する可能性があります。最初または最後のノードから開始したくない場合、どうなるでしょうか。コードが、値なしノードから値を取得しようとする場合、どうなるでしょうか。グローバル配列の複数のレベルにデータがある場合、どうなるでしょうか。以下のセクションでは、これらの状況を処理する方法を説明します。

4.3.2 すべてのレベルのサブノードの検索

次の例では、少し複雑なサブノードのセットを検索します。新しい子ノード "dogs" を ^myNames に追加し、それをこの例のターゲット・ノードとして使用します。

```
^myNames                                        (valueless root node)
   ^myNames("dogs")                             (valueless level 1 node)
      ^myNames("dogs","Balto") = 6
      ^myNames("dogs","Hachiko") = 8
      ^myNames("dogs","Lassie")                 (valueless level 2 node)
         ^myNames("dogs","Lassie","Timmy") = 10 (level 3 node)
      ^myNames("dogs","Whitefang") = 7
   ^myNames("people")                           (valueless level 1 node)
      [five child nodes]                        (as listed in previous example)
```

ターゲット・ノード ^myNames("dogs") には 5 つのサブノードがありますが、そのうちの 4 つのみが子ノードです。4 つのレベル 2 サブノードに加え、レベル 3 サブノード ^myNames("dogs","Lassie","Timmy") もあります。検索では "Timmy" は見つかりません。これは、このサブノードが "dogs" の子ではなく "Lassie" の子であるため、その他のサブノードの兄弟ではないからです。

注釈 **添え字リストおよびノード・レベル**

用語ノード・レベルは、添え字リスト内の添え字数を表します。例えば、^myGlobal("a","b","c") は、“レベル 3 ノード” です。これは、“3 つの添え字を持つノード” を別の方法で表現しているだけです。

ノード ^myNames("dogs","Lassie") には子ノードがありますが、これには値がありません。この場合、getValue() の呼び出しでは null が返されます。以下の例では、逆の照合順で ^myNames("dogs") の子が検索されます。

^myNames("dogs") の下の最後のノードから逆の順序でノードを取得

```
// Read child nodes in descending order while iter.next() is true
  System.out.print("Descend from last node:");
  try {
    IRISIterator iter = irisjv.getIRISIterator("myNames","dogs");
    while (iter.hasPrevious()) {
      iter.previous();
      System.out.print(" \"" + iter.getSubscriptValue() + "\"");
      if (iter.getValue()!=null) System.out.print("=" + iter.getValue());
    };
  } catch  (Exception e) {
    System.out.println( e.getMessage());
  }
```

このコードは以下の行を出力します。

```
Descend from last node: "Whitefang"=7 "Lassie" "Hachiko"=8 "Balto"=6
```

前の例の検索では、グローバル配列 ^myNames のノードのいくつかが検出されません。これは、検索範囲がさまざまな方法で制限されるためです。

- ノード ^myNames("dogs","Lassie","Timmy") は、^myNames("dogs") のレベル 2 サブノードではないため検出されません。
- ^myNames("people") の下のレベル 2 ノードは、^myNames("dogs") の下のレベル 2 ノードの兄弟ではないため検出されません。

いずれの場合でも、問題は、previous() および next() が、同じ親の下にある開始アドレスと同じレベルのノードしか検出しないことです。兄弟ノードの各グループに対して異なる開始アドレスを指定する必要があります。

ほとんどの場合は、既知の構造を処理することになるため、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走査します。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、以下の jdbc.IRIS メソッドを使用して、指定されたノードにサブノードがあるかどうかを確認できます。

- isDefined() ー 指定されたノードが存在しない場合は 0 を返し、ノードが存在し、それに値がある場合は 1 を返します。ノードに値はないが、サブノードがある場合は 10 を返し、値とサブノードの両方がある場合は 11 を返します。

isDefined() が 10 または 11 を返す場合、サブノードが存在し、前述の例で説明したように反復子を作成することによって処理することができます。再帰アルゴリズムでは、このテストを使用して任意の数のレベルを処理できます。

4.4 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、この章の前述の各セクションの例では、常に Integer または String の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応するデータ型固有のメソッドも提供されています。

IRIS.set()

