Native SDK for Python の使用法

Version 2024.1
2024-06-03

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

Native SDK for Python 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 Python 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、"Python 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 PYTHON は、以前は ObjectScript を介して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 メソッドおよび関数の呼び出し ー Python 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、Python 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。
- Python からのデータベース・オブジェクトの制御 ー Python 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 Python 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 埋め込み言語の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。
- Python DB-API のサポート ー PEP 249 バージョン 2.0 Python Database API のインターシステムズの実装を使用して、リレーショナル・データベースにアクセスします。

その他の言語用の Native SDK

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

- Native SDK for Java の使用法
- Native SDK for .NET 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

グローバル配列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

Python からのデータベース・メソッドおよび関数の呼び出し

ここでは、ObjectScript クラス・メソッドおよび関数を Python 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.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。

- Python からのクラス・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。
- Python からの関数とプロシージャの呼び出し ー 関数とプロシージャを呼び出す方法を示します。
- 引数の参照渡し ー IRISReference クラスを使用して引数を渡す方法を示します。

2.1 Python からのクラス・メソッドの呼び出し

classMethodValue() メソッドと classMethodVoid() メソッドは、ほとんどの目的に使用できますが、特定の返りタイプが必要な場合は、classMethodBoolean()、classMethodBytes()、classMethodDecimal()、classMethodFloat()、classMethodIRISList()、classMethodInteger()、classMethodObject()、および classMethodString() の IRIS. 型キャスト・メソッドも使用できます。

これらのメソッドはすべて、class_name および method_name の文字列引数に加え、0 個以上のメソッド引数を取ります。

以下の例のコードでは、User.NativeTest という名前の ObjectScript テスト・クラスからいくつかのデータ型のクラス・メソッドを呼び出します (このセクションの最後に示す“ObjectScript クラス User.NativeTest”を参照)。

Python からの 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 irispy はクラス iris.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 (“Python での接続の作成”を参照)。

```
classname = 'User.NativeTest'

comment = ".cmBoolean() tests whether arguments 2 and 3 are equal: "
boolVal = irispy.classMethodBoolean(className,'cmBoolean',2,3)
print(className + comment + str(boolVal))

comment = ".cmBytes returns integer arguments 72,105,33 as a byte array (string value 'Hi!'):
"
byteVal = irispy.classMethodBytes(className,'cmBytes',72,105,33) #ASCI 'Hi!'
print(className + comment + str(byteVal))

comment = ".cmString() concatenates 'Hello' with argument string 'World': "
stringVal = irispy.classMethodString(className,'cmString','World')
```

```
  print(className + comment + stringVal)

  comment = ".cmLong() returns the sum of arguments 7+8: "
  longVal = irispy.classMethodInteger(className,'cmLong',7,8)
  print(className + comment + str(longVal))

  comment = ".cmDouble() multiplies argument 4.5 by 1.5: "
  doubleVal = irispy.classMethodFloat(className,'cmDouble',4.5)
  print(className + comment + str(doubleVal))

  comment = ".cmList() returns a $LIST containing arguments 'The answer is ' and 42: "
  listVal = irispy.classMethodIRISList(className,"cmList","The answer is ",42);
  print(className + comment+listVal.get(1)+str(listVal.get(2)))

  comment = ".cmVoid assigns argument value 75 to global node ^cmGlobal: "
  try:
    irispy.kill('cmGlobal') # delete ^cmGlobal if it exists
    irispy.classMethodVoid(className,'cmVoid',75)
    nodeVal = irispy.get('cmGlobal');  #get current value of ^cmGlobal
  except:
    nodeVal = 'FAIL'
  print(className + comment + str(nodeVal))
```

この例では、classMethodValue() (決まった形式のない値を返します)、classMethodDecimal() (主に高精度への対応において classMethodFloat() とは異なります)、および classMethodObject() (“Python からのデータベース・オブジェクトの制御” で説明しています) は省略されています。

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 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
  }

  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 Python からの関数とプロシージャの呼び出し

注釈 **プロシージャ・コードのサポート**

以前のインターシステムズ・データベース・プラットフォームでは、コードは、オブジェクト指向のクラスおよびメソッドではなく、関数とプロシージャが含まれるモジュールで構成されていました (″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)。以前のコード・ベースには関数が必要なことが多いですが、新しいコードでは可能であればオブジェクト指向のメソッド呼び出しを使用してください。

function() メソッドと procedure() メソッドは、ほとんどの目的に使用できますが、特定の返りタイプが必要な場合は、functionBoolean()、functionBytes()、functionDecimal()、functionFloat()、functionIRISList()、functionObject()、functionInteger()、および functionString() の IRIS. 型キャスト・メソッドも使用できます。

これらのメソッドは、functionLabel および routineName の string 引数に加え、0 個以上の関数引数を取ります。これは、bool、bytes、bytearray、Decimal、float、int、str、IRISList のいずれかとなります。

以下の例のコードでは、NativeRoutine (この例の直後に示されています) という名前の ObjectScript テスト・ルーチンから関数が呼び出されます。

注釈 **組み込みの ObjectScript $ 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 Python 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。

Python からの ObjectScript ルーチンの呼び出し

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 (この例の直後に示されているファイル NativeRoutine.mac)。irispy はクラス iris.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 (“Python での接続の作成” を参照)。

```
routineName = 'NativeRoutine'

comment = ".fnBoolean() tests whether arguments 2 and 3 are equal: "
boolVal = irispy.functionBoolean('fnBoolean',routineName,2,3)
print(routineName + comment + str(boolVal))

comment = ".fnBytes returns integer arguments 72,105,33 as a byte array (string value 'Hi!'):
"
byteVal = irispy.functionBytes('fnBytes',routineName,72,105,33)  #ASCI 'Hi!'
print(routineName + comment + str(byteVal))

comment = ".fnString() concatenates 'Hello' with argument string 'World': "
stringVal = irispy.functionString("fnString",routineName,"World")
print(routineName + comment + stringVal)

comment = ".fnLong() returns the sum of arguments 7+8: "
longVal = irispy.functionInteger('fnLong',routineName,7,8)
print(routineName + comment + str(longVal))

comment = ".fnDouble() multiplies argument 4.5 by 1.5: "
doubleVal = irispy.functionFloat('fnDouble',routineName,4.5)
print(routineName + comment + str(doubleVal))

comment = ".fnList() returns a $LIST containing arguments 'The answer is ' and 42: "
listVal = irispy.functionIRISList("fnList",routineName,"The answer is ",42);
print(routineName + comment + listVal.get(1)+str(listVal.get(2)));

comment = ".fnProcedure() assigns argument value 66 to global node ^fnGlobal: "
try:
  irispy.kill('fnGlobal') # delete ^fnGlobal if it exists
  irispy.procedure('fnProcedure',routineName,66)
  nodeVal = irispy.get('fnGlobal') # get current value of node ^fnGlobal
```

```
except:
  nodeVal = 'FAIL'
print(routineName + comment + str(nodeVal))
```

この例では、function() (決まった形式のない値を返します)、functionDecimal() (主に高精度への対応においてfunctionFloat() とは異なります)、および functionObject() (“Python からのデータベース・オブジェクトの制御”で説明している classMethodObject() メソッドと機能的に同じです) は省略されています。

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。

```
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 引数の参照渡し

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。Native SDK では、メソッドと関数の両方で参照渡しがサポートされます。これには、以下のように引数の値をクラス 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。

```
ref_object = iris.IRISReference(None); // set inital value to None
irispy.classMethodObject("%SomeClass","SomeMethod",ref_object);
returned_value = ref_object.getValue;  // get the method result
```

以下の例では、標準のクラス・ライブラリ・メソッドが呼び出されます。

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。

Python

```
  dir = "C:/InterSystems/IRIS/mgr/iristemp" # directory to be tested
  value = "error"
  status = 0

  freeMB = iris.IRISReference(None) # set inital value to 0
  print("Variable freeMB is type"+str(type(freeMB)) + ", value=" + str(freeMB.getValue()))
  try:
    print("Calling %SYS.DatabaseQuery.GetDatabaseFreeSpace()... ")
    status = irispy.classMethodObject("%SYS.DatabaseQuery","GetDatabaseFreeSpace",dir,freeMB)
    value = freeMB.getValue()
  except:
    print("Call to class method GetDatabaseFreeSpace() returned error:")
  print("(status=" + str(status) + ") Free space in " + dir + " = " + str(value) + "MB\n")
```

出力：

```
Variable freeMB is type<class 'iris.IRISReference'>, value=None
Calling %SYS.DatabaseQuery.GetDatabaseFreeSpace()...
(status=1) Free space in C:/InterSystems/IRIS/mgr/iristemp = 10MB
```

3

Python からのデータベース・オブジェクトの制御

Native SDK は、インターシステムズの外部サーバと連携し、外部の Python アプリケーションが ObjectScript または組み込み Pythonで記述されたデータベース・クラスのインスタンスを制御できるようにします。Native SDK の逆プロキシ・オブジェクトでは、外部サーバ接続を使用して、ターゲット・データベース・オブジェクトを作成したり、ターゲットのインスタンス・メソッドを呼び出したり、ターゲットがネイティブの Python オブジェクトであるかのように容易にプロパティ値を取得または設定したりすることができます。

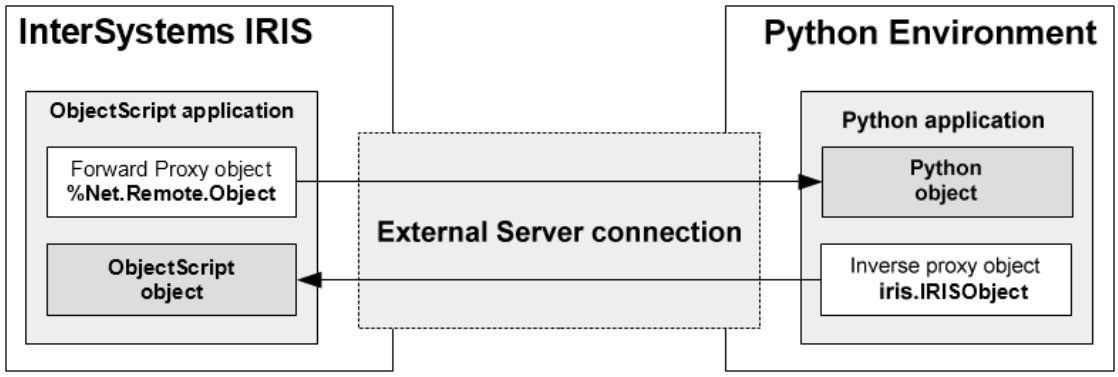
このセクションでは、以下のトピックについて説明します。

- Python 外部サーバの概要 ー 外部サーバの概要を示します。
- Python 逆プロキシ・オブジェクトの作成 ー 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。
- IRISObject によるデータベース・オブジェクトの制御 ー 逆プロキシ・オブジェクトの使用法を示します。

3.1 Python 外部サーバの概要

Python 外部サーバによって、InterSystems IRIS 埋め込み言語オブジェクトと Python Native SDK 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、これを、一方の側のプロキシ・オブジェクトをもう一方の側のターゲット・オブジェクトに接続する単純なブラック・ボックスとして考えることができます。

図 3-1: Python 外部サーバのシステム



図に示すように、フォワード・プロキシは、外部の Python ターゲット・オブジェクトを制御するデータベース・オブジェクトです。対応する Native SDK オブジェクトは、通常の制御フローを逆転させ、外部 Python アプリケーションからデータベースのターゲット・オブジェクトを制御する逆プロキシです。

3.2 Python 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、データベース・クラス・インスタンスの OREF を取得 (例えば、ObjectScript クラスの %New() メソッドを呼び出して) します。IRIS.classMethodObject() および IRIS.functionObject() メソッドは、その呼び出しが有効な OREF を取得する場合、IRISObject インスタンスを返します。以下の例では、IRIS.classMethodObject() を使用して、逆プロキシ・オブジェクトを作成します。

IRISObject のインスタンスの作成

