グローバルの使用法

Version 2023.1
2024-01-02

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

グローバルの使用法

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

テーブル一覧

1

グローバルの概要

多次元ストレージ・エンジンは、InterSystems IRIS® の主な機能の 1 つです。この機能により、アプリケーションはデータをコンパクトで効率的な多次元スパース配列に格納できます。このような配列はグローバルと呼ばれます。

グローバルとは、任意の型のデータを格納できる、添え字付き変数です。このデータはディスク上に保存され、"グローバル" という名前が示すように、複数のプロセス間でグローバルに利用できます。グローバルは、その名前の前に記述されたキャレット (^) で通常の変数と区別できます。

一般的にグローバルは多次元スパース配列とされます。以下に示すように、配列状の構文を持ちながらも、配列にあるすべての添え字にデータが存在するとは限らないからです。

```
^a(1,2,3) = 4
^a(1,2,5000) = 4
```

ただし、多くの言語における配列とは異なり、グローバルの添え字には、以下に示すように負数、実数、または文字列を使用できます。

```
^b(-4, "hello", 3.14) = 4
```

このため、多くの場合、グローバルの概念は、キーと値のペアで構成するディクショナリ、または入れ子構造のディクショナリとされています。以下の例では、学名に従って鳥の名前を格納しています。

```
^bird("Anatidae", "Aix", "sponsa") = "Wood Duck"
^bird("Anatidae", "Anas", "rubripes") = "American Black Duck"
^bird("Anatidae", "Branta", "leucopsis") = "Barnacle Goose"
^bird("Odontophoridae", "Callipepia", "californica") = "California Quail"
^bird("Odontophoridae", "Callipepia", "gambelii") = "Gambel's Quail"
```

多くの言語では、ディクショナリのデータに決まった順序がありません。つまり、ディクショナリのデータを取得すると、そのデータは不定の順序で返されることが考えられます。しかし、グローバルの場合、データは格納されたときの添え字に従ってソートされています。

そのため、グローバルはツリー構造で表現する方が正確です。ツリー構造では、各ノードに値やその子を置くことができます。この点で、普通はツリーのリーフのみにデータが存在する、入れ子になったディクショナリ・モデルよりも柔軟です。以下の例では、ルート・ノードにグローバルの説明を格納し、鳥類を表すノードにその分類の説明を格納しています。

```
^bird="Birds of North America"
^bird("Anatidae") = "Ducks, Geese and Swans"
^bird("Anatidae", "Aix", "sponsa") = "Wood Duck"
^bird("Anatidae", "Anas", "rubripes") = "American Black Duck"
^bird("Anatidae", "Branta", "leucopsis") = "Barnacle Goose"
^bird("Odontophoridae") = "New World Quails"
^bird("Odontophoridae", "Callipepia", "californica") = "California Quail"
^bird("Odontophoridae", "Callipepia", "gambelii") = "Gambel's Quail"
```

ツリー構造にデータを格納する様子を説明した動画は、"ツリー構造" を参照してください。

1.1 特徴

グローバルは、永続多次元配列での簡単なデータ格納法を提供します。

例えば、^Settings というグローバルを使用して、値“Red”を キー“Color”と関連付けることができます。

ObjectScript

```
SET ^Settings("Color")="Red"
```

グローバルの多次元性を利用して、さらに複雑な構造の定義が可能となります。

ObjectScript

```
SET ^Settings("Auto1","Properties","Color") = "Red"
SET ^Settings("Auto1","Properties","Model") = "SUV"
SET ^Settings("Auto2","Owner") = "Mo"
SET ^Settings("Auto2","Properties","Color") = "Green"
```

グローバルには以下の機能があります。

- 使用の容易性 ー グローバルは、他のプログラミング言語の変数と同様に使用が簡単です。
- 多次元 ー 任意の添え字を使用して、グローバルでノードのアドレスを指定できます。例えば `^Settings("AUTO2","Properties","Color")` の場合、添え字 Color は Settings グローバルで 3 番目のノードになります。添え字は、整数、数値、あるいは文字列値で、連続している必要はありません。
- スパース ー グローバル・ノードのアドレス指定をするための添え字は非常にコンパクトで、連続した値を持つ必要はありません。
- 効率的 ー グローバルでの処理 (挿入、更新、削除、走査、検索) はすべて、最高のパフォーマンスと並行処理のために高度に最適化されています。他にも、特定の操作に対応するためのコマンドがあります (データの一括挿入など)。また、記録のソートなど一時的なデータ構造のための特別仕様グローバルもあります。
- 信頼性 ー InterSystems IRIS データベースは、論理レベルのジャーナリング、物理レベルのジャーナリングなど、グローバル内に格納されたデータの信頼性を確保するさまざまなメカニズムを備えています。グローバル内に格納されるデータは、データベースのバックアップ操作が実行されるときにバックアップされます。
- 分散型 ー InterSystems IRIS では、グローバル内に格納されたデータの物理位置をさまざまな方法で制御することができます。グローバルの格納に使用する物理データベースを定義し、複数のデータベースにグローバルの一部を配布できます。InterSystems IRIS の分散型データベース機能を使用することで、データベースおよびアプリケーション・サーバ・システムのネットワーク全体でグローバルを共有することができます。また、ミラーリング・テクノロジにより、システムのグローバル内に格納されたデータを、他のシステムに自動的に複製することができます。
- 並行処理 ー グローバルは、複数プロセス間の同時アクセスをサポートします。個別ノード (配列要素) 内の値の設定と取得は常にアトミックです。信頼性のある同時アクセスを保証するためのロックは必要ありません。また、InterSystems IRIS は強力なロック操作をサポートしており、複数のノードなど、より複雑な状況で並行処理を行うために使用できます。オブジェクトや SQL アクセス使用の際、この並行処理は自動的に実行されます。
- トランザクション ー InterSystems IRIS はトランザクションの境界を指定するコマンドを提供しているので、これを使用するとトランザクションの開始、実行、ロールバックなどが可能です。ロールバックの際、トランザクション内でのグローバルの変更をすべて取り消し、データベースのコンテンツをトランザクション以前の状態にリストアします。InterSystems IRIS のさまざまなロック操作をトランザクションと併用することで、グローバルを使用して従来の ACID トランザクションを実行できます(ACID トランザクションは、アトミック性、一貫性、分離性、持続性を提供します)。オブジェクトや SQL アクセス使用の際、トランザクションは自動的に処理されます。

注釈　このドキュメントで説明するグローバルを、別のタイプの InterSystems IRIS 変数配列であるプロセス・プライベート・グローバルと混同しないようにしてください。プロセス・プライベート・グローバルは永続的ではありません。これを作成したプロセスの間のみ維持されます。プロセス・プライベート・グローバルは並行でもありません。これを作成したプロセスからのみアクセスできます。複数文字の名前の接頭語、^|| または ^|"^"| により、プロセス・プライベート・グローバルはグローバルと簡単に区別できます。

1.2 例

グローバルの容易性とパフォーマンスは、簡単な例を取り上げるとわかります。以下のプログラムは、10,000 個のノードの配列を生成(既存ノードは先に削除)し、データベースに保存します。グローバルのパフォーマンスを実感するために試してみてください。

永続配列の生成

```
Set start = $ZH  // get current time

Kill ^Test.Global
For i = 1:1:10000 {
    Set ^Test.Global(i) = i
}

Set elap = $ZH - start  // get elapsed time
Write "Time (seconds): ",elap
```

配列内の値を繰り返し処理して読み取るまでに、どのくらいの時間がかかるのか見てください (まず上記例を実行して、配列を構築してください)。

永続配列の読み取り

```
Set start = $ZH  // get current time
Set total = 0
Set count = 0

// get key and value for first node
Set i = $Order(^Test.Global(""),1,data)

While (i '= "") {
    Set count = count + 1
    Set total = total + data

   // get key and value for next node
   Set i = $Order(^Test.Global(i),1,data)
}

Set elap = $ZH - start  // get elapsed time

Write "Nodes:          ",count,!
Write "Total:          ",total,!
Write "Time (seconds): ",elap,!
```

1.3 アプリケーションの使用法

InterSystems IRIS アプリケーションでは、さまざまな方法でグローバルを使用します。以下はその例です。

- オブジェクト、および SQL エンジンに共有される基本ストレージ・メカニズムとしての使用。
- オブジェクトおよび SQL データに対するビットマップ・インデックスなど、多様なインデックスを提供するメカニズムとしての使用。
- プロセス・メモリに収まらない可能性のある操作を実行するためのワーク・スペースとしての使用。例えば、用途に合った既存のインデックスがない場合、SQL エンジンはデータをソートするために一時的なグローバルを使用します。

- オブジェクトおよび SQL アクセスという点で、表現が困難もしくは非効率的な永続オブジェクトまたは SQL テーブルでの特別なオペレーションを実行するため。例えば、メソッド（あるいはストアド・プロシージャや Web メソッド）を指定して、テーブルの保持データについて特別なデータ分析を行うことが可能です。メソッドを使用することで、そのような操作は完全にカプセル化され、呼び出し側は単にメソッドを呼び出すだけで済みます。
- アプリケーションに特化したカスタマイズされたストレージ構造を実装するため。リレーショナル形式で表現することが難しいデータの保存を必要とするアプリケーションが数多く存在します。グローバルでカスタム構造を指定し、外部クライアントがその構造をオブジェクト・メソッド経由で利用できるようにします。
- 構成データ、クラス定義、エラー・メッセージ、実行可能なコードなど、InterSystems IRIS システムで使用される特別な目的を持った多様なデータ構造のため。

グローバルは、リレーショナル・モデルの範囲に限定されません。特定のアプリケーションを最大限に利用するカスタマイズされた構造を自由に開発できます。多くのアプリケーションでグローバルを賢明に使用することにより、リレーショナル・アプリケーション開発者達が待ち望んだパフォーマンスを提供することができます。

アプリケーションでグローバルを直接使用するかどうかにかかわらず、その動作を理解しておくと役に立ちます。グローバルとその性能を理解することは、アプリケーションの最適な配置構成を決定するうえで役立つのはもとより、さらに効率的なアプリケーションを設計する際にも役立ちます。

2
グローバル構造

この章では、グローバルの論理ビューについて説明するとともに、物理的にグローバルがどのようにディスクに格納されるかについての概要を説明します。

2.1 グローバルの論理構造

グローバルとは名前付きの多次元配列で、物理 InterSystems IRIS® データベースに格納されます。アプリケーションでは、現在のネーム・スペースに基づいて、グローバルが物理データベースにマッピングされます。ネーム・スペースは、1 つ以上の物理データベースの論理的な統一ビューを提供します。

2.1.1 グローバルの名前付け規約と制限

グローバルの名前は、その目的と使用法を明示します。2 つのタイプのグローバルと、“プロセス・プライベート・グローバル” と呼ばれる個別の変数のセットがあります。

- グローバル – これは、標準グローバルと呼ばれることがあります。通常、これらは、単にグローバルと呼ばれます。現在のネームスペースに存在する永続的な多次元の配列です。
- ″拡張グローバル参照″ – これは、現在のネームスペース以外のネームスペースにあるグローバルです。
- ″プロセス・プライベート・グローバル″ – これは、これを作成したプロセスによってのみアクセス可能な配列変数です。

グローバルの命名規約は、以下のとおりです。

- グローバル名はキャレット文字 (^) で開始します。このキャレット文字は、ローカル変数とグローバルを区別します。
- グローバル名の先頭にあるキャレット (^) 文字の直後には、以下を使用できます。
  - 文字またはパーセント記号 (%) – 標準グローバル用のみです。グローバル名の場合、文字は ASCII 65 から ASCII 255 の範囲のアルファベット文字として定義されます。“%” で始まる名前を持つグローバル (ただし、“%Z” または “%z” で始まるものを除く) は、InterSystems IRIS システム用です。% グローバルは、通常 IRISSYS データベースまたは IRISLIB データベースに格納されます。% 文字およびインターシステムズの命名の詳細は、″サーバ側プログラミングの入門ガイド″ の “識別子のルールとガイドライン” を参照してください。
  - 垂直バー (|) または左角括弧 ([) – 拡張グローバル参照またはプロセス・プライベート・グローバル用。使用法は、後続の文字によって異なります。このリストに続く例を参照してください。

- 上記以外のグローバル名には、文字、数字、またはピリオド記号(.)を使用できます。パーセント記号(%)は、グローバル名の最初の文字としてのみ使用できます。またピリオド記号(.)は、グローバル名の最後の文字としては使用できません。
- グローバル名の長さは、先頭のキャレット文字を除いて最大 31 文字とします。それ以上に長い名前を付けることも可能ですが、InterSystems IRIS でグローバル名として処理されるのは 31 文字までです。
- グローバル名は、大文字と小文字を区別します。
- InterSystems IRIS では、グローバル参照の合計の長さに制限を課しています。同様に、この制限によってあらゆる添え字の値の長さにも制限が課されます。詳細は、“グローバル参照の最大長”を参照してください。

IRISSYS データベースでは、グローバル名はすべて予約されていますが、先頭に“z”、“Z”、“%z”、および“%Z”が付くものについては例外です。他のすべてのデータベースでは、先頭に“ISC.”および“%ISC.”が付くグローバル名はすべて予約されています。

2.1.1.1 サンプル・グローバル名とその使用法

以下に、各種グローバル名とそれぞれの使用法の例を示します。

- `^globalname` ― 標準グローバル
- `^|"environment"|globalname` ― 拡張グローバル参照の環境構文
- `^||globalname` ― プロセス・プライベート・グローバル
- `^|"^"|globalname` ― プロセス・プライベート・グローバル
- `^[namespace]globalname` ― 拡張グローバル参照における明示的ネームスペースのブラケット構文
- `^[directory,system]globalname` ― 拡張グローバル参照における暗示的ネームスペースのブラケット構文
- `^["^"]globalname` ― プロセス・プライベート・グローバル
- `^["^",""]globalname` ― プロセス・プライベート・グローバル

注釈　既定では、グローバル名には上記の説明にある識別子文字のみを使用できます。ただし、NLS(各国言語サポート)の設定により、異なる識別子文字を定義することもできます。グローバル名には Unicode 文字を使用できません。

したがって、以下はすべて有効なグローバル名です。

ObjectScript

```
SET ^a="The quick "
SET ^A="brown fox "
SET ^A7="jumped over "
SET ^A.7="the lazy "
SET ^A1B2C3="dog's back."
WRITE ^a,^A,^A7,!,^A.7,^A1B2C3
KILL ^a,^A,^A7,^A.7,^A1B2C3 // keeps the database clean
```