IRIS.set() メソッドは、データ型 Boolean、byte[]、Double、Integer、Long、Short、Float、String、IRISList に加えて、Java クラス Date、Time、Timestamp、InputStream、Reader、および Serializable を実装するクラスをサポートしています。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 Java 変数への変換を試みます：getBoolean()、getShort()、getInteger()、getLong()、getDouble()、または getFloat()。ターゲット・ノードに値がないか、ターゲット・ノードが存在しない場合、数値フェッチ・メソッドは UndefinedException をスローします。

Integer 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。getInteger() メソッドおよび getLong() メソッドは、Double 値または Float 値に適用しても信頼できる結果が生成されず、これらの値に対して例外がスローされる場合があります。

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、それを希望の形式で返します。

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 Java クラス向けのゲッター

IRIS クラスは、Java クラス Date、Time、Timestamp、InputStream、および Reader 向けのゲッターもサポートします。Serializable を実装するクラスは、getObject() で取得できます。

- getDate()、getTime()、getTimestamp() ー java.sql データ型 Date、Time、および Timestamp を取得します。
- getInputStream() ー java.io.InputStream を実装するオブジェクトを取得します。
- getReader() ー java.io.Reader を実装するオブジェクトを取得します。
- getObject() ー ターゲット・ノードの値を取得し、それを Object として返します。

  java.io.Serializable を実装するオブジェクトは、getObject() の戻り値を適切なクラスにキャストすることによって取得できます。

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。

5
トランザクションとロック

Native SDK for Java 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 JDBC 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・モデルと JDBC (**java.sql**) トランザクション・モデルを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 JDBC/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 JDBC/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。

5.1 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 JDBC 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、JDBC 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 Java には、トランザクションを制御する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";
        irisjv.tStart();

        // getTLevel() is 1: create myGlobal(1) = "firstValue"
        irisjv.set("firstValue", globalName, irisjv.getTLevel());

        irisjv.tStart();
        // getTLevel() is 2: create myGlobal(2) = "secondValue"
        irisjv.set("secondValue", globalName, irisjv.getTLevel());

        irisjv.tStart();
        // getTLevel() is 3: create myGlobal(3) = "thirdValue"
        irisjv.set("thirdValue", globalName, irisjv.getTLevel());

        System.out.println("Node values before rollback and commit:");
        for (int ii=1;ii<4;ii++) {
          System.out.print(globalName + "(" + ii + ") = ");
         if (irisjv.isDefined(globalName,ii) > 1) System.out.println(irisjv.getString(globalName,ii));

          else System.out.println("<valueless>");
        }
      // prints: Node values before rollback and commit:
      //         myGlobal(1) = firstValue
      //         myGlobal(2) = secondValue
      //         myGlobal(3) = thirdValue

        irisjv.tRollbackOne();
        irisjv.tRollbackOne();  // roll back 2 levels to getTLevel 1
        irisjv.tCommit();  // getTLevel() after commit will be 0
        System.out.println("Node values after the transaction is committed:");
        for (int ii=1;ii<4;ii++) {
          System.out.print(globalName + "(" + ii + ") = ");
         if (irisjv.isDefined(globalName,ii) > 1) System.out.println(irisjv.getString(globalName,ii));

          else System.out.println("<valueless>");
        }
      // prints: Node values after the transaction is committed:
      //         myGlobal(1) = firstValue
      //         myGlobal(2) = <valueless>
      //         myGlobal(3) = <valueless>
```

5.2 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、JDBC/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, Integer timeout, String globalName, String... subscripts)
unlock (String lockMode, String globalName, String... 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”

6

Java Native SDK のクイック・リファレンス

これは、com.intersystems.jdbc 内の以下の拡張クラスで構成される InterSystems IRIS Native SDK for Java 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 シリアル化をサポートします。
- クラス IRISObject は、Java 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 JDBC ドライバ (com.intersystems.jdbc) の一部です。これらのクラスは標準の JDBC 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。JDBC 接続の詳細は、"InterSystems ソフトウェアでの Java の使用法" の "JDBC 接続の確立" を参照してください。

注釈　この章は、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、Java 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、InterSystems IRIS JDBC ドライバ・クラスのオンライン・クラス・ドキュメントを参照してください。

6.1 クラス IRIS

クラス IRIS は com.intersystems.jdbc (InterSystems JDBC ドライバ) 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。

6.1.1 IRIS メソッドの詳細

classMethodBoolean()

jdbc.IRIS.classMethodBoolean() 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、Boolean のインスタンスを返します。