```
proxy = irispy.classMethodObject("User.TestInverse","%New");
```

- classMethodObject() 呼び出しは、User.TestInverse という名前の ObjectScript クラスの %New() メソッドを呼び出します。
- %New() の呼び出しは、User.TestInverse のデータベース・インスタンスを作成します。
- classMethodObject() は、逆プロキシ・オブジェクト proxy を返します。これは、データベース・インスタンスにマップされた IRISObject のインスタンスです。

この例は、irispy が接続された IRIS のインスタンスであると想定しています (“Python での接続の作成” を参照してください)。クラス・メソッドの呼び出し方法の詳細は、“Python からのデータベース・メソッドおよび関数の呼び出し” を参照してください。

以下のセクションでは、proxy を使用して ObjectScript の User.TestInverse インスタンスのメソッドとプロパティにアクセスする方法を説明します。

3.3 IRISObject によるデータベース・オブジェクトの制御

iris.IRISObject クラスは、データベース・ターゲット・オブジェクトを制御するいくつかの方法を提供します。invoke() と invokeVoid() は、それぞれ戻り値ありとなしでインスタンス・メソッドを呼び出します。get() と set() は、それぞれプロパティ値を取得、設定します。

この例では、User.TestInverse という ObjectScript クラスを使用しています。これには、メソッド initialize() および add() とプロパティ name の宣言が含まれています。

ObjectScript サンプル・クラス TestInverse

```
Class User.TestInverse Extends %Persistent {
  Method initialize(initialVal As %String = "no name") {
    set ..name = initialVal
    return 0
  }
  Method add(val1 As %Integer, val2 As %Integer) As %Integer {
   return val1 + val2
  }
  Property name As %String;
}
```

次の例の 1 行目で、User.TestInverse の新しいインスタンスを作成し、逆プロキシ・オブジェクト proxy を返します。これはデータベース・インスタンスにマップされます。残りのコードでは、proxy を使用してターゲット・インスタンスにアクセスします。irispy という名前の IRIS の接続済みインスタンスが既に存在することが前提とされています (“Python での接続の作成” を参照)。

Python での逆プロキシ・オブジェクト・メソッドの使用

```
# Create an instance of User.TestInverse and return an inverse proxy object for it
proxy = irispy.classMethodObject("User.TestInverse","%New");

# instance method proxy.initialize() is called with one argument, returning nothing.
proxy.invokeVoid("initialize", "George");
print("Current name is "+ proxy.get("name"));   # display the initialized property value

# instance method proxy.add() is called with two arguments, returning an int value.
print("Sum of 2 plus 3 is " + str(proxy.invoke("add",2,3)));

# The value of property proxy.name is displayed, changed, and displayed again.
proxy.set("name", "Einstein, Albert");  # sets the property to "Einstein, Albert"
print("New name is "+ proxy.get("name"));   # display the new property value
```

この例では、以下のメソッドを使用して、User.TestInverse 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。

- IRISObject.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、プロパティを初期化しますが、値は返しません。
- IRISObject.invoke() は、インスタンス・メソッド add() を呼び出します。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、name プロパティを新しい値に設定します。
- IRISObject.get() は、name プロパティの値を返します。

この例では、get() および invoke() メソッドのバリアントを使用しましたが、IRISObject クラスでは、サポートされるデータ型用に型キャスト・メソッドも提供します:

IRISObject get() 型キャスト・メソッド

get() メソッドのバリアントに加えて、IRISObject は getBytes()、getDecimal()、getFloat()、getInteger()、getString()、getIRISList()、および getObject() の IRISObject. 型キャスト・メソッドを提供します。

IRISObject invoke() 型キャスト・メソッド

invoke() および invokeVoid() に加えて、IRISObject は invokeBytes()、invokeDecimal()、invokeInteger()、invokeString()、invokeIRISList()、および invokeObject() の IRISObject. 型キャスト・メソッド を提供します。

すべての invoke() メソッドは、methodName の str 引数に加え、0 個以上のメソッド引数を取ります。引数は、Python オブジェクトか、サポートされるデータ型の値のいずれかです。サポートされていない型の引数に対しては、データベース・プロキシが生成されます。

invoke() は、IRISObject インスタンスのインスタンス・メソッドのみ呼び出すことができます。クラス・メソッドの呼び出し方法については、“Python からのデータベース・メソッドおよび関数の呼び出し”を参照してください。

4

Python を使用したグローバル配列へのアクセス

Native SDK for Python はグローバル配列を操作するメカニズムを提供します。ここでは、以下の項目について説明します。

- グローバル配列の概要 ー グローバル配列の概念を示し、Native SDK がどのように使用されるかを簡単に説明します。
- 基本的なノード操作 ー set()、get()、および kill() を使用して、グローバル配列内のノードの作成、ノードへのアクセス、およびノードの削除を行う方法を説明します。
- nextSubscript() および isDefined() を使用した反復処理 ー ObjectScript スタイルの反復処理をエミュレートするメソッドを紹介します。

注釈 **Python でのデータベース接続の作成**

この章の例では、iris.IRIS オブジェクトである irispy が既に存在し、サーバに接続されていると想定します。irispy は、以下のコードを使用して作成および接続されます。

```
import iris
conn = iris.connect('127.0.0.1', 51773, 'USER', '_SYSTEM', 'SYS')
irispy = iris.createIRIS(conn)
```

詳細は、iris パッケージの関数 connect() および 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。

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。ディレクトリ構造とは異なり、グローバル配列内のすべてのノードに値または値を持つサブノードが必要です。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')
```

この表記では、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、ノードのノード・アドレ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。最初の引数が割り当てられる値で、それ以外の引数がノード・アドレスを指定します。

```
irispy.set('A', 'root', 'foo', 'SubFoo')
irispy.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') に割り当てるまで存在しません。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。

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、irispy という名前の iris.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

```
# Import the Native SDK module
import iris

# Open a connection to the server
args = {'hostname':'127.0.0.1', 'port':52773,
    'namespace':'USER', 'username':'_SYSTEM', 'password':'SYS'
}
conn = iris.connect(**args)
# Create an iris object
irispy = iris.createIRIS(conn)

# Create a global array in the USER namespace on the server
irispy.set('A', 'root', 'foo', 'SubFoo')
irispy.set(123, 'root', 'bar', 'lowbar', 'UnderBar')

# Read the values from the database and print them
subfoo_value = irispy.get('root', 'foo', 'SubFoo')
underbar_value = irispy.get('root', 'bar', 'lowbar', 'UnderBar')
print('Created two values: ')
print('   root("foo","SubFoo")=', subfoo_value)
print('   root("bar","lowbar","UnderBar")=', underbar_value)

# Delete the global array and terminate
irispy.kill('root') # delete global array root
conn.close()
```

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、データベースへの接続と irispy (iris.IRIS のインスタンスで、接続オブジェクトを含む) の作成に、Native SDK iris パッケージのメソッドが使用されています。Native SDK メソッドは以下のアクションを実行します。

- iris.connect() は、USER ネームスペースに関連付けられているデータベースに接続される、conn という名前の connection オブジェクトを作成します。
- iris.createIRIS() は、サーバ接続 conn を介してデータベースにアクセスする、irispy という名前の iris.IRIS の新しいインスタンスを作成します。

- iris.IRIS.set() は、データベース・ネームスペース USER に新しい永続ノードを作成します。
- iris.IRIS.get() は、指定されたノードの値を返します。
- iris.IRIS.kill() は、指定されたルート・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。
- iris.IRISConnection.close() は、接続を閉じます。

iris.IRIS のインスタンスの接続および作成の詳細は、“iris パッケージのメソッド” を参照してください。set()、get()、および kill() の詳細は、“基本的なノード操作” を参照してください。

この簡単な例には、反復処理といったより高度なトピックは含まれていません。複雑なグローバル配列の作成および反復処理の詳細は、“クラス iris.IRISGlobalNode” を参照してください。

4.1.1 グローバル配列の用語集

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、値を含むか、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で構成される、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)。添え字は、bool、bytes、bytearray、Decimal、float、int、str のいずれかです。

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

ノードに含めることができる値のタイプは、bool、bytes、bytearray、Decimal、float、int、str、IRISList、または None です ("型キャスト・メソッドおよびサポートされるデータ型" を参照)。子ノードを持つノードは値なしにできますが、子ノードのないノードには値を含める必要があります。

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、bool、bytes、bytearray、Decimal、float、int、str のいずれかです。文字列の添え字では大文字と小文字が区別され、あらゆる文字 (出力不能文字を含む) を使用できます。添え字の長さは、ノード・アドレスの最大長によってのみ制限されます。

4.2 基本的なノード操作

ここでは、set() メソッド、get() メソッド、および kill() メソッドを使用して、ノードの作成、ノードへのアクセス、およびノードの削除を行う方法について説明します。これらのメソッドには、以下のシグニチャがあります。

```
set (value, globalName, subscripts)
get (globalName, subscripts)
kill (globalName, subscripts)
```

- value は、bool、bytes、bytearray、Decimal、float、int、str、IRISList、None のいずれかです。
- globalName 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。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。
- subscripts は、bool、bytes、bytearray、Decimal、float、int、str のいずれかです。文字列の添え字では大文字と小文字が区別され、出力不能文字を含めることができます。

このセクションのすべての例で、irispy という名前の IRIS の接続済みインスタンスが既に存在すると想定しています (“Python での接続の作成” を参照)。

ノード値の設定および変更

iris.IRIS.set() は、value 引数、globalname 引数、および *subscripts 引数を取り、指定されたノード・アドレスにその値を格納します。その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。

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。

```
irispy.set('first','myGlobal','A')  # create node myGlobal('A') = 'first'
irispy.set(1,'myGlobal','A')        # change value of myGlobal('A') to 1.
```

get() を使用したノード値の取得

iris.IRIS.get() は、globalname 引数および *subscripts 引数を取り、指定されたノード・アドレスに格納されている値を返すか、そのアドレスに値が存在しない場合は None を返します。

```
irispy.set(23,'myGlobal','A')
value_of_A = irispy.get('myGlobal','A')
```

get() メソッドは、決まった形式のない値を返します。特定のデータ型を返すには、いずれかの IRIS.get() 型キャスト・メソッドを使用します。使用可能なメソッドは、getBoolean()、getBytes()、getDecimal()、getFloat()、getInteger()、getString()、getIRISList()、および getObject() です。

ノードまたはノード・グループの削除

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

グローバル配列 myGlobal には、最初に以下のノードが含まれます。

```
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、その 2 つのサブノードで kill() を呼び出すことで、グローバル配列を削除します。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。

```
irispy.kill('myGlobal','A')    # also kills myGlobal('A',1) and myGlobal('A',2)
```

2 番目の呼び出しによって、値を持つ最後のサブノードが削除され、グローバル配列全体が削除されます。

```
irispy.kill('myGlobal','B',1)  # deletes last value in global array myGlobal
```

- 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。
- 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 nextSubscript() および isDefined() を使用した反復処理

ObjectScript での標準の反復処理メソッドは $ORDER および $DATA です。Native SDK には、これらの ObjectScript メソッドをエミュレートするユーザ向けに、対応するメソッド nextSubscript() および isDefined() が用意されています。