2.1.2 グローバル・ノードと添え字の概要

グローバルは一般的に複数のノードを有しており、通常は添え字または添え字のセットにより識別されます。基本的な例は、以下のようになります。

ObjectScript

```
set ^Demo(1)="Cleopatra"
```

この文はグローバル・ノード ^Demo(1) を参照しており、これは ^Demo グローバル内の 1 つのノードです。このノードは 1 つの添え字によって識別されます。

別の例を示します。

ObjectScript

```
set ^Demo("subscript1","subscript2","subscript3")=12
```

この文はグローバル・ノード ^Demo("subscript1","subscript2","subscript3") を参照しており、これは同一グローバル内の別のノードです。このノードは 3 つの添え字によって識別されます。

さらに別の例を以下に示します。

ObjectScript

```
set ^Demo="hello world"
```

この文はグローバル・ノード ^Demo を参照しており、これは添え字を使用していません。

グローバルのノードは階層構造を形成します。ObjectScript では、この構造を活用するコマンドが用意されています。例えば、ユーザは 1 つのノードを削除すること、または 1 つのノードとそのノードのすべての子を削除することができます。詳細は、次の章を参照してください。

以下のセクションでは、添え字 およびグローバル・ノードの規則について詳細に解説します。

2.1.3 グローバル添え字

添え字には、以下の規則があります。

- 添え字の値では大文字と小文字が区別されます。
- 添え字の値はあらゆる ObjectScript の式とすることができますが、式は NULL 文字列 ("") に評価されないという条件が付きます。

  値には、空白、出力不能文字、Unicode 文字など全種類の文字を使用できます。(出力不能文字は、添え字値において実用性が低いことに注意してください。)
- グローバル参照を解決する前に、InterSystems IRIS はあらゆる他の式の評価と同じ方式で各添え字を評価します。以下の例では、^Demo グローバルの 1 つのノードを設定してから、いくつかの同等の方式にてそのノードを参照します。

```
SAMPLES>s ^Demo(1+2+3)="a value"

SAMPLES>w ^Demo(3+3)
a value

SAMPLES>w ^Demo(03+03)
a value

SAMPLES>w ^Demo(03.0+03.0)
a value

SAMPLES>set x=6

SAMPLES>w ^Demo(x)
a value
```

- InterSystems IRIS では、グローバル参照の合計の長さに制限を課しています。同様に、この制限によってあらゆる添え字の値の長さにも制限が課されます。詳細は、“グローバル参照の最大長”を参照してください。

注意　上記の規則は、InterSystems IRIS でサポートしている照合のすべてに適用されます。“pre-ISM-6.1”など、互換性の理由から引き続き使用されている古い形式の照合については、添え字に対する規則上の制限が多くなっています。例えば、文字の添え字では先頭に制御文字を使用できず、整数の添え字に使用できる数字の桁数にも制限があります。

2.1.4 グローバル・ノード

アプリケーションで、ノードは一般的に以下の構造タイプを含みます。

1. Unicode 本来の文字を含む文字列または数値データ。
2. 特殊文字で区切られた複数のフィールドを持つ文字列。

   ObjectScript

   ```
   SET ^Data(10) = "Smith^John^Boston"
   ```

   ObjectScript の $PIECE 関数を使用して、このようなデータを個別の値に切り離すことができます。
3. InterSystems IRIS の $LIST 構造に含まれた複数のフィールド。$LIST 構造は、複数の長さのエンコード値を含む文字列です。専用の区切り文字は必要ありません。
4. NULL 文字列 (""). 添え字自身がデータとして使用される場合、データは実際のノードに格納されません。
5. ビット文字列。ビットマップ・インデックスの一部を格納するためにグローバルを使用する場合、ノードに格納される値はビット文字列です。ビット文字列は、論理的で、圧縮された一連の 1 と 0 の値を含む文字列です。$Bit 関数を使用して、ビット文字列を構築できます。
6. 大規模な一連のデータの一部。例えば、オブジェクト・エンジンおよび SQL エンジンは、ストリーム (BLOB) を一連の 32 K シーケンシャル・ノードとしてグローバルに格納します。ストリーム・インタフェースにより、ストリームのユーザは、ストリームをこのように格納することを認識していません。

いかなるグローバル・ノードであっても、最大文字列長を超える極端に長い文字列を含めることはできない点に注意してください。"サーバ側プログラミングの入門ガイド" の “一般的なシステム制限”を参照してください。

2.1.5 グローバルの照合

グローバル内で、ノードは照合 (ソート) 順序で格納されます。

アプリケーションは通常、添え字として使用する値に変換を適用して、ノードを格納する順序を制御します。例えば、SQL エンジンで文字列値に対するインデックスを生成するとき、すべての文字列値は大文字に変換され、先頭に空白が付加されます。これにより、インデックスでは大文字と小文字が区別されず、数値が文字列として格納されていても、テキストとして照合されるようになります。

2.2 グローバル参照の最大長

グローバル参照 (つまり、特定のグローバル・ノードやサブツリーへの参照) の合計の長さは、エンコード文字数で最長 511 文字です (入力文字数で考えると、511 文字より少ない場合があります)。

任意のグローバル参照のサイズを慎重に決定する場合は、以下のガイドラインを使用します。

1. グローバル名の場合、1 文字ごとに 1 を加算します。
2. 純粋な数字の添え字の場合、1 桁、符号、または小数点ごとに、1 を加算します。
3. 数値以外の文字を含む添え字の場合、1 文字ごとに、3 を加算します。

   添え字が純粋な数字ではない場合、添え字の実際の長さは、文字列をエンコードするために使用される文字セットによって異なります。マルチバイトの文字は 3 バイトまでとすることができます。

   ASCII 文字は 1 または 2 バイトとなる可能性があることに注意してください。照合で大文字と小文字を区別する場合、ASCII 文字は、文字に対して 1 バイト、および曖昧性解消のバイトに対して 1 バイトとすることができます。照合で大文字と小文字を区別しない場合、ASCII 文字は 1 バイトとなります。
4. 添え字ごとに、1 を追加します。

これらの数の合計が 511 より大きくなった場合、その参照は長過ぎということになります。

制限は決まっているため、長い添え字名やグローバル名にする必要がある場合、多数の添え字レベルを避けることをお勧めします。反対に、複数の添え字レベルを使用している場合は、長いグローバル名や添え字を避けます。使用している文字セットを制御できない場合があるので、グローバル名や添え字は短くすることが有用となります。

特定の参照について懸念がある場合、最長となりそうなグローバル参照と同等の長さか、それより若干長いテスト・バージョンのグローバル参照を作成することをお勧めします。それらテストのデータにより、アプリケーション構築前に、名前付け規約の修正についてのヒントが得られます。

2.3 グローバルの物理構造

グローバルは、最適化された構造を使用して、物理ファイルに格納されます。このデータ構造を管理するコードも、InterSystems IRIS が実行するプラットフォームごとに高度に最適化されています。この最適化により、グローバルの処理では高いスループット (単位時間あたりの処理能力)、高水準の並行処理 (同時アクセスの総数)、そして Caché メモリの有効利用が可能となり、性能に対するメンテナンス (頻繁に行われてきた再構築、再度のインデックス付けや圧縮作業など) も一切必要ありません。

グローバルの格納に使用する物理構造は完全にカプセル化されるため、アプリケーションはどのような状況でも、物理データ構造に注意する必要はありません。

グローバルは、ディスク上の一連のデータ・ブロックに格納されます。各ブロックのサイズ (通常 8 KB) は、物理データベースを生成するときに決まります。データへの効率的なアクセスを実現するために、InterSystems IRIS は、一連のポインタ・ブロックを使用して関連データ・ブロックをリンクする高性能な B ツリー類似構造を保持します。InterSystems IRIS はバッファ・プール (頻繁に参照されるブロックを収めたメモリ内キャッシュ) を保持し、ディスクからブロックを取得するために要する負荷を軽減します。

多くのデータベース技術でデータ・ストレージには B ツリーに類似した構造が使用されていますが、InterSystems IRIS は多くの面で他とは異なる独自性を持っています。この独自性には次のようなものがあります。

- ストレージのメカニズムは、安全で利用しやすいインタフェースを通して公開されます。
- 添え字とデータは、ディスク・スペースと貴重なメモリ内キャッシュ・スペースを節約するために圧縮されます。
- ストレージ・エンジンは、トランザクション処理操作向けに最適化されているので、挿入、更新、削除の高速処理が可能です。リレーショナル・システムとは異なり、パフォーマンスのリストアのときのインデックスやデータの再構築は一切必要ありません。
- ストレージ・エンジンは、最適化により、できる限り多くの同時アクセスを可能にします。
- 効率よく変換するために、データを自動的にクラスタ化します。

2.4 グローバルの参照

グローバルは、特定の InterSystems IRIS データベース内に格納します。適切にマッピングされている場合、グローバルの一部を別のデータベースに置くことも可能です。データベースは、ECP のネットワーク経由でアクセスされるリモート・システムまたは現在のシステムに物理的に格納できます。データセットとは、InterSystems IRIS データベースを含むシステムとディレクトリを指します。リモート・データ・アクセスの詳細は、″スケーラビリティ・ガイド″ の “InterSystems 分散キャッシュによるユーザ数に応じたシステムの水平方向の拡張” の章を参照してください。

ネームスペースとは、関連性のある情報を構成するデータセットとグローバル・マッピングの論理的な定義です。

単純グローバル参照は、現在選択されているネームスペースに適用します。ネームスペースを定義することで、ローカル・システムあるいはリモート・システム上のデータベースに物理的にアクセスできるようになります。各グローバルは、それぞれ異なる位置やデータセットにマップできます (データセットとは、InterSystems IRIS データベースを含むシステムとディレクトリを指します)。

例えば、以下の構文を使用して、現在マップされているネームスペース内のグローバル ORDER へ単純参照を生成します。

```
^ORDER
```

このセクションでは、以下の 2 つのトピックについて説明します。

- グローバル・マッピングの設定
- 拡張グローバル参照

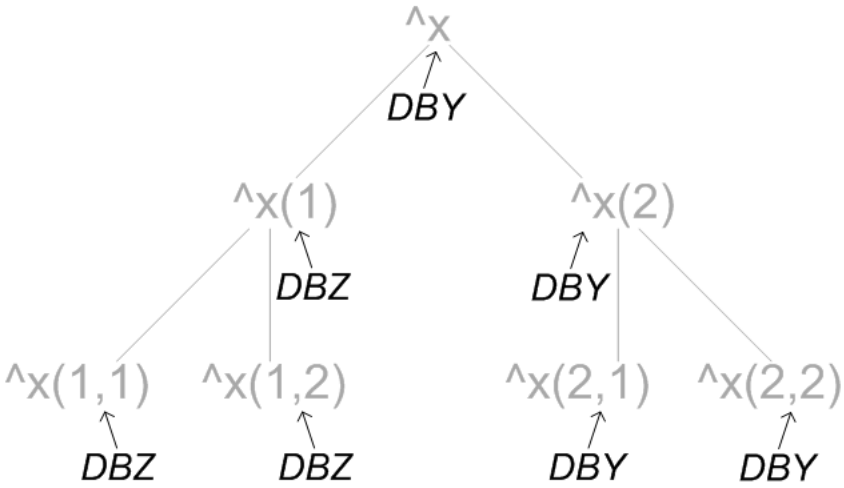
2.4.1 グローバル・マッピングの設定

同じシステムあるいは異なるシステム上に存在するデータベースから別のデータベースにグローバルとルーチンをマップできます。これによって、さまざまな場所に存在するデータを、簡単に参照できるようになります。グローバルは、その全体でも一部でもマップできます。グローバルの一部 (または添え字) のマッピングは、添え字レベル・マッピング (SLM) と呼ばれます。グローバル添え字をマップできるので、データはディスクを簡単に行き来できます。

このタイプのマッピングを行うには、″システム管理ガイド″ の “InterSystems IRIS の構成” の章の “ネームスペースへのグローバル、ルーチン、およびパッケージ・マッピングの追加” のセクションを参照してください。

グローバル・マッピングは、階層的に適用されます。例えば、NSX ネームスペースに関連付けられた DBX データベースがある一方、このネームスペースでは、^x グローバルを DBY データベースに、^x(1) を DBZ データベースにそれぞれマッピングしているとします。この場合、^x グローバルの添え字が付いたあらゆる形式のうち、^x(1) 階層に属していないものは DBY にマッピングされ、^x(1) 階層に属するグローバルは DBZ にマッピングされます。以下の図は、この階層を示しています。

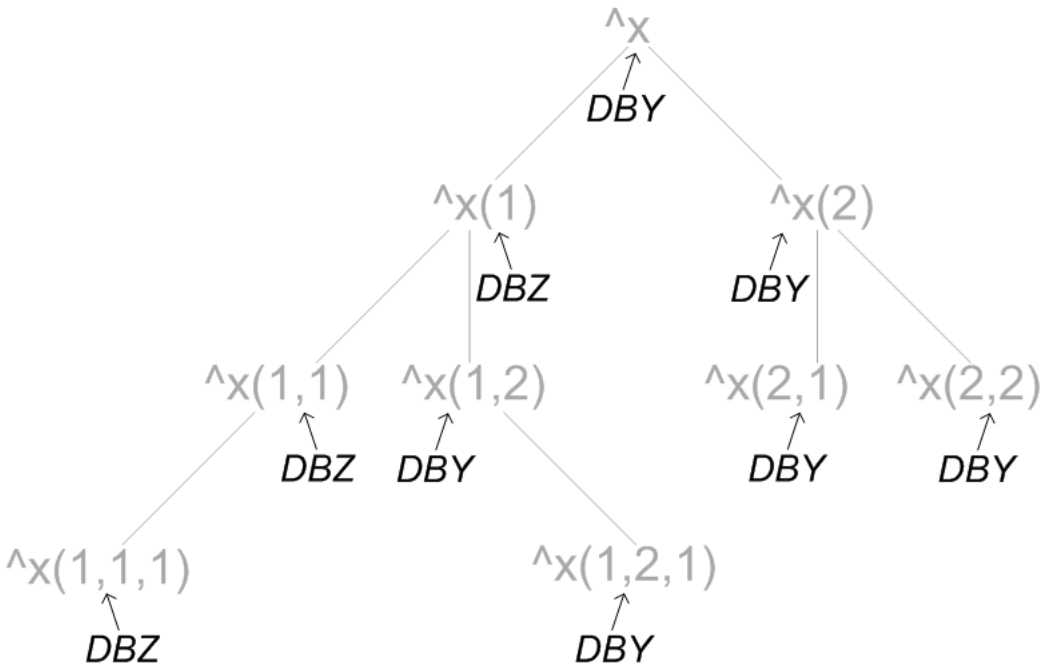
図 2-1: 単純な添え字レベル・マッピング



この図では、グローバルおよびその階層はグレーで表示され、これらのマッピング先のデータベースは黒で表示されています。

マッピング済みの添え字付きグローバルの一部を別のデータベースにマッピングすることも可能です。また、最初にグローバルがマッピングされていたデータベースにマッピング先を戻すことも可能です。前述の例で、^x(1,2) グローバルの追加のマッピング先を、元の DBY データベースに戻したとします。この場合は、以下のような状態になります。