```
final Boolean classMethodBoolean(String className, String methodName, 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。

classMethodBytes()

jdbc.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。

```
final byte[] classMethodBytes(String className, String methodName, 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。

classMethodDouble()

jdbc.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。

```
final Double classMethodDouble(String className, String methodName, 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。

classMethodIRISList()

jdbc.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。

```
final IRISList classMethodIRISList(String className, String methodName, 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` — 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。

classMethodLong()

jdbc.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。

```
final Long classMethodLong(String className, String methodName, 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()

jdbc.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、“Java 逆プロキシ・オブジェクトの使用法”を参照してください。

```
final Object classMethodObject(String className, String methodName, 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()

jdbc.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 RuntimeException エラーがスローされます。

```
final void classMethodStatusCode(String className, String methodName, 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()

jdbc.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。

```
final String classMethodString(String className, String methodName, 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()

jdbc.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。

```
final void classMethodVoid(String className, String methodName, 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()

jdbc.IRIS.close() は IRIS オブジェクトを閉じます。

```
final void close() throws Exception
```

createIRIS() [static]

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

```
static IRIS createIRIS(IRISConnection conn) throws SQLException
```

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。

functionBoolean()

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

```
final Boolean functionBoolean(String functionName, String routineName, 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()

jdbc.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。

```
final byte[] functionBytes(String functionName, String routineName, 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()

jdbc.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。

```
final Double functionDouble(String functionName, String routineName, 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()

jdbc.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。

```
final IRISList functionIRISList(String functionName, String routineName, 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()

jdbc.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。

```
final Long functionLong(String functionName, String routineName, 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()

jdbc.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、“Java 逆プロキシ・オブジェクトの使用法”を参照してください。

```
final Object functionObject(String functionName, String routineName, 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()

jdbc.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。

```
final String functionString(String functionName, String routineName, 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]

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

```
static final String getAPIVersion()
```

getBoolean()

jdbc.IRIS.getBoolean() は、グローバルの値を Boolean 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。

```
final Boolean getBoolean(String globalName, 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()

jdbc.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)。

```
final byte[] getBytes(String globalName, 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。

getDate()

jdbc.IRIS.getDate() は、グローバルの値を java.sql.Date 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)。

```
final java.sql.Date getDate(String globalName, 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()

jdbc.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。

```
final Double getDouble(String globalName, 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。

getFloat()

jdbc.IRIS.getFloat() は、グローバルの値を Float 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。

```
final Float getFloat(String globalName, 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。

getInputStream()

jdbc.IRIS.getInputStream() は、グローバルの値を java.io.InputStream 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)。

```
final InputStream getInputStream(String globalName, 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。

getInteger()

jdbc.IRIS.getInteger() は、グローバルの値を Integer 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。

```
final Integer getInteger(String globalName, 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()

jdbc.IRIS.getIRISIterator() は、指定されたノードの IRISIterator オブジェクト (“クラス IRISIterator”を参照) 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。

```
final IRISIterator getIRISIterator(String globalName, Object... subscripts)
final IRISIterator getIRISIterator(int prefetchSizeHint, String globalName, Object... subscripts)
```

パラメータ：

- prefetchSizeHint — (オプション) サーバからデータを取得する際にフェッチするバイト数のヒント。
- 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()

jdbc.IRIS.getIRISList() は、ノードの値を 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, 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。

getLong()

jdbc.IRIS.getLong() は、グローバルの値を Long 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。

```
final Long getLong(String globalName, 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()

jdbc.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、“Java 逆プロキシ・オブジェクトの使用法”を参照してください。

```
final Object getObject(String globalName, 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。

このメソッドを使用して、java.io.Serializable を実装するオブジェクトを取得します。set() メソッドが Serializable インスタンスを受け入れると、そのインスタンスはシリアライズされた形式で格納され、getObject() の戻り値を適切なクラスにキャストすることによってデシリアライズできます。

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。

getReader()

jdbc.IRIS.getReader() は、グローバルの値を java.io.Reader 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)。

```
final Reader getReader(String globalName, 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()

jdbc.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。