IRIS.nextSubscript() メソッド (`$ORDER` に対応) は、反復処理メソッドとしては node() ほど強力ではありませんが、ほぼ同じように機能し、同一の親の下の一連のノードを反復処理します。ノード・アドレスと反復処理の方向が指定されると、このメソッドは、指定されたノードと同じ親の次のノードの添え字を返すか、指定された方向にこれ以上ノードがない場合は None を返します。

IRIS.isDefined() メソッド (`$DATA` に対応) は、指定されたノードに値、サブノード、またはその両方があるかどうかの確認に使用できます。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。

```
exists = (irispy.isDefined(root,subscripts) > 0)
hasValue = (irispy.isDefined(root,subscripts) in [1,11]) # [value, value+child]
hasChild = (irispy.isDefined(root,subscripts) in [10,11]) # [child, value+child]
```

兄弟ノードの検索および子の有無のテスト

以下のコードでは、nextSubscript() を使用して、heroes('dogs') のノードに対し、heroes('dogs',chr(0)) (考えられる最初の添え字) から開始して反復処理が行われます。各ノードを isDefined() でテストして、子があるかどうか確認します。

```
direction = 0 # direction of iteration (boolean forward/reverse)
next_sub = chr(0) # start at first possible subscript
while next_sub != None:
  if (irispy.isDefined('heroes','dogs',next_sub) in [10,11]): # [child, value+child]
    print('   ', next_sub, 'has children')
  next_sub = irispy.nextSubscript(direction,'heroes','dogs',next_sub)
  print('next subscript = ' + str(next_sub) )
```

出力:

```
next subscript = Balto
next subscript = Hachiko
next subscript = Lassie
    Lassie has children
next subscript = Whitefang
next subscript = None
```

5

Python によるトランザクションの管理とロック

Native SDK for Python は、インターシステムズのトランザクション・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処理 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。

インターシステムズのトランザクション・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。

5.1 Python でのトランザクションの処理

iris.IRIS クラスは、トランザクション処理のための以下のメソッドを提供します。

- 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 (トランザクション内でない場合は 0) を返します。
- IRIS.increment() ー ノードをロックすることなく、ノード値をインクリメントまたはデクリメントします。
- 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・レベルで異なるグローバル・ノードの値を保存します。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。また、この例では、トランザクション時に 2 つのカウンタをインクリメントして、increment() メソッドと += 演算子の違いを示します。

注釈　この例では、新しいストレージ・クラスを作成することなく、データベースに値を保存する便利な方法として、グローバル配列を使用します。この例に直接関係のないグローバル配列操作は、このメインの例の直後にリストされたユーティリティ関数に分離されています。

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作成とロールバック

irispy が接続された iris.IRIS のインスタンスであるとします ("Python での接続の作成" を参照してください)。

グローバル配列ユーティリティ関数 store_values()、show_values()、start_counters()、および show_counters() は、この例の直後にリストされています（“グローバル配列ユーティリティ関数”を参照してください）。

```
  tlevel = irispy.getTLevel()
  counters = start_counters()
  action = 'Initial values:'.ljust(18,' ') + 'tLevel='+str(tlevel)
  print(action + ', ' + show_counters() + ', ' + show_values() )

  print('\nStore three values in three nested transaction levels:')
  while tlevel < 3:
    irispy.tStart()   # start a new transaction, incrementing tlevel by 1
    tlevel = irispy.getTLevel()
    store_values(tlevel)
    counters['add'] += 1                              # increment with +=
    irispy.increment(1,counters._global_name, 'inc')  # call increment()
    action = '  tStart:'.ljust(18,' ') + 'tLevel=' + str(tlevel)
    print(action + ', ' + show_counters() + ', ' + show_values() )

  print('\nNow roll back two levels and commit the level 1 transaction:')
  while tlevel > 0:
    if (tlevel>1):
      irispy.tRollbackOne()         # roll back to level 1
      action = '  tRollbackOne():'
    else:
      irispy.tCommit()              # commit level 1 transaction
      action = '  tCommit():'
    tlevel = irispy.getTLevel()
    action = action.ljust(18,' ') + 'tLevel=' + str(tlevel)
    print(action + ', ' + show_counters() + ', ' + show_values() )
```

出力:

```
Initial values:   tLevel=0, add=0/inc=0, values=[]

Store three values in three nested transaction levels:
  tStart:         tLevel=1, add=1/inc=1, values=["data1"]
  tStart:         tLevel=2, add=2/inc=2, values=["data1", "data2"]
  tStart:         tLevel=3, add=3/inc=3, values=["data1", "data2", "data3"]

Now roll back two levels and commit the level 1 transaction:
  tRollbackOne(): tLevel=2, add=2/inc=3, values=["data1", "data2"]
  tRollbackOne(): tLevel=1, add=1/inc=3, values=["data1"]
  tCommit():      tLevel=0, add=1/inc=3, values=["data1"]
```

グローバル配列ユーティリティ関数

この例では、新しいストレージ・クラスを作成することなく、データベースに値を保存する便利な方法として、グローバル配列を使用します（“Python を使用したグローバル配列へのアクセス”を参照してください）。次の関数では、data_node と counter_node という 2 つの IRISGlobalNode オブジェクトを使用して、永続データを保存および取得します。

data_node オブジェクトは、トランザクション値の保存および取得に使用されます。トランザクションごとに、添え字としてレベル番号を使用して、別個の子ノードが作成されます。

```
irispy.kill('my.data.node') # delete data from previous tests
data_node = irispy.node('my.data.node')  # create IRISGlobalNode object

def store_values(tlevel):
  ''' store data for this transaction using level number as the subscript '''
  data_node[tlevel] = '"data'+str(tlevel)+'"'  # "data1", "data2", etc.

def show_values():
  ''' display values stored in all subnodes of data_node '''
  return 'values=[' + ", ".join([str(val) for val in data_node.values()]) + ']'
```

increment() メソッドは、一般にデータベースへの新しいエントリの追加を試みる前に呼び出され、ノードをロックしなくても、カウンタが迅速かつ安全にインクリメントされるようにします。インクリメントされたノードの値は、トランザクションのロールバックによる影響を受けません。

これを示すために、counter_node オブジェクトを使用して、2 つのカウンタ値を管理します。counter_node('add') サブノードは標準の += 演算子を使用してインクリメントされ、counter_node('inc') サブノードは increment() メソッドを使用してインクリメントされます。counter_node('add') 値とは異なり、counter_node('inc') 値はロールバック後もその値が維持されます。

```
# counter_node object will manage persistent counters for both increment methods
irispy.kill('my.count.node') # delete data left from previous tests
counter_node = irispy.node('my.count.node')  # create IRISGlobalNode object

def start_counters():
  ''' initialize the subnodes and return the IRISGlobalNode object '''
  counter_node['add'] = 0  # counter to be incremented by += operator
  counter_node['inc'] = 0  # counter to be incremented by IRIS.increment()
  return counter_node

def show_counters():
  ''' display += and increment() counters side by side: add=#/inc=#  '''
  return 'add='+str(counter_node['add'])+'/inc='+str(counter_node['inc'])
```

5.2 Python による並列処理の制御

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 (″ObjectScript リファレンス″ の “LOCK” を参照してください)。

クラス iris.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。

- IRIS.lock() ー lockReference および *subscripts 引数で指定されたノードをロックします。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 lockReference および *subscripts 引数で指定されたノードのロックを解放します。
- 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。

パラメータ :

```
lock(lockMode, timeout, lockReference, *subscripts)
unlock(lockMode, lockReference, *subscripts)
releaseAllLocks()
```

- lockMode ー 以前に保持されていたロックを処理する方法を指定する str。有効な引数は、共有ロックを表す S、エスカレート・ロックを表す E、共有とエスカレートの両方を表す SE です。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。この秒数を経過するとタイムアウトになります。
- lockReference ー 曲折アクセント記号 (^) で始まり、その後にグローバル名が続く str (例えば、単なる `myGlobal` ではなく `^myGlobal`)。

  重要:ほとんどのメソッドで使用される `globalName` パラメータとは異なり、`lockReference` パラメータには、先頭に曲折アクセント記号を付ける必要があります。`globalName` ではなく `lockReference` を使用するのは、lock() および unlock() のみです。
- subscripts ー ロックまたはアンロックするノードを指定する 0 個以上の添え字。

これらのメソッドに加え、すべてのロックとその他の接続リソースを解放するために、IRISConnection.close() メソッドが使用されます。

Tipヒント　管理ポータルを使用して、ロックを検証できます。System Operation > View 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。

注釈　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、以下の項目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

Python DB-API の使用法

InterSystems Python DB-API ドライバは、PEP 249 バージョン 2.0 Python Database API の仕様に完全に準拠した実装です。以下のセクションでは、必要なすべての実装機能をリストして、それぞれのサポート・レベルを示し、インターシステムズ固有の機能すべてについて詳細に説明します。

- 使用法

  InterSystems IRIS への接続方法、および Cursor オブジェクトの取得方法について説明します。

- PEP 249 実装リファレンス

  以下に、PEP 249 のすべての要件と実装の詳細を示します。

  - "グローバル" では、必要なグローバル定数 apilevel、threadsafety、および paramstyle の値を示します。
  - "Connection オブジェクト" では、Connection メソッド connect()、close()、commit()、rollback()、および cursor() について説明します。
  - "Cursor オブジェクト" では、次の Cursor メンバについて説明します。
    - 属性 arraysize、description、および rowcount。
    - 標準メソッド callproc()、close()、execute()、executemany()、fetchone()、fetchmany()、fetchall()、nextset()、scroll()、setinputsizes()、および setoutputsize()。
    - インターシステムズの拡張メソッド isClosed() および stored_results()。

- SQLType 列挙値

  有効な SQLType 列挙定数。

注釈 **DB-API ドライバのインストール**

DB-API は、InterSystems IRIS をインストールすると使用できるようになります。InterSystems DB-API ドライバがない場合 (InterSystems IRIS がインストールされていないホストから接続されている場合など)、InterSystems IRIS ドライバのページからこれをダウンロードし、以下を使用してインストールできます。

```
pip install intersystems_irispython-3.2.0-py3-none-any.whl
```

6.1 使用法

次の例では、InterSystems IRIS データベースに接続し、その接続に関連付けられたカーソルを作成して、DB-API 呼び出しを設定し、シャットダウンします。

使用可能なすべての DBAPI メソッドに関する詳細なドキュメントは、次のセクションの“Connection オブジェクト”および“Cursor オブジェクト”を参照してください。

DB-API ドライバへの接続とカーソルの取得

Python

```
import iris

def main():
  connection_string = "localhost:1972/USER"
  username = "_system"
  password = "SYS"

  connection = iris.connect(connection_string, username, password)
  cursor = connection.cursor()

  try:
    pass  # do something with DB-API calls
  except Exception as ex:
    print(ex)
  finally:
    if cursor:
      cursor.close()
    if connection:
      connection.close()

if __name__ == "__main__":
  main()
```

この例で呼び出されるメソッドの詳細は、"iris.connect()"、"Connection.close()"、"Connection.cursor()"、および "Cursor.close()" を参照してください。

"Connecting Your Application to InterSystems IRIS" にも、サンプル・コードを含め、DB-API を使用して Python アプリケーションから InterSystems IRIS サーバに接続する手順が示されています。

6.2 PEP 249 実装リファレンス

このセクションでは、PEP 249 バージョン 2.0 Python Database API の仕様で説明されているすべての必須実装機能をリストして、それぞれのサポート・レベルを示し、インターシステムズ固有の機能すべてについて詳細に説明します。