図 2-2: より複雑な添え字レベル・マッピング



ここでも、グローバルおよびその階層はグレーで表示され、これらのマッピング先のデータベースは黒で表示されています。

あるネームスペースから別のネームスペースにグローバルをマップしておけば、そのグローバルが現在のネームスペースに存在するかのように、マップされたグローバルを ^ORDER や ^X(1) のように容易に参照できます。

重要　添え字レベル・マッピングの範囲を構築する場合、文字列の添え字の機能は整数の添え字の場合と異なります。文字列による添え字の場合は、最初の文字で範囲が決まりますが、整数の範囲は数値で指定します。例えば、添え字範囲 ("A"):("C") には、AA だけでなく AC や ABCDEF も含まれます。一方、添え字範囲 (1):(2) には 11 は含まれません。

2.4.1.1 グローバルおよび添え字の独立した範囲の使用

それぞれのネームスペースのマッピングでは、グローバルまたは添え字の独立した範囲を参照する必要があります。範囲に重複があるとマッピング検証で検出され、マッピングはできません。例えば、管理ポータルを使用して既存のマッピングと重複する新しいマッピングを作成しようとしても、ポータルによってこの重複が拒否され、エラー・メッセージが表示されます。

2.4.1.2 ログの変更

ポータルを使用してマッピングを正常に変更すると、それも messages.log に記録されますが、失敗した変更は記録されません。構成パラメータ (CPF) ファイルの手動編集でマッピングを設定しようとして失敗した場合は、messages.log に記録されます。CPF の編集に関する詳細は、"構成パラメータ・ファイル・リファレンス" の “構成パラメータ・ファイルの概要” の章にある “アクティブな CPF の編集” を参照してください。

2.4.2 拡張グローバル参照

現在のネームスペース以外のネームスペースに格納されているグローバルの参照も可能です。これを拡張グローバル参照、または省略して拡張参照と呼びます。

拡張参照には以下の 2 つの形式があります。

- 明示的なネームスペース参照 – グローバル参照構文の中で、グローバルを格納するネームスペース名を指定します。
- 暗黙のネームスペース参照 – グローバル参照構文の中でディレクトリを指定し、さらに必要に応じてシステム名を指定します。この場合、物理データセット (ディレクトリとシステム) がグローバル参照の一部として割り当てられるため、グローバル・マッピングは適応されません。

明示的なネームスペースを優先的に使用します。これにより、必要に応じて、アプリケーション・コードを変更せずに外部的に論理マッピングを再定義できます。

InterSystems IRIS は、以下の 2 つの拡張参照形式をサポートします。

- ブラケット構文 – 角括弧 ([ ]) で拡張参照を囲みます。
- 環境構文 – 垂直バー (| |) で拡張参照を囲みます。

注釈　拡張グローバル参照の例では、Windows のディレクトリ構造を使用しています。実用面では、このような参照の形式は、使用するオペレーティング・システムで決まります。

2.4.2.1 ブラケット構文

ブラケット構文を使用して、明示的あるいは暗黙のネームスペースで拡張グローバル参照を指定できます。

明示的なネームスペース

```
^[nspace]glob
```

暗黙ネームスペース

```
^[dir,sys]glob
```

明示的なネームスペース参照にある nspace は、現在、グローバル glob のマップ先にも複製先にもなっていない定義済みのネームスペースです。また、暗黙のネームスペース参照において、dir はディレクトリ（ディレクトリ名の最後には円記号“¥”を追加）、sys はシステム、glob はディレクトリ内のグローバルです。nspace または dir がキャレット（“^”）で指定されている場合、プロセス・プライベート・グローバルへの参照です。

ディレクトリとシステム名またはネームスペース名は、変数として指定しない限り引用符で囲みます。ディレクトリとシステムは共に暗黙のネームスペースを構成します。また、暗黙のネームスペースは以下のいずれかを参照できます。

- 指定されたシステムの指定ディレクトリ
- 参照でシステム名を指定していない場合、ローカル・システムで指定したディレクトリ。システム名を暗黙のネームスペース参照から削除する場合、ディレクトリ参照内に二重キャレット文字（^^）を置き、削除したシステム名であることを示す必要があります。

以下はリモート・システムに暗黙のネームスペースを指定します。

```
["dir","sys"]
```

以下はローカル・システムに暗黙のネームスペースを指定します。

```
["^^dir"]
```

例えば、以下は SALES というマシンの C:¥BUSINESS¥ ディレクトリにあるグローバル ORDER にアクセスします。

ObjectScript

```
SET x = ^["C:\BUSINESS\","SALES"]ORDER
```

以下は、ローカル・マシンの C:¥BUSINESS¥ ディレクトリにあるグローバル ORDER にアクセスします。

ObjectScript

```
SET x = ^["^^C:\BUSINESS\"]ORDER
```

MARKETING として定義済みのネームスペースでグローバル ORDER にアクセスします。

ObjectScript

```
SET x = ^["MARKETING"]ORDER
```

プロセス・プライベート・グローバル ORDER にアクセスするには：

ObjectScript

```
SET x = ^["^"]ORDER
```

注釈　ミラーリングされるデータベースを含む暗黙のネームスペース拡張参照を作成するときには、「:mirror:mirror_name:mirror_DB_name」の形式で、そのミラーリングされるデータベースのパスを使用できます。例えば、ミラー CORPMIR に含まれる mirdb1 というミラー・データベース名のデータベースを参照する場合は、以下に示すように暗黙の参照を作成できます。

```
["^^:mirror:CORPMIR:mirdb1"]
```

ミラーリングされるデータベースのパスは、ローカル・データベースとリモート・データベースの両方に使用できます。

2.4.2.2 環境構文

環境構文は以下のように定義されます。

```
^|"env"|global
```

″env″ は、次の 5 つの形式のうちのいずれかです。

- NULL 文字列 (″″) ー ローカル・システムの現在のネームスペース
- "namespace" ー global が現在マップされていない定義済みのネームスペース。ネームスペース名は、大文字と小文字を区別しません。namespace が "^" の特殊値である場合、これはプロセス・プライベート・グローバルとなります。
- "^^dir" ー暗黙のネームスペースであり、既定のディレクトリはローカル・システムの指定ディレクトリです。dir の最後には円記号 (“¥”) を追加します。
- "^system^dir" ー暗黙のネームスペースであり、既定のディレクトリは指定リモート・システムの指定ディレクトリです。dir の最後には円記号 (“¥”) を追加します。
- 省略 ー ″env″ がまったくない場合、これはプロセス・プライベート・グローバルです。

ORDER にマッピングが定義されていない場合、以下の構文を使用し、現システムのネームスペースでグローバル ORDER にアクセスします。

ObjectScript

```
SET x = ^|""|ORDER
```

これは、単純グローバル参照と同じです。

ObjectScript

```
SET x = ^ORDER
```

MARKETING として定義済みのネームスペースにマップされたグローバル ORDER にアクセスします。

ObjectScript

```
SET x = ^|"MARKETING"|ORDER
```

暗黙のネームスペースを使用して、ローカル・システムの C:¥BUSINESS¥ ディレクトリにあるグローバル ORDER にアクセスします。

ObjectScript

```
SET x = ^|"^^C:\BUSINESS\"|ORDER
```

暗黙のネームスペースを使用して、SALES というリモート・システムの C:¥BUSINESS ディレクトリにあるグローバル ORDER にアクセスします。

ObjectScript

```
SET x = ^|"^SALES^C:\BUSINESS\"|ORDER
```

プロセス・プライベート・グローバル ORDER にアクセスするには:

ObjectScript

```
SET x = ^||ORDER
SET x=^|"^"|ORDER
```

3

多次元ストレージの使用法 (グローバル)

この章では、多次元ストレージ (グローバル変数) を使用して実行できるさまざまな処理について説明します。

付録の “一時グローバルと IRISTEMP データベース” も参照してください。

3.1 グローバルへのデータ格納

グローバル・ノードへのデータの格納は単純で、グローバルを他の変数のように処理するだけです。違いは、グローバルに対する処理が自動的にデータベースに書き込まれる点です。

3.1.1 グローバルの生成

新しいグローバルを生成するための設定作業は一切必要ありません。グローバルにデータを設定するだけで、自動的に新しいグローバル構造が生成されます。グローバル (またはグローバル添え字) を生成し、1 回の演算でそこにデータを置くことができます。グローバル (または添え字) を生成し、NULL 文字列を設定すれば空のグローバルのままとなります。ObjectScript では、これらの処理は SET コマンドを使用して実行します。

以下の例は、Color という名のグローバルが存在しなければそれを定義し、値 “Red” と関連付けます。Color という名前のグローバルが既に存在する場合は、新規の情報を格納できるようにそのグローバルを修正します。

ObjectScript では以下のようになります。

ObjectScript

```
SET ^Color = "Red"
```

注釈　アプリケーション内でダイレクト・グローバル・アクセスを使用するとき、名前付け規約を作成し、アプリケーションの異なる部分同士でお互いの名前が “衝突” しないように規約に従ってください。これは、クラス、メソッド、他の変数に名前付け規約を作成するときと同様です。また、InterSystems IRIS® が使用する特定のグローバル名を避けてください。それらのリストについては、″サーバ側プログラミングの入門ガイド″ の付録 “識別子のルールとガイドライン” の “回避する必要があるグローバル変数名” セクションを参照してください。

3.1.2 グローバル・ノードに格納するデータ

グローバル添え字ノードに値を格納するには、他の変数配列と同様に、グローバル・ノードの値を設定します。指定したノードが存在していない場合は、そのノードを生成します。既に存在している場合、既存の値を新しい値に置き換えます。

式を使用して、グローバルにノードを指定します (グローバル参照)。グローバル参照は、キャレット文字 (^)、グローバル名、および必要に応じて 1 つ以上の添え字値で構成します。添え字は括弧 “( )” で囲み、コンマで区切ります。各添え字値は、それ自身がリテラル値、変数、論理式、グローバル参照などの式です。

グローバル・ノードの値の設定は、アトミック処理です。これは、正常に実行できることが保証されており、整合性を確保するためのロックを使用する必要もありません。

以下は、すべて有効なグローバル参照です。

ObjectScript では以下のようになります。

ObjectScript

```
SET ^Data = 2
SET ^Data("Color")="Red"
SET ^Data(1,1)=100        /* The 2nd-level subscript (1,1) is set
                             to the value 100. No value is stored at
                             the 1st-level subscript (^Data(1)). */
SET ^Data(^Data)=10       /* The value of global variable ^Data
                             is the name of the subscript. */
SET ^Data(a,b)=50         /* The values of local variables a and b
                             are the names of the subscripts. */
SET ^Data(a+10)=50
```

間接演算を使用して、実行時にグローバル参照を構築することもできます。

3.1.3 グローバル・ノードへの構造化されたデータ格納

各グローバル・ノードは最長 32K 文字の単独の文字列を含むことが可能です。

データは通常、以下のいずれかの方法でノード内に格納されます。

- 最長 32K 文字の 1 文字列 (正確には 32K マイナス 1 文字)。
- 文字で区切った複数のデータ部分で構成する文字列。

  区切り文字を使用してノード内にフィールド一式を格納するには、結合演算子 (_) を使用して値を連結します。以下の ObjectScript の例は、区切り文字として # 記号を使用しています。

  ObjectScript

  ```
  SET ^Data(id)=field(1)_"#"_field(2)_"#"_field(3)
  ```

  データ取得の際、$PIECE 関数を使用してフィールドを分離できます。

  ObjectScript

  ```
  SET data = $GET(^Data(id))
  FOR i=1:1:3 {
    SET field(i) = $PIECE(data,"#",i)
    }
  QUIT
  ```

- 複数のデータを含む、$LIST でコード化した文字列。

  $LIST 関数は、区切り文字を予約する必要がない特殊な長さエンコード法を使用します (これは InterSystems IRIS オブジェクトと SQL で使用する既定の構造です)。

  ノード内にフィールド一式を格納するには、$LISTBUILD 関数を使用してリストを構築します。

  ObjectScript

  ```
  SET ^Data(id)=$LISTBUILD(field(1),field(2),field(3))
  ```

  データ取得の際、$LIST 関数または $LISTGET 関数を使用してフィールドを分離できます。

ObjectScript

```
SET data = $GET(^Data(id))
FOR i = 1:1:3 {
    SET field(i)=$LIST(data,i)
    }
QUIT
```

- 大きなデータの一部 (ストリームや “BLOB”など)。

  個々のノードの容量が 32K に限られているので、例えばストリームのような大規模な構造は、連続した複数のノードにデータを格納することで実装します。

  ObjectScript

  ```
  SET ^Data("Stream1",1) = "First part of stream...."
  SET ^Data("Stream1",2) = "Second part of stream...."
  SET ^Data("Stream1",3) = "Third part of stream...."
  ```

  ストリームをフェッチするコード (%GlobalCharacterStream クラスで提供されるコードなど) は、連続した文字列としてデータを提供する構造で、継続的にノードをループします。

- 文字列。

  ビットマップ・インデックスを実装する場合は、インデックス・グローバルのノード値をビット文字列に設定します。ビットマップ・インデックスとは、ビット文字列のビットがテーブルの行に対応するインデックスです。InterSystems IRIS は、エンコーディング・ビット文字列に圧縮アルゴリズムを使用します。したがって、ビット文字列は、InterSystems IRIS の $BIT 関数を使用してのみ操作できます。ビット文字列に関する詳細は、“ビット文字列関数の概要” を参照してください。