```
final String getServerVersion()
```

getShort()

jdbc.IRIS.getShort() は、グローバルの値を Short 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。

```
final Short getShort(String globalName, 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()

jdbc.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。Java の空の文字列 "" は、ObjectScript では null 文字 $CHAR(0) に変換されます。Java の null は、ObjectScript では空の文字列に変換されます。この変換は、JDBC によるこれらの値の処理方法と一致しています。

```
final String getString(String globalName, 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。

getTime()

jdbc.IRIS.getTime() は、グローバルの値を java.sql.Time 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）。

```
final java.sql.Time getTime(String globalName, 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。

getTimestamp()

jdbc.IRIS.getTimestamp() は、グローバルの値を java.sql.Timestamp 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)。

```
final java.sql.Timestamp getTimestamp(String globalName, 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()

jdbc.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。

```
final Integer 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、JDBC/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()

jdbc.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。

```
final long increment(Integer value, String globalName, Object... subscripts)
```

パラメータ：

- value – このノードを設定する Integer 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()

jdbc.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、“すべてのレベルのサブノードの検索”を参照してください。

```
final int isDefined(String globalName, 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()

jdbc.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。

```
final void kill(String globalName, 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()

jdbc.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。

```
final boolean lock(String lockMode, Integer timeout, String globalName, 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 – ロックの取得を待機する時間 (秒単位)。
- 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、JDBC/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()

jdbc.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。

```
final void procedure(String procedureName, String routineName, 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()

jdbc.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。

```
final 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、JDBC/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()

jdbc.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。

```
final void set(Boolean value, String globalName, Object... subscripts)
final void set(byte[] value, String globalName, Object... subscripts)
final void set(Short value, String globalName, Object... subscripts)
final void set(Integer value, String globalName, Object... subscripts)
final void set(Long value, String globalName, Object... subscripts)
final void set(Double value, String globalName, Object... subscripts)
final void set(Float value, String globalName, Object... subscripts)
final void set(String value, String globalName, Object... subscripts)

final void set(java.sql.Date value, String globalName, Object... subscripts)
final void set(java.sql.Time value, String globalName, Object... subscripts)
final void set(java.sql.Timestamp value, String globalName, Object... subscripts)
final void set(InputStream value, String globalName, Object... subscripts)
final void set(Reader value, String globalName, Object... subscripts)
final<T extends Serializable> void set(T value, String globalName, Object... subscripts)
final void set(IRISList value, String globalName, Object... subscripts)
final void set(Object value, String globalName, 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。Java の空の文字列 "" は、ObjectScript では null 文字 $CHAR(0) に変換されます。Java の null は、ObjectScript では空の文字列に変換されます。この変換は、Java によるこれらの値の処理方法と一致しています。
- java.io.InputStream – 現在、単一グローバル・ノードの最大サイズに制限されています。
- java.io.Reader – 現在、単一グローバル・ノードの最大サイズに制限されています。
- java.io.Serializable – 値が Serializable を実装するオブジェクトのインスタンスである場合、グローバル値として設定される前にシリアライズされます。値を取得するには、getObject() を使用します。

tCommit()

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

```
final 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、JDBC/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()

jdbc.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。

```
final 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、JDBC/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()

jdbc.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。

```
final 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、JDBC/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()

jdbc.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)。

```
final 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、JDBC/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()

jdbc.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。

```
final void unlock(String lockMode, String globalName, 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、JDBC/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。

6.2 クラス IRISIterator

クラス **IRISIterator** は **com.intersystems.jdbc** (InterSystems JDBC ドライバ) のメンバであり、**java.util.Iterator** を実装します。

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

6.2.1 IRISIterator メソッドの詳細

getSubscriptValue()

jdbc.IRISIterator.getSubscriptValue() 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。

この反復子を持つ現在のノードに対して remove() が呼び出された場合、またはこの反復子から返された最後の要素がない (next() または previous() の呼び出しが正常に実行されなかった) 場合は、**IllegalStateException** をスローします。

```
String getSubscriptValue() throws IllegalStateException
```

getValue()

jdbc.IRISIterator.getValue() は、現在の反復子位置にあるノードの値を取得します。この反復子を持つ現在のノードに対して remove() が呼び出された場合、またはこの反復子から返された最後の要素がない (next() または previous() の呼び出しが正常に実行されなかった) 場合は、**IllegalStateException** をスローします。