6.2.1 グローバル

これらは、必須の実装固有定数です。インターシステムズの実装では、これらのグローバルは次の値に設定されます。

apilevel

`"2.0"` ― PEP 249 バージョン 2.0 に準拠することを指定します。

threadsafety

`0` ― スレッドはモジュールを共有できません。

paramstyle

"qmark" — クエリ・パラメータは疑問符のスタイルを使用します (例: WHERE name=?)。

6.2.2 Connection オブジェクト

このセクションでは、iris.connect() を使用して Connection オブジェクトを作成する方法を説明し、必須 Connection メソッド close()、commit()、rollback()、および cursor() の実装の詳細を示します。

6.2.2.1 Connection オブジェクトの生成

DB-API Connection オブジェクトは、InterSystems iris.connect() メソッドを呼び出すことにより作成されます。

Connect()

iris.connect() は、新しい Connection オブジェクトを返し、InterSystems IRIS のインスタンスへの新規接続の作成を試みます。接続が成功するとオブジェクトは開かれ、それ以外の場合は閉じられます (Cursor.isClosed() を参照)。

```
iris.connect(hostname,port,namespace,username,password,timeout,sharedmemory,logfile)
iris.connect(connectionstr,username,password,timeout,sharedmemory,logfile)
```

最後に成功した接続試行の hostname、port、namespace、timeout、および logfile が Connection オブジェクトのプロパティとして保存されます。

パラメータ:

パラメータは位置またはキーワードによって渡すことができます。

- hostname — サーバ URL を指定する str
- port — スーパーサーバ・ポート番号を指定する int
- namespace — ネームスペース・サーバを指定する str
- 以下のパラメータを、hostname、port、および namespace の引数の代わりに使用できます。
  - connectionstr — *hostname*:*port*/*namespace* の形式の str
- username — ユーザ名を指定する str
- password — パスワードを指定する str
- timeout (オプション) — 接続試行時に待機する最大秒数を指定する int既定値は 10 です。
- sharedmemory (オプション) — bool を True に指定すると、ホスト名が localhost または 127.0.0.1 の場合に共有メモリ接続を試行します。False に指定すると、TCP/IP 経由の接続を強制します。既定値は True です。
- logfile (オプション) — クライアント側のログ・ファイル・パスを指定する strパスの最大長は、255 文字の ASCII 文字です。

6.2.2.2 Connection オブジェクト・メソッド

Connection オブジェクトは、1 つ以上の Cursor オブジェクトの作成に使用できます。1 つのカーソルにより行われたデータベースの変更は、同じ接続から作成されたその他すべてのカーソルに即座に表示されます。ロールバックとコミットは、この接続を使用するカーソルによって行われたすべての変更に影響を与えます。

close()

Connection.close() は、即座に接続を閉じます。接続とその接続に関連付けられたすべてのカーソルは使用できなくなります。関連付けられたカーソルによるコミットされていないすべての変更に対して、暗黙的なロールバックが実行されます。

```
Connection.close()
```

閉じられた接続で、または関連するカーソルで何らかの操作が試みられると、ProgrammingError 例外が発生します。

Commit()

Connection.commit() は、前回のコミット以降にこの接続で実行されたすべての SQL 文をコミットします。ロールバックは、この接続を使用する任意のカーソルによって行われたすべての変更に影響を与えます。このメソッドへの明示的な呼び出しは必要ありません。

```
Connection.commit()
```

Rollback()

Connection.rollback() は、(前回のコミットまたはロールバック以降に) このカーソルを作成した接続で実行されたすべての SQL 文をロールバックします。ロールバックは、この接続を使用する任意のカーソルによって行われたすべての変更に影響を与えます。

```
Connection.rollback()
```

cursor()

Connection.cursor() は、この接続を使用する、新しい Cursor オブジェクトを返します。

```
Connection.cursor()
```

1 つのカーソルにより行われたデータベースへの任意の変更は、同じ接続から作成されたその他すべてのカーソルに即座に表示されます。ロールバックとコミットは、この接続を使用する任意のカーソルによって行われたすべての変更に影響を与えます。

6.2.3 Cursor オブジェクト

このセクションでは、Cursor オブジェクトの作成方法について説明し、以下の必須 Cursor メソッドおよび属性の実装の詳細を示します。

- 属性 arraysize、description、および rowcount。
- 標準メソッド callproc()、close()、execute()、executemany()、fetchone()、fetchmany()、fetchall()、nextset()、scroll()、setinputsizes()、および setoutputsize()。
- インターシステムズの拡張メソッド isClosed() および stored_results()。

6.2.3.1 Cursor オブジェクトの作成

Cursor オブジェクトは、接続を確立した後に Connection.cursor() を呼び出すことにより作成されます。例:

```
connection = iris.connect(connection_string, username, password)
cursor = connection.cursor()
```

1 つのカーソルにより行われたデータベースへの任意の変更は、同じ接続から作成されたその他すべてのカーソルに即座に表示されます。

完全な例は、“DB-APIドライバへの接続とカーソルの取得”を参照してください。接続の作成の詳細は、“Connectionオブジェクトの生成”を参照してください。

6.2.3.2 Cursor の属性

arraysize

Cursor.arraysize は、fetchmany() により一度にフェッチする行数を指定する読み取り/書き込み属性です。既定値は 1 (一度に 1 行をフェッチする) です。

description

Cursor.description は、前回の SQL select 文で返された各結果の列の情報を含むタプルのリストを返します。実行メソッドが呼び出されなかった場合、または前回の操作で行が返されなかった場合、値は None になります。

リスト内の各タプル (列の説明) には、以下の項目が含まれます。

- name ー 列名 (既定値は None)
- type_code ー 整数の SQLType 識別子 (既定値は 0)。有効な値は、“SQLType 列挙値”を参照してください。
- display_size ー 未使用 – 値は None に設定されます。
- internal_size ー 未使用 – 値は None に設定されます。
- precision ー 整数 (既定値は 0)
- scale ー 整数 (既定値は None)
- nullable ー 整数 (既定値は 0)

rowcount

Cursor.rowcount は、前回の SQL 文で変更された行数を指定します。SQL が実行されていない、または行数が不明である場合、値は -1 となります。例えば、CREATE、DROP、DELETE、SELECT 文などの DDL は (パフォーマンス上の理由で) -1 を返します。

バッチ更新でも影響を受ける行数が返されます。

6.2.3.3 Cursor メソッド

callproc()

Cursor.callproc() は、指定された procname のストアド・データベース・プロシージャを呼び出します。

```
Cursor.callproc(procname)
Cursor.callproc(procname, parameters)
```

パラメータ:

- procname – パラメータ化された引数を使用したストアド・プロシージャ呼び出しを含む文字列。
- parameters – ストアド・プロシージャに渡すパラメータ値の list

例:

このコードは、list にパラメータ値 "A" を指定して、ストアド・プロシージャ Sample.SP_Sample_By_Name を呼び出します。

```
cursor.callproc("CALL Sample.SP_Sample_By_Name (?)", ["A"])
row = cursor.fetchone()
while row:
   print(row.ID, row.Name, row.DOB, row.SSN)
   row = cursor.fetchone()
```

出力は以下のようになります。

```
167 Adams,Patricia J. 1964-10-12 216-28-1384
28 Ahmed,Dave H. 1954-01-12 711-67-4091
20 Alton,Samantha E. 2015-03-28 877-53-4204
118 Anderson,Elvis V. 1994-05-29 916-13-245
```

close()

Cursor.close() は、カーソルをクローズします。

```
Cursor.close()
```

閉じられたカーソルで何らかの操作が試みられると、ProgrammingError 例外が発生します。カーソルは削除される (一般的には範囲外になる) と自動的に閉じられるため、これを呼び出すことは通常必要ありません。

execute()

Cursor.execute() は、operation パラメータで指定されたクエリを実行します。Cursor オブジェクトを更新し、rowcount 属性を、クエリの場合は -1 に、更新の場合は 1 に設定します。

```
Cursor.execute(operation)
Cursor.execute(operation, parameters)
```

パラメータ:

- operation – 実行する SQL 文を含む string
- parameters – オプションの値の list。これは Python list である必要があります (タプルまたはセットは受け入れられません)。

例:

パラメータ値は、SQL 文にリテラルや定数ではなく ? (疑問符) が含まれている位置で使用されます。文に疑問符が含まれていない場合、parameters 引数は必要ありません。指定された場合は例外が発生します。

- `sql = "...(1,2)..."; execute(sql)`
- `sql = "...(?,?)..."; params = [1,2]; execute(sql, params)`
- `sql = "...(1,?)..."; params = [2]; execute(sql, params)`

executemany()

Cursor.executemany() は、バッチの挿入または更新に使用されます。これは、データベース操作 (クエリまたはコマンド) を準備し、すべてのパラメータ・シーケンス、またはシーケンス seq_of_parameters 内にあるマッピングに対してこれを実行します。

```
Cursor.executemany(operation)
Cursor.executemany(operation, seq_of_parameters)
```

パラメータ:

- operation – 実行する SQL 文を含む文字列

- `seq_of_parameters` – パラメータ・シーケンスまたはマッピングのシーケンス

fetchone()

Cursor.fetchone() は、クエリ内の次の ResultSetRow.DataRow オブジェクト（データ・オフセットの整数の配列）、またはこれ以上データがない場合 `None` を返します。

```
Cursor.fetchone()
```

データは要求に応じてのみ、インデックス化によってフェッチされます。オブジェクトには、行の値を取得するために使用できる整数のオフセットのリストが含まれます。インデックスの値は正の整数である必要があります（1 という値は列 1 を示し、以下同様です）。

SQL が実行されていない、または結果セットが返されていない場合（SELECT 文ではなかった場合など）、ProgrammingError 例外が発生します。

fetchmany()

Cursor.fetchmany() は、クエリの次の行の結果セットをフェッチし、シーケンスのシーケンス（タプルのリスト）を返します。size 引数が指定されていない場合、一度にフェッチする行数は、Cursor.arraysize 属性で設定されます（既定値は `1`）。これ以上使用できる行がない場合は空のシーケンスが返されます。

```
Cursor.fetchmany()
Cursor.fetchmany(size)
```

パラメータ:

- `size` – オプション。既定値は、属性 Cursor.arraysize の現在の値です。

fetchall()

Cursor.fetchall() は、クエリ結果の残りの行すべてをフェッチします。

```
Cursor.fetchall()
```

isClosed() [InterSystems extension method]

Cursor.isClosed() は Cursor オブジェクトが既に閉じられている場合に `True`、それ以外の場合に `False` を返す、インターシステムズの拡張メソッドです。

```
Cursor.isClosed()
```

nextset() [optional DB-API method]

Cursor.nextset() は、複数の結果セットで反復するためのオプションの DB-API メソッドです。次の結果セットがあれば、そのセットにスキップします。結果セットがある場合は `True`、ない場合は `False` を返します（したがって、結果セットや結果セット行にアクセスするためには使用しないでください）。

```
Cursor.nextset()
```

例:

```
for row in cursor.stored_results():
  row_values = row[0] // data in all columns
  val1 = row[1]       // data in column 1
  cursor.nextset()    // skips to the next result set if multiple result sets
  // does nothing (or breaks out of loop) in case of single result set;
```

scroll() [optional DB-API method]

Cursor.scroll() は、結果セット内でカーソルを新しい位置までスクロールして、その位置の行を返す、オプションの DB-API メソッドです。

```
Cursor.scroll(value)
Cursor.scroll(value, mode)
```