- 空ノード。

  必要なデータがノード自体に収められている場合、一般的には実際の添え字を NULL 文字列 (“”) に設定します。例えば、名前を ID 値に関連付けるインデックスであれば、通常は以下のとおりです。

  ObjectScript

  ```
  SET ^Data("APPLE",1) = ""
  SET ^Data("ORANGE",2) = ""
  SET ^Data("BANANA",3) = ""
  ```

3.2 グローバル・ノードの削除

グローバル・ノード、サブノードのグループ、またはグローバル全体をデータベースから削除するには、ObjectScript の KILL コマンドまたは ZKILL コマンドを使用します。

KILL コマンドは、下位ノードを含むすべてのノード (データおよび配列内にあるそのエントリ) を特定のグローバル参照で削除します。つまり、指定した添え字で開始するすべてのノードが削除されます。

例えば、以下の ObjectScript 文を考えます。

ObjectScript

```
KILL ^Data
```

これは ^Data グローバル全体を削除します。このグローバルを引き続き参照すると <UNDEFINED> エラーを返します。

以下にもう 1 つ ObjectScript 文があります。

ObjectScript

```
KILL ^Data(100)
```

これは、^Data グローバルのノード 100 のコンテンツを削除します。^Data(100,1)、^Data(100,2)、^Data(100,1,2,3) などの下位ノードがある場合、同様に削除されます。

ObjectScript の ZKILL コマンドは、指定されたグローバルやグローバルの添え字ノードを削除します。下位ノードは削除しません。

注釈　大規模なグローバルを削除した後、そのグローバルで使用していた領域が完全には解放されていないことがあります。これは、そのブロックがガベージ・コレクタ・デーモンによってバックグラウンドで解放としてマークされているからです。したがって、大規模なグローバルを削除した直後に SYS.Database クラスの ReturnUnusedSpace メソッドを呼び出すと、そのグローバルで使用していたブロックがまだ解放されていないので、想定よりも少ない領域の値が返される場合があります。

グローバル変数には New コマンドを使用できません。

3.3 グローバル・ノードの存在有無のテスト

特定のグローバル (またはその下位ノード) がデータを含むかどうかをテストするには、$DATA 関数を使用します。

$DATA は、指定したグローバル参照が存在するか否かを示した値を返します。以下がその値の例です。

状態値	意味
0	グローバル変数が定義されていません。
1	グローバル変数が存在し、データを含みますが、下位ノードはありません。NULL 文字列 (″″) もデータと見なされます。
10	グローバル変数に下位ノードがありますが (下位ノードへの下方ポインタを含みます)、そのノード自身はデータを含みません。このような変数への直接参照は、<UNDEFINED> エラーになります。例えば、$DATA(^y) が 10 を返す場合、SET x=^y は <UNDEFINED> エラーを生成します。
11	データと下位ノードを含むグローバル変数です (下位ノードへの下方ポインタも含みます)。

3.4 グローバル・ノード値の検索

特定のグローバル・ノード内に格納された値を取得するには、グローバル参照を式として使用します。

ObjectScript

```
SET color = ^Data("Color")    ; assign to a local variable
WRITE ^Data("Color")          ; use as a command argument
SET x=$LENGTH(^Data("Color")) ; use as a function parameter
```

3.4.1 $GET 関数

$GET 関数を使用してグローバル・ノードの値を取得することもできます。

ObjectScript

```
SET mydata = $GET(^Data("Color"))
```

指定ノードの値を取得する (存在する場合) か、ノードに値がない場合は NULL 文字列 ("") を返します。$GET のオプションの 2 番目の引数を使用して、ノードに値がない場合に指定された既定値を返すこともできます。

3.4.2 WRITE コマンド、ZWRITE コマンド、ZZDUMP コマンド

ObjectScript のさまざまな表示コマンドを使用して、グローバルやグローバル・サブノードのコンテンツを表示できます。WRITE コマンドは、指定されたグローバルやサブノードの値を文字列として返します。ZWRITE コマンドは、グローバル変数名とその値、および下位ノードとその値を返します。ZZDUMP コマンドは、指定されたグローバルやサブノードの値を 16 進数ダンプ形式で返します。

3.5 グローバル内のデータ走査

グローバル内に格納されているデータの走査 (反復) には多くの方法があります。

3.5.1 $ORDER (Next / Previous) 関数

ObjectScript の $ORDER 関数では、グローバル内の各ノードに順にアクセスできます。

$ORDER 関数は、指定されたレベル (添え字番号) で次の添え字値を返します。例えば、以下のグローバルを定義するとします。

ObjectScript

```
Set ^Data(1) = ""
Set ^Data(1,1) = ""
Set ^Data(1,2) = ""
Set ^Data(2) = ""
Set ^Data(2,1) = ""
Set ^Data(2,2) = ""
Set ^Data(5,1,2) = ""
```

最初のファースト・レベル添え字を検索するには、以下を使用します。

ObjectScript

```
SET key = $ORDER(^Data(""))
```

これは、NULL 文字列 ("") に続けて最初のファースト・レベル添え字を返します (NULL 文字列は、最初のエントリの前の添え字値を表すために使用します。また、以降の添え字値がないことを示す返り値としても使用されます)。この例では、key が値 1 を含みます。

1、または $ORDER 式の key を使用して、2 番目のファースト・レベル添え字を検索できます。

ObjectScript

```
SET key = $ORDER(^Data(key))
```

key に初期値 1 がある場合、この文で 2 に設定されます (^Data(2) が次のファースト・レベル添え字であるため)。この文を再度実行すると、key は次のファースト・レベル添え字の 5 に設定されます。^Data(5) に直接格納されているデータはありませんが、5 を返すという点に注意してください。この文を再実行しても、これ以上ファースト・レベル添え字が存在しないため、key は NULL 文字列 ("") に設定されます。

追加の添え字を $ORDER 関数で使用することで、異なる添え字レベルを繰り返すことができます。$ORDER は、引数リストにある最終添え字の次の値を返します。以下の文は、上記のデータを使用した例です。

ObjectScript

```
SET key = $ORDER(^Data(1,""))
```

ここでは、^Data(1,1) が次のセカンド・レベルの添え字であるため、key が 1 に設定されます。この文を再度実行すると、key は次のセカンド・レベル添え字の 2 に設定されます。この文をもう一度実行しても、ノード ^Data(1) にはこれ以上セカンド・レベル添え字が存在しないため、key は “” に設定されます。

3.5.1.1 $ORDER によるループ

以下の ObjectScript コードは単純グローバルを定義し、そのファースト・レベル添え字をすべてループします。

ObjectScript

```
// clear ^Data in case it has data
Kill ^Data

// fill in ^Data with sample data
For i = 1:1:100 {
    // Set each node to a random person's name
    Set ^Data(i) = ##class(%PopulateUtils).Name()
}

// loop over every node
// Find first node
Set key = $Order(^Data(""))

While (key '= "") {
    // Write out contents
    Write "#", key, " ", ^Data(key),!

    // Find next node
    Set key = $Order(^Data(key))
}
```

3.5.1.2 追加の $ORDER 引数

ObjectScript の $ORDER 関数は、オプションとして 2 番目の引数や 3 番目の引数を持ちます。2 番目の引数は方向フラグで、グローバルを検索する方向を示します。既定値の 1 は前方検索を指定し、-1 は後方検索を指定します。

3 番目の引数はローカル変数名を指定します。$ORDER で見つかったノードがデータを含む場合、そのデータがこのローカル変数に書き込まれます。グローバルをループし、添え字値とノード値を取得する場合は、この引数を使用すると効率的です。

3.5.2 グローバルの反復

指定されたグローバルが連続値の添え字を使用する構成であることがわかっている場合は、単純な For ループでその値を使用して繰り返し処理が可能です。例えば、以下のように指定します。

ObjectScript

```
For i = 1:1:100 {
    Write ^Data(i),!
}
```

通常は、上記で説明した $ORDER 関数を使用することをお勧めします。その方が効率も良く、削除されたノードのようなデータ間の欠落部分を心配する必要もありません。

3.5.3 $QUERY 関数

サブノード間を移動して、グローバルにある各ノードと各サブノードにアクセスする場合は、ObjectScript の $QUERY 関数を使用します (または、入れ子にした $ORDER ループでも可能です)。

$QUERY 関数はグローバル参照をとり、グローバルにある次のノードのグローバル参照を含む文字列 (後にノードが続かない場合は "") を返します。$QUERY で返された値を使用するには、ObjectScript の間接演算子 (@) を使用する必要があります。

例えば、以下のグローバルを定義するとします。

ObjectScript

```
Set ^Data(1) = ""
Set ^Data(1,1) = ""
Set ^Data(1,2) = ""
Set ^Data(2) = ""
Set ^Data(2,1) = ""
Set ^Data(2,2) = ""
Set ^Data(5,1,2) = ""
```

以下は $QUERY の呼び出しです。

ObjectScript

```
SET node = $QUERY(^Data(""))
```

これは、node を文字列 "^Data(1)" に設定します。"^Data(1)" は、グローバルの最初のノードのアドレスです。その後、グローバルの次のノードを取得するには、$QUERY を再度呼び出し、node で間接演算子を使用します。

ObjectScript

```
SET node = $QUERY(@node)
```

この時点で、node は文字列 "^Data(1,1)" を含みます。

以下の例では、グローバル・ノードを設定した後、$QUERY を使用して検索し、各ノードのアドレスを記述します。

ObjectScript

```
Kill ^Data // make sure ^Data is empty

// place some data into ^Data
Set ^Data(1) = ""
Set ^Data(1,1) = ""
Set ^Data(1,2) = ""
Set ^Data(2) = ""
Set ^Data(2,1) = ""
Set ^Data(2,2) = ""
Set ^Data(5,1,2) = ""

// now walk over ^Data
// find first node
Set node = $Query(^Data(""))
While (node '= "") {
    Write node,!
    // get next node
    Set node = $Query(@node)
}
```

3.6 グローバル内でのデータのコピー

グローバルのコンテンツ (全体または一部) を別のグローバル (またはローカル配列) にコピーするには、ObjectScript の MERGE コマンドを使用します。

以下の例は、OldData グローバルのコンテンツ全体を NewData グローバルにコピーする MERGE コマンドの使用法を示しています。

ObjectScript

```
Merge ^NewData = ^OldData
```

MERGE コマンドのソース引数に添え字がある場合、そのノード内のすべてのデータと派生ノードがコピーされます。方向引数に添え字がある場合、宛先のアドレスを最上位ノードとして使用し、データをコピーします。以下はコードの例です。

ObjectScript

```
Merge ^NewData(1,2) = ^OldData(5,6,7)
```

これは、^OldData(5,6,7) と、その下にあるすべてのデータを ^NewData(1,2) にコピーします。

3.7 グローバルでの共有カウンタ保持

大規模なトランザクション処理では、一意の識別子を作成することで並行処理上の大きなボトルネックが発生することがあります。例えば新規送り状に、それぞれ一意の識別子番号を付ける注文処理作業を考えてみます。従来の方法としては、カウンタ・テーブルのようなものを保持します。新規送り状作成の各過程では、カウンタのロックを取得し、その値をインクリメントし、ロックの解放を行います。その結果、この単独のレコード上で、リソースが頻繁に競合することになります。

この問題を処理するために、ObjectScript の $INCREMENT 関数があります。$INCREMENT は、自動的にグローバル・ノードの値をインクリメントします (ノードに値がない場合は 1 に設定されます)。$INCREMENT の基本的な性質としてロックは不要です。他のプロセスからの干渉なしで、インクリメントされた値を返す機能が保証されているからです。

以下のようにして $INCREMENT を使用できます。まず、カウンタを保持するグローバル・ノードを決定します。次に、新規のカウンタ値が必要になるたびに $INCREMENT を実行します。

ObjectScript

```
SET counter = $INCREMENT(^MyCounter)
```

InterSystems IRIS オブジェクトと SQL で使用される既定のストレージ構造は、$INCREMENT を使用して固有のオブジェクト (行) 識別子の値を割り当てます。

3.8 グローバルでのデータのソート

グローバルに格納されたデータは、添え字値に従って自動的にソートされます。例えば、以下の ObjectScript コードは、グローバル一式を (順不同で) 定義し、繰り返すことにより、グローバル・ノードが添え字により自動的にソートされることを示します。

ObjectScript

```
// Erase any existing data
Kill ^Data

// Define a set of global nodes
Set ^Data("Cambridge") = ""
Set ^Data("New York") = ""
Set ^Data("Boston") = ""
Set ^Data("London") = ""
Set ^Data("Athens") = ""

// Now iterate and display (in order)
Set key = $Order(^Data(""))
While (key '= "") {
    Write key,!
    Set key = $Order(^Data(key)) // next subscript
}
```

グローバルが提供する自動ソートをアプリケーションで活用すると、ソート処理や、順序付けされた相互参照付インデックスを特定値に保持する処理などが可能です。InterSystems SQL と ObjectScript は、グローバルを使用してこのようなタスクを自動的に実行します。

3.8.1 グローバル・ノードの照合

グローバルのノードがソートされる順序 (照合と呼びます) は、グローバル自体とそのグローバルを使用しているアプリケーションの 2 段階で制御されます。

アプリケーション・レベルでは、添え字として使用される値のデータ変換を行うことで、グローバル・ノードの照合方法を制御できます (InterSystems SQL とオブジェクトはユーザ指定の照合機能でこれを実行します)。例えば、大文字小文字は関係なくアルファベット順でソートされた名前リストを生成する場合、一般的にその名前を大文字で表記して添え字として使用します。

ObjectScript

```
// Erase any existing data
Kill ^Data

// Define a set of global nodes for sorting
For name = "Cobra","jackal","zebra","AARDVark" {
    // use UPPERCASE name as subscript
    Set ^Data($ZCONVERT(name,"U")) = name
}

// Now iterate and display (in order)
Set key = $Order(^Data(""))
While (key '= "") {
    Write ^Data(key),!  // write untransformed name
    Set key = $Order(^Data(key)) // next subscript
}
```

この例は、添え字が大文字小文字を区別せずにソートされるように、名前をそれぞれ大文字に変換します ($ZCONVERT 関数を使用します)。元の値が表示されるように、各ノードには未変換の値を保持します。

3.8.2 数値添え字と文字列値添え字

数値は文字列値より先に照合されます。つまり、1 の値は “a” の値よりも先に処理されます。指定された添え字に対して数値と文字列値の両方を使用する場合は、この事実は認識しておく必要があります。(値を基にデータをソートするため) インデックスにグローバルを使用する場合、通常、値は数字 (例えば給与) としてソートするか、文字列 (例えば郵便番号) としてソートします。