```
Object getValue() throws IllegalStateException
```

hasNext()

jdbc.IRISIterator.hasNext() は、反復処理に他の要素がある場合、true を返します (つまり、next() が例外をスローするのではなく、要素を返す場合、true を返します)。

```
boolean hasNext()
```

hasPrevious()

jdbc.IRISIterator.hasPrevious() は、反復処理に前の要素がある場合、true を返します (つまり、previous() が例外をスローするのではなく、要素を返す場合、true を返します)。

```
boolean hasPrevious()
```

next()

jdbc.IRISIterator.next() は、反復処理の次の要素を返します。反復処理にそれ以上要素がない場合は、**NoSuchElementException** をスローします。

```
String next() throws NoSuchElementException
```

previous()

jdbc.IRISIterator.previous() は、反復処理の前の要素を返します。反復処理に前の要素がない場合は、NoSuchElementException をスローします。

```
String previous() throws NoSuchElementException
```

remove()

jdbc.IRISIterator.remove() は、この反復子から返された最後の要素を基のコレクションから削除します。このメソッドは、next() または previous() の呼び出しごとに 1 回のみ呼び出すことができます。この反復子を持つ現在のノードに対して remove() が呼び出された場合、またはこの反復子から返された最後の要素がない (next() または previous() の呼び出しが正常に実行されなかった) 場合は、IllegalStateException をスローします。

```
void remove() throws IllegalStateException
```

startFrom()

jdbc.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。

```
void startFrom(Object subscript)
```

このメソッドを呼び出した後、next() または previous() を使用して、アルファベットの照合順序で次に定義されているサブノードに反復子を進めます。これらのいずれかのメソッドが呼び出されるまで、反復子は、定義されているサブノードに配置されません。反復子が進む前に getSubscriptValue() または getValue() を呼び出すと、IllegalStateException がスローされます。

6.3 クラス IRISList

クラス IRISList は com.intersystems.jdbc (InterSystems JDBC ドライバ) 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 Java 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、jdbc.IRIS のメソッド classMethodIRISList()、functionIRISList()、getIRISList() によっても返されます。

6.3.1 IRISList コンストラクタ

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

```
IRISList ()
IRISList (IRISList list)
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。

6.3.2 IRISList メソッドの詳細

add()

jdbc.IRISList.add() は、Object、Object[] の各要素、または Collection の各要素を、IRISList の末尾に追加します。サポートされないデータ型を追加しようとすると、RuntimeException がスローされます。

サポートされているデータ型は、Boolean、Short、Integer、Long、java.math.BigInteger、java.math.BigDecimal、Float、Double、String、Character、byte[]、char[]、IRISList、および null です。

```
void add(Object value)
void add(Object[] array)
void add(Collection collection)
```

パラメータ：

- value – サポートされている任意の型の Object インスタンス。
- array – サポートされている型の Object 要素を含む Object[]。配列内の各 Object は、別個の要素としてリストに追加されます。
- collection – サポートされている型の Object 要素を含む Collection。コレクション内の各 Object は、コレクションの反復子から返された順でリストに追加されます。

IRISList 要素の追加

add() メソッドが IRISList の 2 つのインスタンスを連結することはありません。add() が IRISList インスタンスを検出すると (単一の値として、または配列やコレクションの要素として)、そのインスタンスは常に単一の IRISList 要素として追加されます。

ただし、IRISList.toArray() を使用して、IRISList を Object[] に変換できます。結果として生成される配列で add() を呼び出すと、各要素が個別に追加されます。

clear()

jdbc.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()

jdbc.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]

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

```
static int DateToHorolog(Date value, Calendar calendar)
```

パラメータ：

- value – 変換する Date の値。
- calendar – GMT および DST のオフセットを決定するために使用される Calendar。既定のカレンダーの場合は NULL にすることができます。

equals()

jdbc.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。

```
boolean equals (Object list)
```

パラメータ：

- list – 比較する IRISList のインスタンス。

get()

jdbc.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、IndexOutOfBoundsException 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()

jdbc.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、IndexOutOfBoundsException 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。

HorologToDate() [static]

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

```
static Date HorologToDate(int HorologValue, Calendar calendar)
```