スクロール操作で結果セットを残したままにすると、IndexError が発生します。

パラメータ:

- `value` – 新しいターゲット位置を指定する整数値。
  - mode が `relative` (既定値) の場合、value は、結果セット内の現在の位置に対する正または負のオフセットです。
  - mode が `absolute` の場合、value は、ターゲットの絶対位置です (負の値は無効です)。
- `mode` – オプション。有効な値は、`relative` または `absolute` です。

例:

例えば結果セットが全部で `10` 行あり、フェッチされる行数の初期値が `5` だとします。結果セットのインデックス値は 0 から始まるため、現在の位置は `rs[4]` (5 行目) となります。

```
# Scroll forward 3 rows relative to rs[4] (mode defaults to 'relative')
   datarow = Cursor.scroll(3)             // returns rs[7] (8th row in resultset)
# Scroll to absolute position 3
   datarow = Cursor.scroll(3,'absolute')  // returns rs[2]  (3rd row in resultset)
# Scroll backward 4 rows relative to rs[4] (mode defaults to 'relative')
   datarow = Cursor.scroll(-4)            // returns rs[0]
# Scroll to absolute position -4
   datarow = Cursor.scroll(-4,'absolute') // Throws IndexError (negative row number not valid)
```

setinputsizes()

Cursor.setinputsizes() は InterSystems IRIS には適用できません。ここではこの機能は実装されず、必要とされません。呼び出された場合、NotImplementedError をスローします。

setoutputsize()

Cursor.setoutputsize() は InterSystems IRIS には適用できません。ここではこの機能は実装されず、必要とされません。呼び出された場合、NotImplementedError をスローします。

stored_results() [InterSystems extension method]

Cursor.stored_results() は、プロシージャ・タイプが `'query'` の場合はリスト反復子 (各結果セットの最初の行を含む) を返し、プロシージャ・タイプが `'function'` の場合は空のリストを返す、インターシステムズの拡張メソッドです。

```
Cursor.stored_results()
```

例:

```
for row in cursor.stored_results(): // row is DataRow object for 1st row of result set
  row_values = row[0] // data in all columns
  val1 = row[1] // data in column 1

Incorrect Syntax:
  row = cursor.stored_results() // row values not accessible using row[0] since it is a list
iterator
```

6.3 SQLType 列挙値

Cursor.description 属性の有効な値。

- BIGINT = -5
- BINARY = -2
- BIT = -7
- CHAR = 1
- DECIMAL = 3
- DOUBLE = 8
- FLOAT = 6
- GUID = -11
- INTEGER = 4
- LONGVARBINARY = -4
- LONGVARCHAR = -1
- NUMERIC = 2
- REAL = 7
- SMALLINT = 5
- DATE = 9
- TIME = 10
- TIMESTAMP = 11
- TINYINT = -6
- TYPE_DATE = 91
- TYPE_TIME = 92
- TYPE_TIMESTAMP = 93
- VARBINARY = -3
- VARCHAR = 12
- WCHAR = -8
- WLONGVARCHAR = -10
- WVARCHAR = -9
- DATE_HOROLOG = 1091
- TIME_HOROLOG = 1092
- TIMESTAMP_POSIX = 1093

7

Native SDK for Python のクイック・リファレンス

これは、InterSystems IRIS Native SDK for Python のクイック・リファレンスで、以下のクラスに関する情報を提供します。

- パッケージ iris
  - iris Package のメソッド ― IRIS のインスタンスを接続および作成します。
  - クラス IRIS は、Native SDK の主なエントリ・ポイントです。
  - クラス IRISConnection は、IRIS インスタンスをデータベースに接続します。
  - 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 iris.IRISGlobalNode は、グローバル配列を操作します。
  - クラス iris.IRISGlobalNodeView は、子ノードのディクショナリのようなビューです。
  - クラス IRISReference は、メソッドの引数を参照渡しします。
  - クラス IRISObject は、外部サーバの逆プロキシ・オブジェクトのためのメソッドを提供します。

- 従来のサポート対象クラス
  - クラス LegacyIterator 反復子メソッド。以前の実装との互換性のために維持されています。
  - クラス IRISNative 接続メソッド。以前の実装との互換性のために維持されています。

型キャスト・メソッドおよびサポートされるデータ型

Native SDK では、データ型 bool、bytes、bytearray、Decimal、float、int、str、IRISList、および None がサポートされます。

多くの場合、クラスには、既定の値タイプを返す汎用メソッド (多くの場合 get() と呼ばれる) に加えて、その汎用メソッドとまったく同じように機能するだけでなく、返り値を特定のデータ型にキャストする一連の型キャスト・メソッドが含まれます。例えば、IRIS.get() の後ろには、getBoolean() や getBytes() といった型キャスト・メソッドが続きます。

ほぼ同じリストを多数提示することでこのクイック・リファレンスが長くならないように、すべての型キャスト・メソッドを汎用バージョン (IRIS.get() 型キャスト・メソッドなど) の後にひとまとめにして示しています。

一連の型キャスト・メソッドは、IRIS.classMethodValue()、IRIS.function()、IRIS.get()、IRISList.get()、IRISObject.get()、IRISObject.invoke()、および IRISReference.getValue() で使用できます。

7.1 iris パッケージのメソッド

iris パッケージには、接続を作成するためのメソッド connect()、およびその接続を使用するクラス IRIS の新しいインスタンスを作成するためのメソッド createIRIS() が含まれています。Native SDK を使用するには、接続された iris.IRIS のインスタンスが必要です。

7.1.1 Python での接続の作成

以下のコードは、データベース接続を開き、irispy. という名前の iris.IRIS のインスタンスを作成する方法を示しています。このドキュメントのほとんどの例で、接続済みの irispy のインスタンスが既に存在すると想定しています。

```
import iris

# Open a connection to the server
args = {'hostname':'127.0.0.1', 'port':52773,
    'namespace':'USER', 'username':'_SYSTEM', 'password':'SYS'
}
conn = iris.connect(**args)
# Create an iris object
irispy = iris.createIRIS(conn)
```

iris.connect() および iris.createIRIS() の詳細は、以下のセクションを参照してください。

7.1.2 iris パッケージのメソッドの詳細

connect()

iris.connect() は、新しい IRISConnection オブジェクトを返し、データベースへの新規接続の作成を試みます。このオブジェクトは、接続が成功すると開かれます。接続を確立できないと、例外がスローされます。

```
iris.connect(hostname,port,namespace,username,password,timeout,sharedmemory,logfile)
iris.connect(connectionstr,username,password,timeout,sharedmemory,logfile)
```

最後に成功した接続試行の hostname、port、namespace、timeout、および logfile が接続オブジェクトのプロパティとして保存されます。

パラメータ：

パラメータは位置またはキーワードによって渡すことができます。

- `hostname` ― サーバ URL を指定する str
- `port` ― スーパーサーバ・ポート番号を指定する int
- `namespace` ― ネームスペース・サーバを指定する str
- 以下のパラメータを、`hostname`、`port`、および `namespace` の引数の代わりに使用できます。
  - `connectionstr` ― *`hostname:port/namespace`* の形式の str
- `username` ― ユーザ名を指定する str
- `password` ― パスワードを指定する str
- `timeout` (オプション) ― 接続試行時に待機する最大ミリ秒数を指定する int既定値は 10000 です。
- `sharedmemory` (オプション) ― bool を `True` に指定すると、ホスト名が `localhost` または `127.0.0.1` の場合に共有メモリ接続を試行します。`False` に指定すると、TCP/IP 経由の接続を強制します。既定値は `True` です。

- `logfile` (オプション) — クライアント側のログ・ファイル・パスを指定する strパスの最大長は、255 文字の ASCII 文字です。

createIRIS()

iris.createIRIS() は、指定された IRISConnection を使用する IRIS の新規インスタンスを返します。接続が閉じられている場合は、例外をスローします。

```
iris.createIRIS(conn)
```

返り値 : iris. の新しいインスタンスIRIS

パラメータ :

- `conn` — サーバ接続を提供する IRISConnection オブジェクト

7.2 クラス iris.IRIS

Native SDK を使用するには、アプリケーションで、データベースに接続された IRIS のインスタンスを作成する必要があります。IRIS のインスタンスは、iris パッケージのメソッド createIRIS() を呼び出すことによって作成されます。

7.2.1 IRIS メソッドの詳細

classMethodValue()

IRIS.classMethodValue() は、クラス・メソッドを呼び出して、0 個以上の引数を渡し、ObjectScript の返り値のデータ型に対応するタイプとして値を返します。ObjectScript の返り値が空の文字列 ($$$NULLOREF) の場合は、None を返します。詳細と例は、“Python からのクラス・メソッドの呼び出し” を参照してください。

```
classMethodValue (className, methodName, args)
```

返り値 : bool、bytes、bytearray、int、float、Decimal、str、IRISList、または None。特定のタイプを返す方法は、″classMethodValue() 型キャスト・メソッド″ を参照してください。

パラメータ :

- `class_name` — 呼び出されるメソッドが属するクラスの完全修飾名。
- `method_name` — クラス・メソッドの名前。
- `*args` — 0 個以上のメソッド引数。タイプが bool、bytes、Decimal、float、int、str、および IRISList の引数は、リテラルとして投影されます。その他のタイプはすべて、プロキシ・オブジェクトとして投影されます。

classMethodValue() に似ていて値を返さない ″classMethodVoid()″ も参照してください。

classMethodValue() 型キャスト・メソッド

以下に示す IRIS.classMethodValue() 型キャスト・メソッドのすべてが、IRIS.classMethodValue() とまったく同じように機能するだけでなく、返り値を特定のタイプにキャストします。ObjectScript の返り値が空の文字列 ($$$NULLOREF) の場合、これらはすべて None を返します。詳細と例は、“Python からのクラス・メソッドの呼び出し” を参照してください。

```
classMethodBoolean (className, methodName, args)
classMethodBytes (className, methodName, args)
classMethodDecimal (className, methodName, args)
classMethodFloat (className, methodName, args)
classMethodInteger (className, methodName, args)
classMethodIRISList (className, methodName, args)
classMethodString (className, methodName, args)
classMethodObject (className, methodName, args)
```

返り値 : bool、bytes、bytearray、int、float、Decimal、str、IRISList、Object、または None。

パラメータ :

- `class_name` ― 呼び出されるメソッドが属するクラスの完全修飾名。
- `method_name` ― クラス・メソッドの名前。
- `*args` ― サポートされるタイプの 0 個以上のメソッド引数。タイプが bool、bytes、Decimal、float、int、str、および IRISList の引数は、リテラルとして投影されます。その他のタイプはすべて、プロキシ・オブジェクトとして投影されます。

classMethodValue() に似ていて値を返さない ″classMethodVoid()″ も参照してください。

classMethodVoid()

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。詳細と例は、“Python からのクラス・メソッドの呼び出し” を参照してください。

```
classMethodVoid (className, methodName, args)
```

パラメータ :

- `class_name` ― 呼び出されるメソッドが属するクラスの完全修飾名。
- `method_name` ― クラス・メソッドの名前。
- `*args` ― サポートされるタイプの 0 個以上のメソッド引数。タイプが bool、bytes、Decimal、float、int、str、および IRISList の引数は、リテラルとして投影されます。その他のタイプはすべて、プロキシ・オブジェクトとして投影されます。

このメソッドは、返り値がないことを想定していますが、任意のクラス・メソッドの呼び出しに使用できます。classMethodVoid() を使用して、値を返すメソッドを呼び出した場合、メソッドは実行されますが、返り値は無視されます。

close()