数値的に照合されたノードに対する一般的な解決法としては、単項演算子 + を使用して、添え字値を強制的に数値にします。例えば、id 値を age でソートするインデックスを構築する場合、以下のように age が必ず数値になるように強制できます。

ObjectScript

```
Set ^Data(+age,id) = ""
```

値を文字列としてソートする場合は (例えば “0022”、“0342”、“1584”)、スペース文字 (“ ”) を付けることで、添え字値が必ず文字列となるように強制できます。例えば、id 値を zipcode (郵便番号) でソートするインデックスを構築する場合、以下のように zipcode が必ず文字列になるように強制できます。

ObjectScript

```
Set ^Data(" "_zipcode,id) = ""
```

これにより、“0022” など、先頭に 0 が付く値は常に文字列として扱われます。

3.8.3 $SORTBEGIN 関数と $SORTEND 関数

通常、InterSystems IRIS 内データのソートに関しては心配する必要はありません。SQL を使用するか直接グローバル・アクセスを使用するかによって、自動的にソート処理されます。

しかし、場合によっては、さらに効率的なソート処理が可能な場合もあります。特に、(1) 順不同 (つまりソートされていない状態) で多数のグローバル・ノードを設定する必要があり、(2) 結果グローバルの合計サイズが InterSystems IRIS バッファ・プールの大部分を占める場合は、(データがキャッシュに収まりきらないので) SET 処理の多くはディスクで処理されるようになります。この結果、パフォーマンスが悪影響を受けることがあります。通常は、一時グローバルの大容量データのロード、インデックスの集合、インデックスなしの値のソートなど、インデックス・グローバルの生成にかかわる場合、上記のような状況になります。

これらの状況に効率的に対処するため、ObjectScript では $SORTBEGIN 関数と $SORTEND 関数が用意されています。$SORTBEGIN 関数はグローバル (またはその部分) の特別ノードを初期化します。グローバルへのデータ・セットはスクラッチ・バッファに書き込まれ、メモリ (または一時ディスク・ストレージ) でソートされます。$SORTEND 関数が処理の最後に呼び出されると、データは実際のグローバル・ストレージに順に書き込まれます。ディスク処理を以前ほど要求されずに、書き込みが適切に終了しているため、操作全体がより効率的です。

$SORTBEGIN 関数の使用法は非常に簡単です。ソートを開始する前にソート対象のグローバル名で起動し、処理が完了した時点で $SORTEND を呼び出します。

ObjectScript

```
// Erase any existing data
Kill ^Data

// Initiate sort mode for ^Data global
Set ret = $SortBegin(^Data)

// Write random data into ^Data
For i = 1:1:10000 {
    Set ^Data($Random(1000000)) = ""
}

Set ret = $SortEnd(^Data)

// ^Data is now set and sorted

// Now iterate and display (in order)
Set key = $Order(^Data(""))
While (key '= "") {
    Write key,!
    Set key = $Order(^Data(key)) // next subscript
}
```

$SORTBEGIN 関数はグローバル作成の特殊なケースに対して設計されており、使用の際には注意が必要です。特に $SORTBEGIN モードの場合、書き込みをしているグローバルからの読み取りはできません。これは、データが書き込まれていないと、読み取りが正しく行われないためです。

InterSystems SQL は自動的にこれらの関数を使用して、一時インデックス・グローバルを作成します。これは、インデックスの付いていないフィールドでのソート処理などで使用します。

3.9 グローバルを使用した間接演算の使用法

間接演算を使用して、ObjectScript は実行時のグローバル参照の作成方法を提供します。これはプログラムの完了時、グローバル構造や名前がわからないアプリケーションにおいて便利です。

間接演算は間接演算子 @ でサポートされ、式を含んだ文字列をデリファレンスします。間接演算には @ 演算子の使用法によって複数のタイプがあります。

以下のコードは、グローバル参照を含む文字列をデリファレンスするときに @ 演算子を使用する、名前間接演算の例を提供します。

ObjectScript

```
// Erase any existing data
Kill ^Data

// Set var to an global reference expression
Set var = "^Data(100)"

// Now use indirection to set ^Data(100)
Set @var = "This data was set indirectly."

// Now display the value directly:
Write "Value: ",^Data(100)
```

添え字間接演算を使用して、式 (変数またはリテラル値) を間接演算文内で混在させることもできます。

ObjectScript

```
// Erase any existing data
Kill ^Data

// Set var to a subscript value
Set glvn = "^Data"

// Now use indirection to set ^Data(1) to ^Data(10)
For i = 1:1:10 {
    Set @glvn@(i) = "This data was set indirectly."
}

// Now display the values directly:
Set key = $Order(^Data(""))
While (key '= "") {
    Write "Value ",key, ": ", ^Data(key),!
    Set key = $Order(^Data(key))
}
```

間接演算は ObjectScript の基本機能で、グローバル参照に制限されません。詳細は、″ObjectScript の使用法″ の “演算子” の章の “間接演算” セクションを参照してください。間接演算は直接アクセスほど効果的ではないので、その点を考慮して使用してください。

3.10 トランザクション管理

InterSystems IRIS は、グローバルを使用した本格的なトランザクション処理の実装に必要な初期演算を提供します。InterSystems IRIS オブジェクトと SQL は、これらの機能を自動的に利用します。トランザクション用データをグローバルに直接書き込む場合に、これらの演算子を使用できます。

トランザクション・コマンドは、トランザクションの開始を定義する TSTART、現在のトランザクションを実行する TCOMMIT、現在のトランザクションを一時停止して、トランザクションの始めからのグローバルへの変更を元に戻す TROLLBACK です。

例えば、以下の ObjectScript コードは、トランザクションの開始を定義し、いくつかのグローバル・ノードを設定して、ok 値によってトランザクションを実行またはロールバックします。

ObjectScript

```
TSTART

Set ^Data(1) = "Apple"
Set ^Data(2) = "Berry"

If (ok) {
    TCOMMIT
}
Else {
    TROLLBACK
}
```

TSTART は、トランザクション開始マーカを InterSystems IRIS ジャーナル・ファイルに書き込みます。これは、トランザクション開始の境界線を定義します。変数 ok が上記の例で True (ゼロ以外) の場合、TCOMMIT コマンドはトランザクションの終わりをマークし、トランザクション終了マーカがジャーナル・ファイルに書き込まれます。ok が False (0) の場合、TROLLBACK コマンドはセットすべてを元に戻すか、またはトランザクションの開始時点からの演算を無効にします。この場合、^Data(1) と ^Data(2) は前の値にリストアされます。

トランザクションの正常終了の時点で、書き込まれるデータはありません。これは、トランザクション中のデータベースの変更すべてが、正常としてトランザクションの過程で実行されるためです。ロールバックの場合にのみ、データベースのデータに影響します。これは、この例のトランザクションが備えている分離が、限定された範囲にとどまることを意味します。つまり、トランザクションがコミットされていないうちは、修正されたグローバル値を他のプロセスからも見ることができます。これは通常、コミットされていない読み取りと呼ばれます。これが良いか悪いかは、アプリケーションの要件によって異なりますが、多くの場合では問題ありません。アプリケーションに強力な分離が必要な場合は、ロックを使用します。これは、以下のセクションで説明します。

3.10.1 ロックとトランザクション

トランザクションを分離し、修正されたデータがトランザクションのコミット前に他のプロセスから見えないようにするには、ロックを使用する必要があります。ObjectScript では、LOCK コマンドを使用して、ロックを直接取得および解放できます。ロックは規約に従って機能します。指定されたデータ構造 (永続オブジェクトに使用されるものなど) に対し、ロックを必要とするすべてのコードは、すべて同じ論理ロック参照 (例えば LOCK コマンドによって同じアドレスが使用される) を使用します。

トランザクションでは、ロックは独特の動作をします。つまり、トランザクション中に取得されたすべてのロックは、トランザクションが終了するまで解放されません。その理由を、通常のトランザクションで実行されるアクションで考えてみます。

1. TSTART を使用して、トランザクションを開始します。
2. 修正を希望するノード (単数または複数) で、ロック (単数または複数) を取得します。これは通常 “書き込み” ロックと呼ばれます。
3. ノード (1 つ、または複数) を変更します。
4. ロック (1 つ、または複数) を解放します。トランザクション中のため、これらのロックはこの時点では実際には解放されません。
5. TCOMMIT を使用して、トランザクションを終了します。この時点で、前の手順で解放したすべてのロックが実際に解放されます。

他のプロセスがこのトランザクションに関連するノードを見る際に、コミットされていない変更箇所が表示されないようにする場合は、そのノードからデータを読み取る前にロック (“読み取り” ロックと呼ばれます) のテストを行います。書き込み

ロックはトランザクション終了まで保持されているので、トランザクションが完了 (コミットまたはロールバック) するまで、読み取りプロセスにはデータが表示されません。

大半のデータベース管理システムが、類似した機能を使用して、トランザクション分離を提供します。InterSystems IRIS は、このメカニズムを開発者が使用できるという点がユニークです。トランザクションをサポートしているときでも、新規アプリケーション・タイプのカスタム・データベース構造の生成を可能にします。当然、InterSystems IRIS オブジェクトまたは SQL でデータの管理やトランザクション管理を自動的に行うことができます。

3.10.2 入れ子にされた TSTART の呼び出し

InterSystems IRIS には、特別なシステム変数 $TLEVEL があります。これは、TSTART コマンドを呼び出した回数をトラッキングします。$TLEVEL は値 0 から始まります。TSTART を呼び出すたびに $TLEVEL の値は 1 ずつインクリメントされます。また、TCOMMIT を呼び出すたびに値が 1 ずつディクリメントされます。TCOMMIT の呼び出しによって $TLEVEL が 0 に戻ると、トランザクションは (コミットされて) 終了します。

TROLLBACK コマンドへの呼び出しは常に、現在のトランザクションを終了させ、$TLEVEL をその値に関係なく元の 0 に設定します。

この動作をアプリケーションで利用すると、トランザクションを含むコード (オブジェクト・メソッドなど) を別のトランザクションで包むことができます。例えば、永続オブジェクトに備わっている %Save メソッドは、常に、トランザクションとしてのオペレーションを実行します。TSTART と TCOMMIT を明示的に呼び出すことで、複数のオブジェクト保存処理を含むさらに大きなトランザクションを生成することができます。

ObjectScript

```
TSTART
Set sc = object1.%Save()
If ($$$ISOK(sc)) {
    // first save worked, do the second
   Set sc = object2.%Save()
}

If ($$$ISERR(sc)) {
    // one of the saves failed, rollback
    TROLLBACK
}
Else {
    // everything is ok, commit
 TCOMMIT
}
```

3.11 並行処理の管理

シングル・グローバル・ノードの設定、または検索はアトミックです。 必ず、常に一定の結果が得られます。複数ノードの操作やトランザクションの分離制御について (″ロックとトランザクション″ のセクションを参照してください)、InterSystems IRIS はロックの取得機能と解放機能を提供しています。

ロックは、InterSystems IRIS ロック・マネージャで管理されます。ObjectScript では、LOCK コマンドを使用して、ロックを直接取得および解放できます (InterSystems IRIS オブジェクトと SQL は、ロックの取得と解放を必要に応じて自動的に行います)。

LOCK コマンドの詳細は、リファレンス・ページの “LOCK コマンド” を参照してください。″ロックと並行処理の制御″ も参照してください。

3.12 最新のグローバル参照のチェック

最新のグローバル参照は、ObjectScript の $ZREFERENCE 特殊変数に記録されます。$ZREFERENCE には、指定によって添え字と拡張グローバル参照など最新のグローバル参照が組み込まれています。$ZREFERENCE は、グローバル参照が成功したかどうか、あるいは指定グローバルが存在するかどうかを示すものではありません。InterSystems IRIS は、指定された最新のグローバル参照を記録しているにすぎません。

3.12.1 ネイキッド・グローバル参照

InterSystems IRIS は、添え字付きグローバル参照の後にグローバル名と添え字レベルを示すネイキッド・インジケータを設定します。その後、ネイキッド・グローバル参照を使用して、同じグローバルと添え字レベルに連続して参照を作成します。グローバル名と上位レベルの添え字は削除されます。これにより、同じ (または下位の) 添え字レベルの同じグローバルに対する参照の繰り返しを能率的に行います。

ネイキッド参照に下位の添え字レベルを指定すると、そのレベルに合わせてネイキッド・インジケータがリセットされます。したがって、ネイキッド・グローバル参照を使用する場合、常に最新のグローバル参照で構築した添え字レベルで作業することになります。

ネイキッド・インジケータ値は、$ZREFERENCE 特殊変数に記録されます。この値は NULL 文字列に初期化されます。ネイキッド・インジケータが設定されていない状態でネイキッド・グローバル参照を試みると、<NAKED> エラーが生じます。ネームスペースを変更すると、ネイキッド・インジケータも再初期化されます。$ZREFERENCE を NULL 文字列 ("") に設定することで、ネイキッド・インジケータを再初期化できます。

以下の例は、添え字付きのグローバル ^Produce("fruit",1) が最初の参照に指定されています。InterSystems IRIS は、このグローバル名と添え字をネイキッド・インジケータに保存します。したがって、後続のネイキッド・グローバル参照ではグローバル名 "Produce" と上位の添え字レベル "fruit" を省略できます。^(3,1) ネイキッド参照が下位の添え字レベルに移動した場合、この新規の添え字レベルが、その後に続くネイキッド・グローバル参照の条件となります。

ObjectScript

```
SET ^Produce("fruit",1)="Apples"  /* Full global reference  */
SET ^(2)="Oranges"                /* Naked global references */
SET ^(3)="Pears"                  /* assume subscript level 2 */
SET ^(3,1)="Bartlett pears"       /* Go to subscript level 3  */
SET ^(2)="Anjou pears"            /* Assume subscript level 3 */
WRITE "latest global reference is: ",$ZREFERENCE,!
ZWRITE ^Produce
KILL ^Produce
```

この例は、グローバル変数 ^Produce("fruit",1)、^Produce("fruit",2)、^Produce("fruit",3)、^Produce("fruit",3,1)、^Produce("fruit",3,2) を設定します。