パラメータ：

- HorologValue – $Horolog 文字列の day フィールドを表す int。時間フィールドはゼロ (つまり、深夜 0 時) に設定されるため、時間フィールドが最初にゼロでない場合は、Date はラウンドトリップしません。

- `calendar` — GMT および DST のオフセットを決定するために使用される Calendar。既定のカレンダーの場合は NULL にすることができます。

HorologToTime() [static]

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

```
static Time HorologToTime(int HorologValue, Calendar calendar)
```

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 Time は、ラウンドトリップしません。
- `calendar` — GMT および DST のオフセットを決定するために使用される Calendar。既定のカレンダーの場合は NULL にすることができます。

PosixToTimestamp() [static]

jdbc.IRISList.PosixToTimestamp() は、%Library.PosixTime の値を Timestamp に変換します。指定された Calendar 値を使用して (NULL の場合は既定のタイム・ゾーンを使用)、%PosixTime をローカル時刻として処理します。ラウンドトリップするには、同等のカレンダーを指定する必要があります。"TimestampToPosix()" も参照してください。

```
static Timestamp PosixToTimestamp(long PosixValue, Calendar calendar)
```

パラメータ：

- `PosixValue` — %PosixTime の値 (64 ビットの符号付き整数)。
- `calendar` — GMT および DST のオフセットを決定するために使用される Calendar。既定のカレンダーの場合は NULL にすることができます。

remove()

jdbc.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。

```
boolean 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()

jdbc.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)
```

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。

オブジェクトは、Boolean、Short、Integer、Long、java.math.BigInteger、java.math.BigDecimal、Float、Double、String、Character、byte[]、char[]、IRISList、および null のいずれかの型にできます。

size()

jdbc.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()

jdbc.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、IndexOutOfBoundsException 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。

TimestampToPosix() [static]

jdbc.IRISList.TimestampToPosix() は Timestamp オブジェクトを %Library.PosixTime の値 (64 ビットの符号付き整数) に変換します。指定された Calendar 値を使用して (NULL の場合は既定のタイム・ゾーンを使用)、Timestamp をローカル時刻に変換します。ラウンドトリップするには、同等のカレンダーを指定する必要があります。"PosixToTimestamp()" も参照してください。

```
static long TimestampToPosix(Timestamp Value, Calendar calendar)
```

パラメータ：

- Value ― 変換する Timestamp の値。
- calendar ― GMT および DST のオフセットを決定するために使用される Calendar。既定のカレンダーの場合は NULL にすることができます。

TimeToHorolog() [static]

jdbc.IRISList.TimeToHorolog() は、Time の値を $Horolog 文字列の 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(Time value, Calendar calendar)
```

パラメータ：

- value ― 変換する Time の値。
- calendar ― GMT および DST のオフセットを決定するために使用される Calendar。既定のカレンダーの場合は NULL にすることができます。

toArray()

jdbc.IRISList.toArray() は、このリスト内のすべての要素を含む Object 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。

```
Object[] toArray()
```

toList()

jdbc.IRISList.toList() は、この IRISList 内のすべての要素を含む java.util.ArrayList<Object> を返します。リストが空の場合、長さがゼロの ArrayList が返されます。

```
ArrayList< Object> toList()
```

toString()

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

```
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

6.4 クラス IRISObject

クラス IRISObject は com.intersystems.jdbc (InterSystems JDBC ドライバ) のメンバです。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 (詳細と例は、"Java 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。

- jdbc.IRIS.classMethodObject()
- jdbc.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、"Java 逆プロキシ・オブジェクトの使用法" を参照してください。

6.4.1 IRISObject メソッドの詳細

close()

jdbc.IRISObject.close() はオブジェクトを閉じます。

```
void close()
```

get()

jdbc.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。

getBoolean()

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

```
Boolean getBoolean(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()

jdbc.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()

jdbc.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()

jdbc.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()

jdbc.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()

jdbc.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()

jdbc.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()

jdbc.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, 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。

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()

jdbc.IRISObject.invokeBoolean() 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 Boolean 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。

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()

jdbc.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。

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()

jdbc.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。

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()

jdbc.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。

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()

jdbc.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。

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()

jdbc.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。

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()

jdbc.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。

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()

jdbc.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。

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()

jdbc.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, 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。

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()

jdbc.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 propertyValue)
```

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。