IRIS..close() は IRIS オブジェクトを閉じます。

```
close ()
```

function()

IRIS.function() は、関数を呼び出して、0 個以上の引数を渡し、ObjectScript の返り値のデータ型に対応するタイプとして値を返します。ObjectScript の返り値が空の文字列 ($$$NULLOREF) の場合は、None を返します。詳細と例は、“Python からの関数とプロシージャの呼び出し” を参照してください。

```
function (functionName, routineName, args)
```

返り値 : bool、bytes、bytearray、int、float、Decimal、str、IRISList、Object、または None。特定のタイプを返す方法は、″function() 型キャスト・メソッド″ を参照してください。

パラメータ :

- `function_name` ― 呼び出す関数の名前。
- `routine_name` ― 関数を含むルーチンの名前。
- `*args` ― サポートされるタイプの 0 個以上のメソッド引数。タイプが bool、bytes、Decimal、float、int、str、および IRISList の引数は、リテラルとして投影されます。その他のタイプはすべて、プロキシ・オブジェクトとして投影されます。

function() に似ていて値を返さない ″procedure()″ も参照してください。

function() 型キャスト・メソッド

以下に示す IRIS.function() 型キャスト・メソッドのすべてが、IRIS.function() とまったく同じように機能するだけでなく、返り値を特定のタイプにキャストします。ObjectScript の返り値が空の文字列 ($$$NULLOREF) の場合、これらはすべて None を返します。詳細と例は、“Python からの関数とプロシージャの呼び出し” を参照してください。

```
functionBoolean (functionName, routineName, args)
functionBytes (functionName, routineName, args)
functionDecimal (functionName, routineName, args)
functionFloat (functionName, routineName, args)
functionInteger (functionName, routineName, args)
functionIRISList (functionName, routineName, args)
functionString (functionName, routineName, args)
functionObject (functionName, routineName, args)
```

返り値 : bool、bytes、bytearray、int、float、Decimal、str、IRISList、Object、または None

パラメータ :

- `function_name` ― 呼び出す関数の名前。
- `routine_name` ― 関数を含むルーチンの名前。
- `*args` ― サポートされるタイプの 0 個以上のメソッド引数。タイプが bool、bytes、Decimal、float、int、str、および IRISList の引数は、リテラルとして投影されます。その他のタイプはすべて、プロキシ・オブジェクトとして投影されます。

function() に似ていて値を返さない ″procedure()″ も参照してください。

get()

IRIS.get() は、プロパティの ObjectScript データ型に対応するタイプとしてグローバル・ノードの値を返します。ノードが空の文字列であるか、値がないか、またはノードが存在しない場合は、None を返します。

```
get (globalName, subscripts)
```

返り値 : bool、bytes、int、float、Decimal、str、IRISList、Object、または None。他のタイプを返す方法は、″get() 型キャスト・メソッド″ を参照してください。

パラメータ：

- global_name – 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

get() 型キャスト・メソッド

以下に示す IRIS.get() 型キャスト・メソッドのすべてが、IRIS.get() とまったく同じように機能するだけでなく、返り値を特定のタイプにキャストします。ノードが空の文字列であるか、値がないか、またはノードが存在しない場合は、これらすべてのメソッドが None を返します。

```
getBoolean (globalName, subscripts)
getBytes (globalName, subscripts)
getDecimal (globalName, subscripts)
getFloat (globalName, subscripts)
getInteger (globalName, subscripts)
getIRISList (globalName, subscripts)
getString (globalName, subscripts)
getObject (globalName, subscripts)
```

返り値：bool、bytes、int、float、Decimal、str、IRISList、Object、または None

パラメータ：

- global_name – 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

非推奨の型キャスト・メソッド "getLong()" も参照してください。

getAPIVersion() [static]

IRIS.getAPIVersion() は、このバージョンの Native SDK のバージョン文字列を返します。"getServerVersion()" も参照してください。

```
getAPIVersion ()
```

返り値：str

getLong() [非推奨]

IRIS.getLong()。Python long 型は、Python 3 で不要になりました。同じ精度をサポートする IRIS.get() 型キャスト・メソッド getInteger() を使用してください。

getServerVersion()

IRIS.getServerVersion() は、現在接続されている InterSystems IRIS サーバのバージョン文字列を返します。"getAPIVersion()" も参照してください。

```
getServerVersion ()
```

返り値：str

getTLevel()

IRIS.getTLevel() は、セッションで現在開いている入れ子になったトランザクションの数を返します（現在のトランザクションが入れ子になっていない場合は 1、開いているトランザクションがない場合は 0）。これは、ObjectScript $TLEVEL 特殊変数の値のフェッチと同じです。詳細と例は、“Python でのトランザクションの処理” を参照してください。

```
getTLevel ()
```

返り値：int

increment()

IRIS.increment() は、value 引数だけグローバル・ノードをインクリメントして、グローバル・ノードの新しい値を返します。指定されたアドレスに既存の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。このメソッドはきわめて高速の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。詳細と例は、“Python でのトランザクションの処理” を参照してください。

```
increment (value, globalName, subscripts)
```

返り値：float

パラメータ：

- value — インクリメントする int または float の数値
- global_name — 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

$INCREMENT は通常、新しいエントリをデータベースに追加する前に、カウンタをインクリメントするために使用します。$INCREMENT を使用すると、LOCK コマンドを使用しなくても、この操作を迅速に行うことができます。″ObjectScript リファレンス″ の “$INCREMENT とトランザクション処理” を参照してください。

isDefined()

IRIS.isDefined() は、ノードが値、子ノード、またはその両方を持つかどうかを示す値を返します。

```
isDefined (globalName, subscripts)
```

返り値：以下のいずれかの int 値

- 0：ノードが存在しない場合
- 1：ノードに値はあるが、子ノードがない場合
- 10：ノードに値はないが、1 つ以上の子ノードがある場合
- 11：ノードに値と子ノードの両方がある場合

パラメータ：

- global_name — 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

iterator() [非推奨]

IRIS.iterator() は非推奨です。代わりに node() を使用してください。既存のアプリケーションで継続使用される機能の詳細は、″クラス ″ を参照してください。

kill()

IRIS.kill() は、指定されたグローバル・ノードとそのすべてのサブノードを削除します。指定されたアドレスにノードがない場合、このコマンドは何も行いません。<UNDEFINED> 例外はスローされません。

```
kill (globalName, subscripts)
```

パラメータ：

- global_name – 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()

IRIS.lock() は、グローバルをロックします。このメソッドは増分ロックを実行します (前のロックをすべてアンロックする場合は、まず releaseAllLocks() メソッドを呼び出す必要があります)。ロックの取得を待機中に timeout 値に到達した場合は、<TIMEOUT> 例外がスローされます。詳細は、“Python による並列処理の制御” を参照してください。

```
lock (lock_mode, timeout, lock_reference, subscripts)
```

パラメータ :

- lock_mode – 次のいずれかの文字列 : 共有ロックの場合は "S"、エスカレート・ロックの場合は "E"、両方の場合は "SE"、どちらでもない場合は ""。空の文字列が既定モードです (非共有または非エスカレート)。
- timeout – ロックの取得を待機する秒数
- lock_reference – 曲折アクセント記号 (^) で始まり、その後にグローバル名が続く文字列 (例えば、単なる myGlobal ではなく ^myGlobal)。

  注意 : ほとんどのメソッドで使用される global_name パラメータとは異なり、lock_reference パラメータには先頭に曲折アクセント記号を付ける必要があります。global_name ではなく lock_reference を使用するのは、lock() および unlock() のみです。
- *subscripts – ターゲット・ノードを指定する 0 個以上の添え字

ロックの詳細は、″ObjectScript リファレンス″ の “LOCK” を参照してください。

nextSubscript()

IRIS.nextSubscript() は、ノード・アドレスを受け入れ、照合順で次の兄弟ノードの添え字を返します。指定された方向にノードがそれ以上見つからない場合は None が返されます。このメソッドは、ObjectScript の $ORDER と同様です。

```
nextSubscript (reversed, globalName, subscripts)
```

返り値 : bytes、int、str、または float (指定された方向の次の添え字)

パラメータ :

- reversed – ブーリアン値 true は、ノードを逆方向の照合順に走査する必要があることを示します。
- global_name – 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

″node()″ も参照してください。これは、指定されたノードの子を反復処理するオブジェクトを返します (指定されたノードと同じレベルのノードに対する反復処理を可能にする nextSubscript() とは異なります)。

node()

IRIS.node() は、指定されたノードの子に対する反復処理を可能にする IRISGlobalNode オブジェクトを返します。IRISGlobalNode は、グローバル・ノードの直接の子を表す仮想ディクショナリのように動作します。これは、反復処理、逆の操作、インデックス付け、およびスライスが可能です。

```
def  node (self, globalName, subscripts)
```

パラメータ：

- `global_name` — 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

"nextSubscript()"も参照してください。これを使用すると、指定されたノードと同じレベルのノードを反復処理できます (指定されたノードの子を反復処理する node() とは異なります)。

procedure()

IRIS.procedure() は ObjectScript プロシージャまたは関数を呼び出して、0 個以上の引数を渡し、何も返しません (IRIS.function() も参照)。詳細と例は、"Python からの関数とプロシージャの呼び出し" を参照してください。

```
procedure (procedureName, routineName, args)
```

パラメータ：

- `procedureName` — 呼び出すプロシージャの名前。
- `routine_name` — プロシージャを含むルーチンの名前。
- `*args` — 0 個以上のメソッド引数。タイプが bool、bytes、Decimal、float、int、str、および IRISList の引数は、リテラルとして投影されます。その他のタイプはすべて、プロキシ・オブジェクトとして投影されます。

このメソッドは、返り値がないことを想定していますが、任意の関数の呼び出しに使用できます。procedure() を使用して、値を返す関数を呼び出した場合、関数は実行されますが、返り値は無視されます。

releaseAllLocks()

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します。詳細は、"Python による並列処理の制御" を参照してください。

```
releaseAllLocks ()
```

set()

IRIS.set()は、value を現在のノード値として割り当てます。新しい値は、bool、bytes、bytearray、Decimal、float、int、str、または IRISList のいずれかです。

```
set (value, globalName, subscripts)
```

パラメータ：

- `value` — グローバル・ノードの新しい値
- `global_name` — 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

tCommit()

IRIS..tCommit() は、現在のトランザクションをコミットします。詳細と例は、“Python でのトランザクションの処理”を参照してください。

```
tCommit ()
```

tRollback()

IRIS..tRollback() は、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。詳細と例は、“Python でのトランザクションの処理”を参照してください。

```
tRollback ()
```

tRollbackOne()

IRIS..tRollbackOne() は、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。これは入れ子になったトランザクションを対象としており、呼び出し元が 1 レベルのみをロールバックする場合に使用します。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。詳細と例は、“Python でのトランザクションの処理”を参照してください。

```
tRollbackOne ()
```

tStart()

IRIS.tStart() は、トランザクションを開始するか、開きます。詳細と例は、“Python でのトランザクションの処理”を参照してください。

```
tStart ()
```

unlock()

IRIS.unlock() は、指定されたロックのロック・カウントをデクリメントし、ロック・カウントが 0 の場合はアンロックします。共有ロックまたはエスカレート・ロックを削除するには、適切な lockMode を指定する必要があります (共有ロックの場合は "S"、エスカレート・ロックの場合は "E")。 詳細は、“Python による並列処理の制御”を参照してください。

```
unlock (lock_mode, lock_reference, subscripts)
```