例外はありますが、ほとんどのグローバル参照 (完全あるいはネイキッド) がネイキッド・インジケータを設定します。$ZREFERENCE 特殊変数は、ネイキッド・グローバル参照の場合でも、最新のグローバル参照の完全なグローバル名と添え字を保持しています。ZWRITE コマンドも、ネイキッド参照を使用して設定したかどうかにかかわらず、各グローバルの完全な名前と添え字を表示します。

ネイキッド・グローバル参照は注意して使用する必要があります。InterSystems IRIS は、ネイキッド・インジケータを常に明確に設定するわけではないからです。以下はその例です。

- ネイキッド・インジケータは、完全グローバル参照によって最初に設定されます。そのグローバル参照が失敗しても、それ以降の完全グローバル参照やネイキッド・グローバル参照によって、ネイキッド・インジケータが変更されます。例えば、存在しないグローバルの値を WRITE しようとすると、ネイキッド・インジケータが設定されます。
- 添え字付きのグローバルを参照するコマンド後置条件では、InterSystems IRIS が後置条件を評価する方法に関係なく、ネイキッド・インジケータが設定されます。

- 添え字付きのグローバルを参照するオプション関数の引数は、InterSystems IRIS がすべての引数を評価するかどうかによって、ネイキッド・インジケータを設定する場合としない場合があります。例えば、$GET の 2 番目の引数を指定すると、その既定値が使用されなくても、必ずネイキッド・インジケータが設定されます。InterSystems IRIS は、左から右の順番で引数を評価します。したがって、最後の引数は、最初の引数で設定されたネイキッド・インジケータをリセットする場合があります。
- トランザクションをロールバックする TROLLBACK コマンドは、ネイキッド・インジケータをトランザクションの最初の値にはロールバックしません。

完全グローバル参照に拡張グローバル参照を含む場合、その後に続くネイキッド・グローバル参照は、同じ拡張グローバル参照と見なされます。つまり、ネイキッド・グローバル参照の一部として拡張参照を指定する必要はありません。

4

多次元ストレージの SQL およびオブジェクトの使用法

この章では、InterSystems IRIS® オブジェクトおよび SQL エンジンが、どのように多次元ストレージ (グローバル) を利用して永続オブジェクト、リレーショナル・テーブル、インデックスを格納するかについて説明します。

データ・ストレージ構造は、InterSystems IRIS オブジェクトおよび SQL エンジンで自動的に提供および管理されますが、これらエンジンの動作を詳しく理解しておくことは無駄ではありません。

データのオブジェクト・ビューとリレーショナル・ビューで使用されるストレージ構造は同じものです。簡潔にするため、ここではオブジェクトの見地からのストレージのみ説明します。

4.1 データ

%Storage.Persistent ストレージ・クラス (既定) を使用する各永続クラスは、多次元ストレージ (グローバル) の 1 つ以上のノードを使用して InterSystems IRIS データベース内にそれ自体のインスタンスを格納できます。

各永続クラスには、プロパティのグローバル・ノードへの格納法を指定したストレージ定義があります。このストレージ定義 (“既定構造” と呼ばれます) は、クラス・コンパイラによって自動的に管理されます。このストレージ定義は変更でき、必要に応じて代替バージョンを提供できます。これについてはこのドキュメントでは説明しません。

4.1.1 既定構造

永続オブジェクトの格納に使用される既定構造は、非常に単純です。

- データは、グローバル名が完全なクラス名 (パッケージ名など) で始まるグローバル内に格納されます。データ・グローバル名には “D” を、インデックス・グローバルには “I” を付けて、それぞれの名前を作成します。
- 各インスタンスのデータは、$List 構造内に置かれた、すべての一時的でないプロパティと併せて、データ・グローバルのシングル・ノード内に格納されます。
- データ・グローバル内の各ノードは、オブジェクト ID 値で添え字が付けられています。既定では、オブジェクト ID 値は、データ・グローバルのルート (添え字なし) で格納されているカウンタ・ノードの $Increment 機能を呼び出すことにより提供される整数です。

例えば、2 つのリテラル・プロパティを持つ、単純な永続クラス MyApp.Person を定義するとします。

Class Definition

```
Class MyApp.Person Extends %Persistent
{
Property Name As %String;
Property Age As %Integer;
}
```

このクラスの 2 つのインスタンスを生成して保存すると、結果のグローバルは以下のようになります。

```
^MyApp.PersonD = 2  // counter node
^MyApp.PersonD(1) = $LB("",530,"Abraham")
^MyApp.PersonD(2) = $LB("",680,"Philip")
```

各ノードに格納されている $List 構造の最初の部分は空で、クラス名用に確保されます。この Person クラスのサブクラスのいずれかを定めると、このスロットはサブクラス名を含みます。(%Persistent クラスが提供する) %OpenId メソッドは、複数のオブジェクトが同じエクステント内に保存されている場合、この情報を使用して多様な形態で正しいタイプのオブジェクトを開きます。このスロットはクラス・ストレージ定義に、プロパティ名 “%%CLASSNAME” として表示されます。

詳細は、以下の ″サブクラス″ セクションを参照してください。

注意　エクステントの一部であるグローバルは、対応する ObjectScript および SQL コードにより管理されます。そのようなグローバルを直接グローバル・アクセスで変更すると、グローバルの構造が破壊されることや (そのデータがアクセス不能になる)、ObjectScript や SQL を通じてもデータにアクセスできなくなることがあります。

このような事態を回避するには、エクステントにあるすべてのデータの削除などのタスクで、グローバルに対して kill コマンドを使用しないようにします。代わりに、%KillExtent() や TRUNCATE TABLE などの API メソッドを使用します。これらのメソッドによって、関連付けられたメモリ内カウンタのリセットなどの重要なメンテナンスが実行されます。ただし、カスタマイズされたストレージ定義を使用してグローバルからデータを投影するクラスは、アプリケーション・コードで全面的に管理されているので、このルールの例外となります。そのようなクラスでは、READONLY を 1 に設定するか、MANAGEDEXTENT を 0 に設定するか、その両方の設定を検討します。

4.1.2 IDKEY

IDKEY 機能によって、オブジェクト ID として使用する値を明示的に定義できます。これを行うには、IDKEY インデックス定義をクラスに追加し、ID 値を提供するプロパティを指定します。保存したオブジェクトのオブジェクト ID 値は変更できません。つまり、IDKEY 機能を使用したオブジェクトを保存した後は、オブジェクト ID が基としているプロパティを変更することはできません。

例えば、IDKEY インデックスを使用するために上記の例で使用した Person クラスは変更できます。

Class Definition

```
Class MyApp.Person Extends %Persistent
{
Index IDKEY On Name [ Idkey ];

Property Name As %String;
Property Age As %Integer;
}
```

Person クラスの 2 つのインスタンスを生成し保存すると、結果のグローバルは以下のようになります。

```
^MyApp.PersonD("Abraham") = $LB("",530,"Abraham")
^MyApp.PersonD("Philip") = $LB("",680,"Philip")
```

定義されたカウンタ・ノードは、既に存在していないことに注意してください。また、Name プロパティに基づいてオブジェクト ID を設定することで、Name の値はオブジェクトごとに一意とすることも必要になります。

IDKEY インデックスが複数のプロパティに基づく場合、メイン・データ・ノードは複数の添え字を持ちます。例えば以下のようになります。

Class Definition

```
Class MyApp.Person Extends %Persistent
{
Index IDKEY On (Name,Age) [ Idkey ];

Property Name As %String;
Property Age As %Integer;
}
```

この場合、結果のグローバルは以下のようになります。

```
 ^MyApp.PersonD("Abraham",530) = $LB("",530,"Abraham")
 ^MyApp.PersonD("Philip",680) = $LB("",680,"Philip")
```

重要　プロパティが永続クラスのインスタンスに対する有効な参照ではない場合、IDKEY インデックスで使用するプロパティの値の中に、垂直バーの連続ペア(||)を入れることはできません。この制限は、InterSystems SQL のメカニズムが動作するための方法に起因しています。IDKey プロパティで || を使用すると、予測できない動作を起こす場合があります。

4.1.3 サブクラス

既定では、永続オブジェクトのサブクラスにより発生したフィールドは、追加ノードに格納されます。サブクラス名は、追加の添え字値として使用されます。

例えば、2 つのリテラル・プロパティを持つ、単純な永続クラス MyApp.Person を定義するとします。

Class Definition

```
Class MyApp.Person Extends %Persistent
{
Property Name As %String;

Property Age As %Integer;
}
```

ここで、2 つの追加リテラル・プロパティを取り込む、永続サブクラス MyApp.Student を定義します。

Class Definition

```
Class MyApp.Student Extends Person
{
Property Major As %String;

Property GPA As %Double;
}
```

この MyApp.Student クラスの 2 つのインスタンスを生成し保存すると、結果のグローバルは以下のようになります。

```
^MyApp.PersonD = 2  // counter node
^MyApp.PersonD(1) = $LB("Student",19,"Jack")
^MyApp.PersonD(1,"Student") = $LB(3.2,"Physics")

^MyApp.PersonD(2) = $LB("Student",20,"Jill")
^MyApp.PersonD(2,"Student") = $LB(3.8,"Chemistry")
```

Person クラスから派生したプロパティはメイン・ノードに格納され、Student クラスから取得したプロパティは追加サブノードに格納されます。この構造により、Student データと Person データを入れ替えて使用できるようになります。例えば、すべての Person オブジェクト名を扱っている SQL クエリは、Person データと Student データの両方を正確に取得します。また、この構造により、プロパティはスーパークラスかサブクラスのいずれかに追加されるため、クラス・コンパイラは、容易にデータ互換性を保持することができるようになります。

メイン・ノードの最初の部分は文字列“Student”を含みますが、これが、Student データが含まれるノードを識別します。

4.1.4 親子リレーションシップ

親子リレーションシップ内で、子オブジェクトのインスタンスは、それが属する親オブジェクトのサブノードとして格納されます。この構造により、子インスタンス・データが親データと物理的にクラスタ化するようになります。

例えば、以下は 2 つの関連したクラスの 1 つ、Invoice の定義です。

Class Definition

```
/// An Invoice class
Class MyApp.Invoice Extends %Persistent
{
Property CustomerName As %String;

/// an Invoice has CHILDREN that are LineItems
Relationship Items As LineItem  [inverse = TheInvoice, cardinality = CHILDREN];
}
```

次に、LineItem は以下のようになります。

Class Definition

```
/// A LineItem class
Class MyApp.LineItem Extends %Persistent
{
Property Product As %String;
Property Quantity As %Integer;

/// a LineItem has a PARENT that is an Invoice
Relationship TheInvoice As Invoice [inverse = Items, cardinality = PARENT];
}
```

Invoice オブジェクトの複数のインスタンスを、関連付けた LineItem オブジェクトと共に格納すると、結果のグローバルは以下のようになります。

```
^MyApp.InvoiceD = 2  // invoice counter node
^MyApp.InvoiceD(1) = $LB("","Wiley Coyote")
^MyApp.InvoiceD(1,"Items",1) = $LB("","Rocket Roller Skates",2)
^MyApp.InvoiceD(1,"Items",2) = $LB("","Acme Magnet",1)

^MyApp.InvoiceD(2) = $LB("","Road Runner")
^MyApp.InvoiceD(2,"Items",1) = $LB("","Birdseed",30)
```

リレーションシップについての詳細は、"クラスの定義と使用" の“リレーションシップ”の章を参照してください。

4.1.5 埋め込みオブジェクト

埋め込みオブジェクトは、まずシリアル化された状態に変換され (既定では $List で、オブジェクトのプロパティを含む構造です)、その後他のプロパティと同様に、そのシリアル状態で格納されます。

例えば、2 つのリテラル・プロパティを持つ、単純な連続した (埋め込み) クラスを定義するとします。

Class Definition

```
Class MyApp.MyAddress Extends %SerialObject
{
Property City As %String;
Property State As %String;
}
```

前述の例を変更して、埋め込みの Home アドレス・プロパティを追加します。

Class Definition

```
Class MyApp.MyClass Extends %Persistent
{
Property Name As %String;
Property Age As %Integer;
Property Home As MyAddress;
}
```

このクラスの 2 つのインスタンスを生成して保存すると、結果のグローバルは以下のようになります。

```
 ^MyApp.MyClassD = 2  // counter node
 ^MyApp.MyClassD(1) = $LB(530,"Abraham",$LB("UR","Mesopotamia"))
 ^MyApp.MyClassD(2) = $LB(680,"Philip",$LB("Bethsaida","Israel"))
```

4.1.6 ストリーム

グローバル・ストリームは、32,000 バイト未満のデータの塊に分けられ、その塊の集合とされます。それがシーケンシャル・ノードに書き込まれ、グローバルに格納されます。ファイル・ストリームは外部ファイルに格納されます。

4.2 インデックス

永続クラスは、1 つ以上のインデックスを定義できます。インデックスは、処理 (ソートや条件付き検索など) をさらに効率的にするために使用される追加のデータ構造です。InterSystems SQL は、クエリを実行するときに、このようなインデックスを使用します。InterSystems IRIS オブジェクト、および SQL は、挿入、更新、削除の処理が実行されるときに、インデックス内に自動的に正しい値を保持します。

4.2.1 標準インデックスのストレージ構造

標準インデックスは、順序付けられた 1 つ以上のプロパティ値を、プロパティを含むオブジェクトのオブジェクト ID 値に関連付けます。

例えば、2 つのリテラル・プロパティ、および Name プロパティにインデックスを持つ、単純な永続クラスの MyApp.Person を定義するとします。

Class Definition

```
Class MyApp.Person Extends %Persistent
{
Index NameIdx On Name;

Property Name As %String;
Property Age As %Integer;
}
```

この Person クラスの複数のインスタンスを生成し保存すると、結果のデータおよびインデックス・グローバルは以下のようになります。

```
 // data global
 ^MyApp.PersonD = 3  // counter node
 ^MyApp.PersonD(1) = $LB("",34,"Jones")
 ^MyApp.PersonD(2) = $LB("",22,"Smith")
 ^MyApp.PersonD(3) = $LB("",45,"Jones")


 // index global
 ^MyApp.PersonI("NameIdx"," JONES",1) = ""
 ^MyApp.PersonI("NameIdx"," JONES",3) = ""
 ^MyApp.PersonI("NameIdx"," SMITH",2) = ""
```