パラメータ：

- `lock_mode` ― 次のいずれかの文字列：共有ロックの場合は "S"、エスカレート・ロックの場合は "E"、両方の場合は "SE"、どちらでもない場合は ""。空の文字列が既定モードです (非共有または非エスカレート)。
- `lock_reference` ― 曲折アクセント記号 (^) で始まり、その後にグローバル名が続く文字列 (例えば、単なる `myGlobal` ではなく `^myGlobal`)。

  注意：ほとんどのメソッドで使用される `global_name` パラメータとは異なり、`lock_reference` パラメータには先頭に曲折アクセント記号を付ける必要があります。`global_name` ではなく `lock_reference` を使用するのは、lock() および unlock() のみです。
- `*subscripts` ― ターゲット・ノードを指定する 0 個以上の添え字

7.3 クラス iris.IRISList

クラス iris.IRISList 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 Python インタフェースを実装します。IRISList は、Native SDK でサポートされるタイプです（“型キャスト・メソッドおよびサポートされるデータ型”を参照）。

7.3.1 IRISList コンストラクタ

IRISList コンストラクタは、以下のパラメータを取ります。

```
IRISList(buffer = None, locale = "latin-1", is_unicode = True, compact_double = False)
```

パラメータ：

- `buffer` — オプションのリスト・バッファ。IRISList のインスタンスまたは $LIST 形式のバイト配列（IRIS.getBytes() などの Native SDK メソッドから返されます）のいずれかです。
- `locale` — バッファのロケール設定を示すオプションの str。
- `is_unicode` — バッファが Unicode かどうかを示すオプションの bool。
- `compact_double` — Compact Double が有効になっているかどうかを示すオプションの bool。

IRISList の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

```
list = IRISList()
```

- 別の IRISList のコピーを作成する

```
listcopy = IRISList(myOtherIRISList)
```

- IRIS.getBytes() やその他さまざまな iris メソッド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構築する

```
globalBytes = myIris.getBytes("myGlobal",1)
listFromByte = IRISList(globalBytes)
```

iris パッケージ内の多くのメソッドは、Native SDK でサポートされるタイプの 1 つである IRISList を返します。

7.3.2 IRISList メソッドの詳細

add()

IRISList.add() は、IRISList の末尾に値を追加して、その IRISList オブジェクトを返します。

```
add (value)
```

返り値：self（IRISList オブジェクト）

パラメータ：

- `value` — いずれかのサポート対象タイプの値、または値の配列かコレクション。配列またはコレクションの各要素は、個別に追加されます。IRISList のインスタンスは、常に単一要素として追加されます。

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。ただし、IRISList.toArray() を使用して、IRISList を配列に変換できます。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()

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て、その IRISList オブジェクトを返します。

```
clear ()
```

返り値 : self (IRISList オブジェクト)

count()

IRISList.count() は、リスト内のデータ要素の数を返します。

```
count ()
```

返り値 : int

equals()

IRISList.equals() は、指定された irislist2 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。

```
equals (irislist2)
```

返り値 : bool

パラメータ :

- `irislist2` — 比較する IRISList のインスタンス。

get()

IRISList.get() は、リストのカーソルを指定された index に移動し (指定されている場合)、カーソル位置にある要素を Object として返します。インデックスが範囲外の場合 (1 未満であるか、リストの末尾を越えている)、IndexOutOfBoundsException をスローします。以下の場合、None を返します。

```
get (index)
```

返り値 : bool、bytes、bytearray、int、float、Decimal、str、IRISList、Object、または None

パラメータ :

- `index` — 新しいカーソル位置のインデックス (1 が基点) を指定するオプションの int

get() 型キャスト・メソッド

以下に示す IRISList.get() 型キャスト・メソッドのすべてが、IRISList.get() とまったく同じように機能するだけでなく、返り値を特定のタイプにキャストします。インデックスが範囲外の場合 (1 未満であるか、リストの末尾を越えている)、これらはすべて IndexOutOfBoundsException をスローします。

```
getBoolean (index)
getBytes (index)
getDecimal (index)
getFloat (index)
getInteger (index)
getIRISList (index)
getString (index)
```

返り値 : bool、bytes、bytearray、int、float、Decimal、str、IRISList、Object、または None

パラメータ :

- index — 新しいカーソル位置のインデックス (1 が基点) を指定するオプションの int

remove()

IRISList.remove() は、リストの index にある要素を削除して、その IRISList オブジェクトを返します。

```
remove (index)
```

返り値 : self (IRISList オブジェクト)

パラメータ :

- index — 削除する要素のインデックス (1 が基点) を指定する int

set()

IRISList.set() は、index にあるリスト要素を value に置き換えて、その IRISList オブジェクトを返します。

value が配列である場合、各配列要素が index の位置から開始して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。index が 1 未満の場合は、IndexOutOfBoundsException をスローします。

```
set (index, value)
```

返り値 : self (IRISList オブジェクト)

パラメータ :

- index — 設定するリスト要素のインデックス (1 が基点) を指定する int。
- value — index に挿入する Object 値または Object 配列。オブジェクトはサポートされているどのタイプでもかまいません。

size()

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。

```
size ()
```

返り値 : int

7.4 クラス iris.IRISConnection

IRISConnection のインスタンスは、パッケージのメソッド iris.connect() を呼び出すことによって作成されます。接続の作成および使用の詳細は、“iris パッケージのメソッド” を参照してください。

close()

IRISConnection.close() は、iris インスタンスの接続を閉じます (開かれている場合)。接続が既に閉じられている場合は、何もしません。

```
connection.close()
```

返り値 : self

isClosed()

IRISConnection.isClosed() は、接続が成功すると True を返し、それ以外の場合は False を返します。

```
connection.isClosed()
```

返り値 : bool

isUsingSharedMemory()

IRISConnection.isUsingSharedMemory() は、接続が開かれていて共有メモリが使用されている場合、True を返します。

```
connection.isUsingSharedMemory()
```

返り値 : bool

プロパティ

最後に成功した接続試行の hostname、port、namespace、timeout、および logfile が接続オブジェクトのプロパティとして保存されます。

7.5 クラス iris.IRISObject

クラス iris.IRISObject は、外部サーバの逆プロキシ・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、“Python からのデータベース・オブジェクトの制御” を参照)。

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。

7.5.1 IRISObject コンストラクタ

IRISObject コンストラクタは、以下のパラメータを取ります。

```
IRISObject(connection, oref)
```

パラメータ :

- connection — IRISConnection オブジェクト
- oref — 制御するデータベース・オブジェクトの OREF

7.5.2 IRISObject メソッドの詳細

close()

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

```
close ()
```

返り値：self

get()

IRISObject.get() は、プロキシ・オブジェクトのプロパティ値をフェッチします。IRIS の空の文字列（$$$NULLOREF）の場合に None を返し、それ以外の場合は、プロパティの ObjectScript データ型に対応するタイプの変数を返します。

```
get (propertyName)
```

返り値：bool、bytes、bytearray、int、float、Decimal、str、IRISList、Object、または None

パラメータ：

- property_name — 返されるプロパティの名前。

get() 型キャスト・メソッド

以下に示す IRISObject.get() 型キャスト・メソッドのすべてが、IRISObject.get() とまったく同じように機能するだけでなく、返り値を特定のタイプにキャストします。これらはすべて、IRIS の空の文字列（$$$NULLOREF）の場合に None を返し、それ以外の場合は、指定されたタイプの値を返します。

```
getBoolean (propertyName)
getBytes (propertyName)
getDecimal (propertyName)
getFloat (propertyName)
getInteger (propertyName)
getIRISList (propertyName)
getString (propertyName)
getObject (propertyName)
```

返り値：bool、bytes、bytearray、int、float、Decimal、str、IRISList、Object、または None

パラメータ：

- property_name — 返されるプロパティの名前。

getConnection()

IRISObject.getConnection() は、現在の接続を IRISConnection オブジェクトとして返します。

```
getConnection ()
```

返り値：IRISConnection オブジェクト

getOREF()

IRISObject.getOREF() は、この IRISObject にマップされているデータベース・オブジェクトの OREF を返します。

```
getOREF ()
```

返り値：OREF

invoke()

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、プロパティの ObjectScript データ型に対応するタイプの変数を返します。

```
invoke (methodName, args)
```

返り値：bool、bytes、bytearray、int、float、Decimal、str、IRISList、Object、または None

パラメータ：

- `method_name` — 呼び出されるインスタンス・メソッドの名前。
- `*args` — サポートされるタイプの 0 個以上の引数。

invoke() に似ていて値を返さない "invokeVoid()" も参照してください。

invoke() 型キャスト・メソッド

以下に示す IRISObject.invoke() 型キャスト・メソッドのすべてが、IRISObject.invoke() とまったく同じように機能するだけでなく、返り値を特定のタイプにキャストします。これらはすべて、IRIS の空の文字列（$$$NULLOREF）の場合に None を返し、それ以外の場合は、指定されたタイプの値を返します。

```
invokeBoolean (methodName, args)
invokeBytes (methodName, args)
invokeDecimal (methodName, args)
invokeFloat (methodName, args)
invokeInteger (methodName, args)
invokeIRISList (methodName, args)
invokeString (methodName, args)
invokeObject (methodName, args)
```

返り値：bool、bytes、bytearray、int、float、Decimal、str、IRISList、Object、または None

パラメータ：

- `method_name` — 呼び出されるインスタンス・メソッドの名前。
- `*args` — サポートされるタイプの 0 個以上の引数。

invoke() に似ていて値を返さない "invokeVoid()" も参照してください。

invokeVoid()

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。

```
invokeVoid (methodName, args)
```

パラメータ：

- `method_name` — 呼び出されるインスタンス・メソッドの名前。
- `*args` — サポートされるタイプの 0 個以上の引数。

set()

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。

```
set (propertyName, propertyValue)
```

パラメータ：

- property_name – 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。値はサポートされているどのタイプでもかまいません。

7.6 クラス iris.IRISReference

iris.IRISReference クラスは、データベースのクラス・メソッドを呼び出す際に、参照渡し引数を使用できるようにします。詳細と例は、“引数の参照渡し”を参照してください。

7.6.1 IRISReference コンストラクタ

IRISReference コンストラクタは、以下のパラメータを取ります。

```
IRISReference(value, type = None)
```

パラメータ：

- value – 参照される引数の初期値。
- type – 引数の変更された値のマーシャリング解除にタイプ・ヒントとして使用されるオプションの Python タイプ。サポートされるタイプは、bool、bytes、bytearray、Decimal、float、int、str、IRISList、または None です。None が指定されている場合、元のデータベース・タイプに一致するタイプが使用されます。

7.6.2 IRISReference メソッドの詳細

get_type() [非推奨]

IRISReference.get_type() は非推奨です。目的のタイプの返り値を得るには、getValue() 型キャスト・メソッドのいずれかを使用してください。

```
get_type ()
```

get_value() [非推奨]

IRISReference.get_value() は非推奨です。代わりに、getValue() または getObject()を使用してください。

getValue()

IRISReference.getValue() は、参照されたパラメータの値を、データベース値のデータ型に対応するタイプとして返します。

```
getValue ()
```

返り値：参照されたパラメータの値

getValue() 型キャスト・メソッド

以下に示す IRISReference.getValue() 型キャスト・メソッドのすべてが、IRISReference.getValue() とまったく同じように機能するだけでなく、返り値を特定のタイプにキャストします。

```
getBoolean ()
getBytes ()
getDecimal ()
getFloat ()
getInteger ()
getIRISList ()
getString ()
getObject ()
```

返り値 : bool、bytes、bytearray、int、float、Decimal、str、IRISList、Object、または None

set_type() [非推奨]

IRISReference.set_type() は非推奨です。IRISReferenceを呼び出す際は、タイプを設定するオプションがあります。

```
set_type (type)
```

setValue()

IRISReference.setValue() は、参照される引数の値を設定します。

```
setValue (value)
```

パラメータ :

- value — この IRISReference オブジェクトの値

set_value() [非推奨]

IRISReference.set_value() は非推奨です。代わりに setValue() を使用してください。

7.7 クラス iris.IRISGlobalNode

IRISGlobalNode は、グローバル・ノードの直接の子を表す仮想ディクショナリのように動作する反復処理可能なインタフェースを提供します。これは、反復処理、スライス、逆の操作、インデックス付けが可能で、ビューおよびメンバシップ・テストをサポートします。

- IRISGlobalNode では、反復処理可能なインタフェースがサポートされています。

  例えば、以下の for ループは、すべての子ノードの添え字を一覧表示します。

```
for x in node:
    print(x)
```

- IRISGlobalNode では、ビューがサポートされています。

  メソッド keys()、subscripts()、values()、items()、および nodes() は、IRISGlobalNodeView ビュー・オブジェクトを返します。これらは、それぞれのデータを生成するために反復処理可能な、IRISGlobalNode エントリに関する特定のビューを提供します。また、メンバシップ・テストをサポートします。例えば、items() メソッドは、添え字と値のペアのリストが含まれるビュー・オブジェクトを返します。

```
for sub, val in node.items():
    print('subscript:',sub,' value:', val)
```

- IRISGlobalNode は、スライス可能です。

  標準の Python スライス構文を使用して、IRISGlobalNode を添え字のより制限された範囲に対して反復処理できます。

  ```
  node[start:stop:step]
  ```

  これにより、start (含まれる) から stop (含まれない) までに限定された添え字範囲を持つ新しい IRISGlobalNode オブジェクトが作成されます。step は、1 または -1 で、前方向または逆方向の走査を意味します。

- IRISGlobalNode は、逆向きの処理が可能です。

  スライス構文で "step" 変数が -1 の場合、IRISGlobalNode は、標準の順序とは逆の後ろ方向に反復処理します。例えば、以下の文では、8 (含まれる) から 2 (含まれない) までの添え字を走査します。

  ```
  for x in node[8:2:-1]: print(x)
  ```

- IRISGlobalNode は、インデックス付け可能で、メンバシップ・テストをサポートします。

  例えば、添え字 x を持つ子ノードの場合、ノード値を `node[x] = y` と設定して、`z = node[x]` で返すことができます。`x in node` のメンバシップ・テストは、ブーリアン値を返します。

7.7.1 IRISGlobalNode コンストラクタ

IRISGlobalNode のインスタンスは、IRIS.node()、IRISGlobalNode.node()、または IRISGlobalNode.nodes() を呼び出すことで作成できます。

7.7.2 IRISGlobalNode メソッドの詳細

get()

iris.IRISGlobalNodeget() は、指定された subscript にあるノードの値を返します。ノードに値がない、またはノードが存在しない場合は、default_value を返します。

```
get (subscript, default_value)
```

パラメータ：

- `subscript` — 取得する値が含まれる、子ノードの添え字
- `default_value` — 指定された添え字に値がない場合に返す値

items()

iris.IRISGlobalNode.items() は、このノードのすべての子ノードについて添え字と値のタプルを示したビューを返します。

```
items ()
```

返り値：添え字と値のタプルを示した IRISGlobalNodeView オブジェクト

keys()

iris.IRISGlobalNode.keys() は、このノードのすべての子ノードの添え字を示したビューを返します。subscripts() とまったく同じです。

```
keys ()
```

返り値：添え字のみを示した IRISGlobalNodeView オブジェクト

node()

iris.IRISGlobalNode.node() は、指定された添え字にある子ノードを表す IRISGlobalNode オブジェクトを返します。

```
node (subscript)
```

返り値：IRISGlobalNode オブジェクト

パラメータ：

- `subscript` — 目的の子ノードの添え字

nodes()

iris.IRISGlobalNode.nodes() は、このノードの子ノードごとの IRISGlobalNode オブジェクトを示したビューを返します。

```
nodes ()
```

返り値：IRISGlobalNode オブジェクトを示す IRISGlobalNodeView オブジェクト

subscripts()

iris.IRISGlobalNode.subscripts() は、このノードのすべての子ノードの添え字を示したビューを返します。keys() とまったく同じです。

```
subscripts ()
```

返り値：添え字のみを示した IRISGlobalNodeView オブジェクト

values()

iris.IRISGlobalNode.values() は、このノードのすべての子ノードの値を示したビューを返します。ノードに値がない場合は None を返します。

```
values ()
```

返り値：値のみを示した IRISGlobalNodeView オブジェクト

7.8 クラス iris.IRISGlobalNodeView

クラス iris.IRISGlobalNodeView は、子ノードからデータを生成するために反復処理可能な、IRISGlobalNode のビュー・オブジェクトを実装します。IRISGlobalNodeView オブジェクトは、IRISGlobalNode メソッド keys()、subscripts()、values()、items()、および nodes() によって返されます。

7.9 従来のサポート対象クラス

これらのクラスは、従来のアプリケーションが変更なしに動作できるように維持されています。これらのクラスを新しいコードで使用することはできません。従来のコードは変更なしで機能します。

7.9.1 クラス iris.LegacyIterator [非推奨]

このクラスは、下位互換性のためだけに提供されています。クラス iris.LegacyIterator を使用すると、非推奨の irisnative.Iterator クラスを使用して記述されているコードを変更なしで引き続き実行できます。

新しいコードでは、代わりに iris.IRISGlobalNode クラスを使用する必要があります。

7.9.1.1 LegacyIterator メソッドの詳細

next() [非推奨]

iterator.next() は、次の子ノードに反復子を配置して、現在有効になっている返りタイプに応じて、ノード値、ノードの添え字、またはその両方を含むタプルを返します (以下を参照)。反復処理にこれ以上ノードがない場合は、StopIteration 例外をスローします。

```
iterator.next()
```

返り値 : 以下のいずれかの返りタイプのノード情報。

- `subscript and value (default)` — 反復処理の次のノードの添え字および値を含むタプル。添え字はタプルの最初の要素で、値は 2 番目の要素です。現在、反復子が別の返りタイプに設定されている場合は、items() を呼び出すことによって、このタイプを有効にします。
- `subscript only` — subscripts() を呼び出すことによって、この返りタイプを有効にします。
- `value only` — values() を呼び出すことによって、この返りタイプを有効にします。

startFrom() [非推奨]

iterator.startFrom() は、開始位置が指定の添え字に設定された反復子を返します。next() を呼び出して、指定した位置の後にある次の子ノードに反復子を進めるまで、反復子はノードを指しません。

```
iterator.startFrom(subscript)
```

返り値 : iterator の呼び出し側のインスタンス

パラメータ :

- `subscript` — 開始位置を示す単一の添え字。

None を引数としてこのメソッドを呼び出すことは、既定の開始位置を使用することと同じです。既定の開始位置は、反復処理の方向に応じて、最初のノードの直前または最後のノードの直後です。

reversed() [非推奨]

iterator.reversed() は、反復処理の方向を前の設定から逆にした反復子を返します (反復子の作成時に、方向は順方向の反復処理に設定されます)。

```
iterator.reversed()
```

返り値 : iterator の同じインスタンス

subscripts() [非推奨]

iterator.subscripts() は、返りタイプが添え字のみに設定された反復子を返します。

```
iterator.subscripts()
```

返り値 : iterator の同じインスタンス

values() [非推奨]

iterator.values() は、返りタイプが値のみに設定された反復子を返します。

```
iterator.values()
```

返り値 : iterator の同じインスタンス

items() [非推奨]

iterator.items() は、返りタイプが、添え字とノード値の両方を含むタプルに設定された反復子を返します。添え字はタプルの最初の要素で、値は 2 番目の要素です。これは、反復子の作成時の既定の設定です。

```
iterator.items()
```

返り値 : iterator の同じインスタンス

7.9.1.2 従来の反復処理の例

以下の例では、iterDogs を設定して、グローバル配列 heroes('dogs') の子ノードを反復処理します。subscripts() メソッドは反復子の作成時に呼び出されるため、next() を呼び出すたびに、現在の子ノードの添え字のみが返されます。添え字はそれぞれ output 変数に追加され、ループが終了すると、リスト全体が出力されます。シーケンスにこれ以上子ノードがない場合、StopIteration 例外がスローされます。

next() を使用したノード heroes('dogs') の下の添え字の一覧表示

```
# Get a list of child subscripts under node heroes('dogs')
iterDogs = irispy.iterator('heroes','dogs').subscripts()
output = "\nSubscripts under node heroes('dogs'): "
try:
  while True: output += '%s ' % iterDogs.next()
except StopIteration:  # thrown when there are no more child nodes
  print(output + '\n')
```

このコードによって、以下のような出力が生成されます。

```
Subscripts under node heroes('dogs'): Balto Hachiko Lassie Whitefang
```

7.9.2 クラス irisnative.IRISNative [非推奨]

このクラスは、下位互換性のためだけに提供されています。クラス irisnative.IRISNative を使用すると、非推奨の irisnative 接続メソッドを使用して記述されているコードを変更なしで引き続き実行できます。

以前の createConnection() メソッドおよび createIRIS() メソッドは、iris パッケージのメソッド connect() および createIRIS() (“iris” を参照) として公開されるようになっており、新しいコードでは常にこれらを使用する必要があります。

createConnection() [非推奨]

このメソッドは、iris パッケージのメソッド iris.connect() に置き換えてください。

IRISNative.createConnection() は、IRIS インスタンスへの新しい接続の作成を試み、新しい接続オブジェクトを返します。このオブジェクトは、接続が成功すると開かれ、それ以外の場合は閉じられます。

```
irisnative.createConnection(hostname,port,namespace,username,password,timeout,sharedmemory,logfile)
   irisnative.createConnection(connectionstr,username,password,timeout,sharedmemory,logfile)
```

最後に成功した接続試行の hostname、port、namespace、timeout、および logfile が接続オブジェクトのプロパティとして保存されます。

返り値：connection の新しいインスタンス

パラメータ：パラメータは位置またはキーワードによって渡すことができます。

- hostname ― サーバ URL を指定する str
- port ― スーパーサーバ・ポート番号を指定する int
- namespace ― ネームスペース・サーバを指定する str
- 以下のパラメータを、hostname、port、および namespace の引数の代わりに使用できます。
  - connectionstr ― *hostname*:*port*/*namespace* の形式の str
- username ― ユーザ名を指定する str
- password ― パスワードを指定する str
- timeout (オプション) ― 接続試行時に待機する最大ミリ秒数を指定する int既定値は 10000 です。
- sharedmemory (オプション) ― bool を True に指定すると、ホスト名が localhost または 127.0.0.1 の場合に共有メモリ接続を試行します。False に指定すると、TCP/IP 経由の接続を強制します。既定値は True です。
- logfile (オプション) ― クライアント側のログ・ファイル・パスを指定する strパスの最大長は、255 文字の ASCII 文字です。

createIris() [非推奨]

このメソッドは、iris パッケージのメソッド iris.createIRIS() に置き換えてください。

IRISNative.createIris() は、指定された接続を使用する IRIS の新規インスタンスを返します。接続が閉じられている場合は、例外をスローします。

```
irisnative.createIris(conn)
```

返り値：IRIS の新しいインスタンス

パラメータ：

- conn ― サーバ接続を提供するオブジェクト