インデックス・グローバルについては、以下の点に注意してください。

1. 既定では、“I”（インデックス）が付いたクラス名をグローバル名として持つグローバルに配置します。
2. 既定では、最初の添え字文字がインデックス名です。これにより、衝突することなく、同じグローバルへ複数のインデックスを格納することができます。
3. 2 つ目の添え字は、照合されたデータ値を含みます。この場合、データは既定の SQLUPPER 照合機能を使用して照合されます。これによって、（大文字小文字の区別なしにソートするために）すべての文字を大文字に変換し、（強制的にすべてのデータを文字列として照合するために）空白文字を付加します。
4. 3 つ目の添え字は、インデックス付きのデータ値を含むオブジェクトのオブジェクト ID 値を含みます。
5. ノード自体は空の状態です。必要なデータはすべて添え字内にあります。データをインデックスと共に格納することを指定しているインデックス定義は、インデックス・グローバルのノードに配置されます。

すべての Person クラスを Name の順番でリストするクエリを初めとして、多数のクエリを満たすうえで十分な情報が、このインデックスに収められています。

4.3 ビットマップ・インデックス

ビットマップ・インデックスは標準インデックスと類似していますが、唯一異なる点は、インデックスが付いた値に対応するオブジェクト ID 値のセットを、一連のビット文字列を使用して格納することです。

4.3.1 ビットマップ・インデックスの論理処理

ビット文字列は、一連のビット（0 と 1 の値）を専用の圧縮形式で保持する文字列です。InterSystems IRIS には、ビット文字列を効率的に生成して使用するための関数が用意されています。以下はその説明です。

テーブル 4-1: ビット文字列の処理

関数	説明
$Bit	ビット文字列内のビットを設定または取得します。
$BitCount	ビット文字列内のビット数を数えます。
$BitFind	ビット文字列内で、あるビットが次に置かれている位置を見つけます。
$BitLogic	2 つ以上のビット文字列に対し、論理演算（AND、OR）を実行します。

ビットマップ・インデックスでは、ビット文字列内の並び位置が、インデックス付きテーブル内の行（オブジェクト ID 番号）に対応しています。ビットマップ・インデックスでは、指定された値が存在する行に対応する位置に 1 を持ち、その値が存在しない行に対応する位置に 0 を持つビット文字列が保持されます。ビットマップ・インデックスは、システムが割り当てた数値のオブジェクト ID を持つ、既定のストレージ構造を使用しているオブジェクトに対してのみ作用します。

例えば、以下のようなテーブルがあるとします。

ID	州	製品
1	MA	Hat
2	NY	Hat
3	NY	Chair
4	MA	Chair
5	MA	Hat

State 列および Product 列にビットマップ・インデックスがある場合、そのインデックスは次の値で構成されます。

State 列のビットマップ・インデックスは以下のビット文字列値を含みます。

MA	1	0	0	1	1
NY	0	1	1	0	0

State が“MA”である行に対応した場所 (1、4、5) の値は 1 です。

同様に、Product 列のビットマップ・インデックスは、以下のビット文字列値を含みます (インデックス内では値が大文字に置き換えて照合されています)。

CHAIR	0	0	1	1	0
HAT	1	1	0	0	1

InterSystems SQL エンジンは、これらのインデックスで保持されているビット文字列に対して、繰り返し、文字列内部のビット数のカウント、または論理的な組み合わせ (AND、OR) などによるさまざまな演算を実行できます。例えば、State が“MA”で、Product が“HAT”である行を見つけるには、SQL エンジンでは、該当する複数のビット文字列を論理 AND で結合するだけです。

システムは、これらのインデックスに加え、“エクステント・インデックス”と呼ばれる別のインデックスを保持します。これは、存在する行に対応するすべての位置に 1 を持ち、存在しない行 (削除された行など) に対応するすべての位置に 0 を持つインデックスです。これは否定演算子など、特定の演算に対して使用します。

4.3.2 ビットマップ・インデックスのストレージ構造

ビットマップ・インデックスでは、順序付けられた 1 つ以上のプロパティ値の集合を、そのプロパティ値に対応したオブジェクト ID 値を持つ 1 つ以上のビット文字列と関連付けます。

例えば、2 つのリテラル・プロパティを持つ単純な永続クラスの MyApp.Person を定義し、クラスの Age プロパティにビットマップ・インデックスを定義するとします。

Class Definition

```
Class MyApp.Person Extends %Persistent
{
Index AgeIdx On Age [Type = bitmap];

Property Name As %String;
Property Age As %Integer;
}
```

この Person クラスの複数のインスタンスを生成し保存すると、結果のデータおよびインデックス・グローバルは以下のようになります。

```
 // data global
 ^MyApp.PersonD = 3  // counter node
 ^MyApp.PersonD(1) = $LB("",34,"Jones")
 ^MyApp.PersonD(2) = $LB("",34,"Smith")
 ^MyApp.PersonD(3) = $LB("",45,"Jones")

 // index global
 ^MyApp.PersonI("AgeIdx",34,1) = 110...
 ^MyApp.PersonI("AgeIdx",45,1) = 001...

 // extent index global
 ^MyApp.PersonI("$Person",1) = 111...
 ^MyApp.PersonI("$Person",2) = 111...
```

インデックス・グローバルについては、以下の点に注意してください。

1. 既定では、“I”（インデックス）が付いたクラス名をグローバル名として持つグローバルに配置します。
2. 既定では、最初の添え字文字がインデックス名です。これにより、衝突することなく、同じグローバルへ複数のインデックスを格納することができます。
3. 2 つ目の添え字は、照合されたデータ値を含みます。これは数値データのインデックスのため、照合関数は適用されません。
4. 3 つ目の添え字は、チャンク番号を含みます。効率を上げるため、ビットマップ・インデックスはそれぞれがテーブルのおよそ 64,000 行の情報を含む一連のビット文字列に分けられます。これら各ビット文字列がチャンクと呼ばれます。
5. ノードには、このようなビット文字列が含まれます。

また、このテーブルにはビットマップ・インデックスが含まれるため、エクステント・インデックスが自動的に保持されます。このエクステント・インデックスは、インデックス・グローバル内に格納され、最初の添え字同様“$”文字の付いたクラス名を使用します。

4.3.3 ビットマップ・インデックスの直接アクセス

以下の例はクラス・エクステント・インデックスを使用して、保存されたオブジェクト・インスタンス（行）の合計数を計算します。$Order を使用して、エクステント・インデックスのチャンクを順番に処理します（各チャンクはおよそ 64,000 行の情報を含みます）。

Class Member

```
/// Return the number of objects for this class.<BR>
/// Equivalent to SELECT COUNT(*) FROM Person
ClassMethod Count() As %Integer
{
    New total,chunk,data
    Set total = 0

    Set chunk = $Order(^MyApp.PersonI("$Person",""),1,data)
    While (chunk '= "") {
        Set total = total + $bitcount(data,1)
        Set chunk = $Order(^MyApp.PersonI("$Person",chunk),1,data)
    }

    Quit total
}
```

5
グローバルの管理

管理ポータルには、グローバルの管理用ツールが用意されていますが、システム・クラスにも同じタスクのいくつかを実行できるメソッドが用意されています。この章では、それらのツールの使用方法を説明します。

グローバル・マッピングの定義に関する詳細は、"システム管理ガイド" の "ネームスペースの構成" の章を参照してください。

5.1 一般的なアドバイス

ObjectScript コマンド (SET、MERGE、KILL など) と同様に、ここで説明するツールによって、グローバルに直接アクセスして操作することができます。グローバル・アクセスを使用して削除や変更を行う場合、すべてのオブジェクトおよび SQL の整合性チェックを省略することになります。また、元に戻すためのオプションはありません。そのため、これらのタスクは非常に慎重に行うことが重要です (表示やエクスポートはデータベースに影響を与えないため、安全な操作です)。

注意　エクステントの一部であるグローバルは、対応する ObjectScript および SQL コードにより管理されます。そのようなグローバルを直接グローバル・アクセスで変更すると、グローバルの構造が破壊されることや (そのデータがアクセス不能になる)、ObjectScript や SQL を通じてもデータにアクセスできなくなることがあります。

このような事態を回避するには、エクステントにあるすべてのデータの削除などのタスクで、グローバルに対して kill コマンドを使用しないようにします。代わりに、%KillExtent() や TRUNCATE TABLE などの API メソッドを使用します。これらのメソッドによって、関連付けられたメモリ内カウンタのリセットなどの重要なメンテナンスが実行されます。ただし、カスタマイズされたストレージ定義を使用してグローバルからデータを投影するクラスは、アプリケーション・コードで全面的に管理されているので、このルールの例外となります。そのようなクラスでは、READONLY を 1 に設定するか、MANAGEDEXTENT を 0 に設定するか、その両方の設定を検討します。

この章で説明するツールの使用時には、以下のことに注意してください。

- InterSystems IRIS でどのグローバルが使用されるかを知っておくこと。これらすべてが "システム" グローバルとして扱われるとは限りません。つまり、その一部は **[システム]** チェック・ボックスにチェックを付けていない場合でも表示されます。これらのグローバルのいくつかには、コードが保存されます。これには、ユーザのコードも含まれます。

  "サーバ側プログラミングの入門ガイド" の付録 "識別子のルールとガイドライン" にあるセクション "回避する必要があるグローバル変数名" を参照してください。

- アプリケーションでどのグローバルが使用されるかを知っておくこと。

アプリケーションで直接グローバル・アクセスを実行しない場合でも、アプリケーションによってグローバルが使用されます。永続クラスを作成する場合は、それらのデータおよびすべてのインデックスがグローバルに格納され、(既定では) それらのグローバルの名前がクラス名に基づくことに注意してください。前の章の“データ”を参照してください。

5.2 グローバル・ページの概要

管理ポータルには [**グローバル**] ページがあり、このページでグローバルを管理できます。このページでは、次の操作を実行できます。

- 目的のグローバルに対応する行で [**表示**] を選択すると、そのグローバルを調べることができます。
- 目的のグローバルに対応する行で [**編集**] を選択すると、そのグローバルを変更できます。
- [**エクスポート**] を選択するとグローバルをエクスポートできます。
- [**インポート**] を選択するとグローバルをインポートできます。
- [**検索**] を選択すると、グローバル内の値を検索できます。
- [**置換**] を選択すると、グローバル内の値を置換できます。
- [**削除**] を選択するとグローバルを削除できます。

また、このページには、ルーチンやクラスを表示するためのオプションも用意されていますが、ここではこれらのオプションについては説明しません。

管理ポータルのホーム・ページからこのページにアクセスする手順は以下のとおりです。

1. [**システムエクスプローラ**]→[**グローバル**] を選択します。
2. 目的のネームスペースまたはデータベースを選択します。
   - [**検索**] リストから [**ネームスペース**] または [**データベース**] を選択します。
   - 表示されたリストで、目的のネームスペースまたはデータベースの名前を選択します。

   ネームスペースまたはデータベースを選択すると、ページが更新され、そのグローバルが表示されます。
3. 特定のグローバルを探していて、その名前が最初に見つからない場合は、以下の操作を行います。
   - 必要に応じて、検索マスクを指定します。そのためには、[**グローバル**] フィールドに値を入力します。文字列の末尾にアスタリスク“*”を付けると、そのアスタリスクはワイルドカードとして処理され、アスタリスクの前の文字列で始まる名前を持つすべてのグローバルがページに表示されます。

     値を入力したら、Enter キーを押します。
   - 必要に応じて、[**システム・アイテム**] を選択すると、すべてのシステム・グローバルが検索に含まれます。
   - 必要に応じて、[**SQL テーブル名を表示**] を選択して、グローバルのテーブルに [Table] および [Usage] の各列を追加します。グローバルが SQL テーブルで使用されている場合、これらの列にはそのテーブルの名前と使用法が表示されます。例えば、それがデータ/マスタ・マップであるかどうかや、インデックス・タイプなどの情報です。
   - 必要に応じて、[**ページサイズ**] から値を選択すると、任意のページにリストされるグローバルの数を制御できます。

5.3 グローバル・データの表示

[**グローバルデータ表示**] ページには、指定したグローバルのノードがリストされます。そのテーブルでは、最初の列に行番号、次の列にノード、右側の列に値が表示されます。このページには最初、グローバル内の最初の 100 のノードが表示されます。

このページにアクセスするには、[グローバル] ページを表示し、グローバルの名前の横にある[**表示**] リンクを選択します。または、[**表示**] ボタンをクリックします。

このページでは、次の操作を実行できます。

- 検索マスクを指定します。そのためには、以下のように [**グローバル検索マスク**] の値を編集します。
  - 1 つのノードを表示するには、完全なグローバル参照を使用します。例 : `^Sample.PersonD(9)`
  - サブツリーを表示するには、右側の括弧のない部分的なグローバル参照を使用します。例 : `^%SYS("JOURNAL"`
  - 特定の添え字と一致するすべてのノードを表示するには、目的の添え字を含め、その他の添え字フィールドを空のままにします。例 : `^IRIS.Msg(,"en")`
  - 特定の添え字と一致するすべてのサブツリーを表示するには、前のオプションのような値を使用しますが、さらに右側の括弧を省略します。例 : `^IRIS.Msg(,"en"`
  - 特定の範囲の添え字と一致するノードを表示するには、添え字の場所に subscriptvalue1:subscriptvalue2 の形式で指定します。例 : `^Sample.PersonD(50:60)`

    前のオプションと同様に、右側の括弧を省略した場合は、サブツリーが表示されます。

  次に、[**表示**] をクリックするか、Enter キーを押します。
- 表示するノードの数に別の値を指定します。そのためには、[**最大行数**] に整数を入力します。
- 前の検索を繰り返します。そのためには、[**検索履歴**] ドロップダウンで検索マスクを選択します。
- [**編集を許可**] を選択して、データを編集可能にします。次のトピックを参照してください。

このページを閉じるには、[**キャンセル**] をクリックします。

5.4 グローバルの編集

注意　　編集する前に、InterSystems IRIS で使用されているグローバルと、アプリケーションで使用されているグローバルについて必ず把握してください。“一般的なアドバイス” を参照してください。元に戻すためのオプションはありません。そのため、変更したグローバルはリストアできません。

[**グローバルデータ編集**] ページでは、グローバルを編集できます。そのテーブルでは、最初の列に行番号、次の列にノード、右側の列に値が表示されます (青色の下線は、その値が編集可能であることを示します)。このページには最初、グローバル内の最初の 100 のノードが表示されます。

このページにアクセスして使用するには、次の手順を実行します。

1. [グローバル] ページを表示します。
2. グローバルの名前の横にある[**編集**] リンクを選択します。
3. 必要に応じて、[**グローバル検索マスク**] フィールドを使用して、表示される内容を絞り込みます。“グローバル・データの表示” を参照してください。

4. 必要に応じて、表示するノードの数に別の値を指定します。そのためには、[**最大行数**] に整数を入力します。
5. 必要であれば、それに対応する添え字を選択して、編集する値に移動します。
6. 編集する値を選択します。

   すると、以下のように 2 つの編集可能なフィールドが表示されます。

   - 上のフィールドには、編集するノードの完全なグローバル参照が含まれます。例：`^Sample.PersonD("18")`

     これを編集して、別のグローバル・ノードを参照することができます。その場合、その操作によって影響を受けるのは、新しく指定したグローバル・ノードです。
   - 下のフィールドには、このノードの現在の値が含まれます。以下はその例です。

     ```
     $lb("",43144,$lb("White","Orange"),$lb("8262 Elm Avenue","Islip","RI",57581),"Rogers,Emilio
     L.",
     $lb("7430 Washington Street","Albany","GA",66833),"650-37-4263","")
     ```

   必要に応じて、値を編集します。
7. 編集する場合は、[**保存**] クリックして変更を保存します。それ以外の場合は、[**キャンセル**] をクリックします。

また、ノードを削除する手順は以下のとおりです。

1. 必要に応じて、[**削除時にグローバルサブノードを削除**] を選択します。
2. [**削除**] をクリックします。
3. [OK] をクリックし、この操作を確定します。

また、この章で後述する“大規模な置換の実行”も参照してください。

5.5 グローバルのエクスポート

注意　グローバルのインポート(回復不可能な変更)は非常に簡単なため、最善策としてインポートする必要のあるグローバルのみをエクスポートします。すべてのグローバルをエクスポートすると、コードを格納しているすべてのグローバルがエクスポートに含まれる点に注意してください。InterSystems IRIS で使用されているグローバルと、アプリケーションで使用されているグローバルについて必ず把握してください。“一般的なアドバイス”を参照してください。

[**グローバルエクスポート**] ページでは、グローバルをエクスポートできます。

このページにアクセスして使用するには、次の手順を実行します。

1. [グローバル] ページを表示します。
2. 操作対象のグローバルを指定します。これを行うには、“グローバル・ページの概要”セクションの手順 2 と 3 を参照してください。
3. [**エクスポート**] ボタンをクリックします。
4. グローバルのエクスポート先のファイルを指定します。ファイルを指定するには、そのファイルの絶対パス名または相対パス名を [**エクスポートするファイルのサーバ <ホスト名> 上のパスと名前を入力する**] フィールドに入力するか、[**参照**] をクリックして目的のファイルに移動します。
5. [**文字セット**] リストを使用して、エクスポートするファイルの文字セットを選択します。
6. ページ中央のボックスで、以下の操作を行います。

- [出力形式] を選択します。
- [レコード形式] を選択します。

7. [削除をバックグラウンドで実行する場合はここをチェックします] にチェックを付けるか、チェックを外します。
8. [エクスポート] をクリックします。
9. そのファイルが既に存在する場合は、[OK] をクリックして、そのファイルを新しいバージョンで上書きします。

このエクスポートによって、.gof ファイルが作成されます。

5.6 グローバルのインポート

注意　グローバルをインポートする前に、InterSystems IRIS で使用されているグローバルと、アプリケーションで使用されているグローバルについて必ず把握してください。“一般的なアドバイス”を参照してください。元に戻すためのオプションはありません。既存のグローバルにグローバルをインポート(つまり、そのデータをマージ) した後に、そのグローバルを前の状態にリストアする方法はありません。

[グローバルインポート] ページでは、グローバルをインポートできます。このページにアクセスして使用するには、次の手順を実行します。

1. [グローバル] ページを表示します。
2. [インポート] ボタンをクリックします。
3. インポートするファイルを指定します。ファイルを指定するには、そのファイルの絶対パス名または相対パス名を [インポートするファイルのパスと名前を入力する] フィールドに入力するか、[参照] をクリックして目的のファイルに移動します。
4. [文字セット] リストを使用して、インポート・ファイルの文字セットを選択します。
5. [次へ] を選択します。
6. このテーブル内のチェック・ボックスを使用して、インポートするグローバルを選択します。
7. 必要に応じて、[バックグラウンドでインポートを実行] にチェックを付けます。このオプションにチェックを付けると、タスクがバックグラウンドで実行されます。
8. [インポート] をクリックします。

5.7 グローバル内の値の検索

[グローバル文字列検索] ページでは、添え字または選択したグローバルの値に含まれる特定の文字列を検索できます。

このページにアクセスして使用するには、次の手順を実行します。

1. [グローバル] ページを表示します。
2. 作業対象のグローバルを選択します。これを行うには、“グローバル・ページの概要”セクションの手順 2 と 3 を参照してください。
3. [検索] ボタンをクリックします。
4. [検索対象] に検索する文字列を入力します。

5. 必要に応じて、[**大文字小文字を区別**] のチェックを外します。検索では、既定で大文字と小文字が区別されます。
6. [**最初を検索**] または [**すべて検索**] のどちらかをクリックします。

   ページに、選択したグローバル内にある、指定した文字列を含む添え字または値を持つ、最初のノードまたはすべてのノードが表示されます。テーブルには、左側にノードの添え字、右側に対応する値が表示されます。
7. [**最初を検索**] を使用した場合は、必要に応じて [**次を検索**] をクリックすると、次のノードを表示できます。
8. 完了したら、[**ウインドウを閉じる**] をクリックします。

5.7.1 大規模な置換の実行

注意　編集する前に、InterSystems IRIS で使用されているグローバルと、アプリケーションで使用されているグローバルについて必ず把握してください。“一般的なアドバイス”を参照してください。このオプションでは、データが永久に変更されます。プロダクション・システムでの使用はお勧めしません。

開発での使用を目的として、[**グローバル文字列検索**] ページには、グローバル・ノード内の値の大規模な変更を行うためのオプションも用意されています。このオプション使用するには、以下の操作を実行します。

1. [グローバル] ページを表示します。
2. 作業対象のグローバルを選択します。これを行うには、“グローバル・ページの概要”セクションの手順 2 と 3 を参照してください。
3. [**置換**] ボタンをクリックします。
4. 前のセクションの説明に従って、このページを使用して値を検索します。
5. [**置換文字列**] に値を指定します。
6. [**すべて置換**] をクリックします。
7. [OK] をクリックし、この操作を確定します。

   すると、ページに変更のプレビューが表示されます。
8. その結果に問題がなければ、[**保存**] をクリックします。
9. [OK] をクリックし、この操作を確定します。

5.8 グローバルの削除

注意　グローバルを削除する前に、InterSystems IRIS で使用されているグローバルと、アプリケーションで使用されているグローバルについて必ず把握してください。“一般的なアドバイス”を参照してください。元に戻すためのオプションはありません。そのため、削除したグローバルはリストアできません。

[**グローバル削除**] ページでは、グローバルを削除できます。このページにアクセスして使用するには、次の手順を実行します。

1. [グローバル] ページを表示します。
2. 作業対象のグローバルを選択します。これを行うには、“グローバル・ページの概要”セクションの手順 2 と 3 を参照してください。
3. [**削除**] ボタンをクリックします。
4. [OK] をクリックし、この操作を確定します。

5.9 管理タスク用 API

InterSystems IRIS には、この章で説明するタスクのいくつかを実行するための以下の API も用意されています。

- %SYSTEM.OBJ クラスには、以下のメソッドが用意されています。
  - Export() を使用すると、グローバルを XML ファイルにエクスポートできます。
  - Load() および LoadDir() を使用すると、XML ファイルに格納されているグローバルをインポートできます。

  いずれのメソッドも、$SYSTEM 変数を使用して使用できます。例：$SYSTEM.OBJ.Export
- %Library.Global クラスには、以下のメソッドが用意されています。
  - Export() を使用すると、グローバルを ..gof などのファイル形式 (XML を除く) にエクスポートできます。
  - Import() を使用すると、グローバルを ..gof などのファイル形式 (XML を除く) にインポートできます。

  %Library.Global には、Get() クラス・クエリも用意されています。これを使用すると、検索条件を指定してグローバルを検索できます。

追加の API へのポインタについては、"インターシステムズ・プログラミング・ツールの索引" の “グローバル” を参照してください。

A

一時グローバルと IRISTEMP データベース

特定の処理に対して、データを無期限に保存する必要なく、グローバルの強力な性能が必要になる場合もあります。例えば、グローバルを使用して、ディスクに保存する必要のないデータをソートするとします。このような処理に備え、InterSystems IRIS には一時グローバルのメカニズムが用意されています。

一時グローバルには、以下のような特性があります。

- 一時グローバルは IRISTEMP データベース内に保存されます。このデータベースは常にローカル・データベース (つまり、非ネットワーク・データベース) として定義されます。IRISTEMP データベースにマッピングされるグローバルはすべて一時グローバルとして扱われます。
- 一時グローバルに対する変更はディスクに書き込まれません。代わりに、それらの変更はメモリ内バッファ・プールに保持されます。バッファ・プール内に十分な容量がない場合、サイズの大きい一時グローバルはディスクに書き込まれることがあります。
- 効率性を最大限にするために、一時グローバルに対する変更はジャーナル・ファイルに記録されません。
- 一時グローバルは、InterSystems IRIS の再起動時に常に自動的に削除されます (注意:ライブ・システムを再起動する時間がなかなか取れない場合があります。このため、一時グローバルを削除するためにこの方法に頼らないようにしてください)。

A.1 一時グローバルの使用法

一時グローバルを使用するための仕組みは以下のように機能します。

- アプリケーション・ネームスペースでは、特定の命名規則を持つグローバルが IRISTEMP データベースにマッピングされるように、グローバル・マッピングを定義します。このデータベースは、後述するように特別なデータベースです。

  例えば、^AcmeTemp* という形式の名前を持つすべてのグローバルが IRISTEMP データベースにマッピングされるように、グローバル・マッピングを定義することができます。
- コードで、一時的にデータを格納して再度読み取る必要がある場合、そのコードは、この命名規則を使用するグローバルとの間で読み書きを行います。

  例えば、値を保存する場合、コードで以下を実行します。

```
set ^AcmeTempOrderApp("sortedarray")=some value
```

その後、以下を実行します。

```
set somevariable = ^AcmeTempOrderApp("sortedarray")
```

一時グローバルを使用することにより、IRISTEMP データベースがジャーナルされないという事実を利用します。データベースはジャーナルされないので、データベースを使用する操作では、ジャーナル・ファイルは作成されません。ジャーナル・ファイルは大きくなり、スペースの問題を引き起こす可能性があります。ただし、以下の点について注意してください。

- IRISTEMP データベースでグローバルを変更するトランザクションをロールバックすることはできません。この動作は IRISTEMP に固有です。トランザクションを介して一時的な作業を管理する必要がある場合は、その目的のために IRISTEMP でグローバルを使用しないでください。
- IRISTEMP は、保存する必要のない作業にのみ使用するようにしてください。
- IRISTEMP データベースでより多くのメモリが必要な場合は、データベースのサイズが増えます。MaxIRISTempSizeAtStart パラメータを使用して IRISTEMP のサイズを管理できます。

A.2 一時グローバルのマッピングの定義

一時グローバルのマッピングを定義するには、以下の操作を実行します。

1. 命名規則を選択し、すべての開発者がそれを認識していることを確認します。以下の点に注意してください。
   - 多数の一時グローバルを用意するか、複数のノードを持つ少数の一時グローバルを用意するかを検討してください。InterSystems IRIS は、同等の数の別個のグローバルを読み書きするのと比べて、同じグローバル内の異なるノードを効率的に読み書きする方が簡単です。この効率の差は、グローバルの数が少ない場合は無視できますが、数百の別個のグローバルがある場合には顕著になります。
   - 複数のネームスペースで同じグローバル・マッピングを使用する場合は、あるネームスペースの作業が別のネームスペースの作業を妨げないようにシステムを開発してください。例えば、グローバル内のサブスクリプトとしてネームスペース名を使用できます。
   - 同様に、1 つのネームスペース内であっても、干渉を避けるために、コードの各部分が、異なるグローバルを使用するか、または同じグローバル内で異なるサブスクリプトを使用するようにシステムを開発してください。
   - システム予約グローバル名は使用しないでください。"サーバ側プログラミングの入門ガイド" の付録 "識別子のルールとガイドライン" にある "回避する必要があるグローバル変数名" を参照してください。
2. 管理ポータルで、**[ネームスペース]** ページに移動します (**[システム管理]→[構成]→[システム構成]→[ネームスペース]**)。
3. アプリケーション・ネームスペースの行で、**[グローバルマッピング]** をクリックします。
4. **[グローバルマッピング]** ページで、**[新規グローバルマッピング]** をクリックします。
5. **[グローバルデータベース位置]** で、IRISTEMP を選択します。
6. **[グローバル名]** で、アスタリスク (*) で終わる名前を入力します。名前の先頭にキャレットを含めないでください。

   例 : `AcmeTemp*`

   このマッピングにより、`AcmeTemp*` で始まる名前を持つすべてのグローバルが IRISTEMP データベースにマッピングされます。
7. [OK] をクリックします。

注釈　新規のマッピング行の最初の列に表示される [>>] の記号は、マッピングを編集可能で開いていることを示します。

8. InterSystems IRIS で使用するようにマッピングを保存するには、**[変更を保存]** をクリックします。

詳細は、"システム管理ガイド" の "ネームスペースの構成" の章を参照してください。

A.3 IRISTEMP のシステム使用

インターシステムズでは、一時システム・グローバルをスクラッチ・スペースとして使用します。例えば、特定のクエリ (ソート、グループ分け、集約の計算用など) の実行中に、一時インデックスとして使用します。これらのグローバルは自動的に IRISTEMP にマッピングされます。以下のようなものがあります。

- ^IRIS.Temp*
- ^CacheTemp*
- ^mtemp*

これらのグローバルは変更しないでください。

A.4 ^CacheTemp グローバル

従来、名前の先頭に "^CacheTemp" の付いたグローバルが一時グローバルとして使用されてきました。慣例として、これらのグローバルでは、先頭に "^CacheTempUser" の付いた名前を使用して、一時システム・グローバルとの競合が発生する可能性を回避しています。ただし、ベスト・プラクティスは、独自の一時グローバルを定義して IRISTEMP にマッピングすることです。これについては、"一時グローバルの使用法" で説明しています。