記事
Toshihiko Minamoto · 2021年3月4日 10m read

グローバルをクラスにマッピングする技術 (5/3)

マッピングの例

三連載で 4 記事目を書いてしまったら、これまでのハイライトとして 5
記事目を書かないわけにはいかないでしょう！

注意: 何年か前に Dan Shusman 氏が私に「グローバルのマッピングは芸術だ」と言いました。
そのやり方に正解も不正解もありません。
どのようなマッピングを行うかは、データをどう解釈するかで決まります。
例のごとく、最終的な結論を出す方法は 1 つに限られません。
ここでご紹介する例の中には、同じ型のデータを異なる方法でマッピングする例がいくつかあります。

この記事の最後には、私が長年お客様のために書いてきたマッピングの例をまとめた zip
ファイルをご用意しています。 過去 4
つの記事で触れた内容をまとめたハイライトとして、いくつか例を挙げていきたいと思います。
この記事は単なるハイライトですので、過去 4 記事ほどの詳細はカバーいたしません。
不明な点があれば、遠慮なくご連絡ください。もっと詳しく説明させていただきます。

Row ID Spec: クラス例: Mapping.RowIdSpec.xml

これについては、過去の記事で何度か断言していますが、
これを定義する必要があるのは、添え字の式が単純なフィールドではない場合に限ります。
ご紹介した例では、添え字に格納された値を 100 で乗算しましたので、グローバルを見ると必要なのは 1.01
ですが、論理値として欲しいのは 101 ということになります。 したがい、Subscript 2 には {ClassNumber}/100
という式があり、RowIdSpec は {L2}*100 となっています。 最初はいつも逆にやってしまいます。 Subscript
Expression は論理値を受け取り、それをグローバルの $ORDER() で使うので、100 で除算します。 一方の RowId
Spec は、グローバルから値を受け取り、論理値を作成しているので、100 で乗算します。

これは、Next Code に加え、乗算と除算を処理する Invalid Condition を書いて行うこともできます。

Subscript Expression Type: Other / Next Code / Invalid Condition / Data Access: クラス例:
Mapping.TwoNamespacesOneGlobal.xml

このクラスはまさに便利な機能満載です！ ご紹介したい機能の半分がこのクラスで使用されます。
このクラスは、同じグローバルを 2 つの異なるネームスペースでループします。
これらのネームスペースでは、添え字の同じ値が使用されるので、添え字レベルのマッピングは使用できません。
代わりに、Global Reference の拡張構文 ^|namespace|GlobalName を使用し、最初に USER
ネームスペースでグローバルをイテレーションしてから、今度は SAMPLES
ネームスペースで同じグローバルをイテレーションします。 このテーブルの IDKey は、Namespace と Sub1 の 2
つの要素で構成されます。

Subscript Expression Other: Subscript Level 1 では、$ORDER() や $PIECE()
を使わずに、ハードコーディングされた 2 つの値 USER と SAMPLES のどちらかに {L1} を設定します。
ここでは、グローバルが一切使用されていないため、タイプは ‘ Other ’ になっています。

Next Code: Subscript Expression のタイプ ‘ Other ’ を使用している場合は、Next Code を指定する必要があります

(これを実行するために複雑なコードを書いても構いませんが、どちらにしろコードを提供する必要があります)。 Next Code が最初に呼び出されるとき、{L1} は ‘ Start Value ’ に設定されます。デフォルトは空の文字列です。 ループが終了したら、Next Code は {L1} を空の文字列に設定します。 この例では、{L1} は 3 回呼び出され、「USER」、「SAMPLES」、「“ ”」に設定されます。 Subscript Level 2 の Next Code は、Subscript Level 1 が有効な値を 2 つ返した後、「“ ”」にリセットされます。 この Next Code は、拡張参照付きのグローバルに対し実行されるシンプルな $$ORDER() です。

Invalid Condition: Subscript Level 1 と 2 に Next Code があるということは、両方が Invalid Condition を必要とすることを意味します。 この Subscript Level の値を考慮すると、条件の評価結果として値が無効となれば、1 が返されます。 {L1} については、値が「USER」でも「SAMPLES」でもなければ、1 が返されます。 例えば、

SELECT * FROM Mapping.RowIdSpec WHERE NS = “ %SYS ”

を実行しても、{L1} の Invalid Condition が評価され、行は 1 つも返されません。

Data Access: グローバルの 2 つのプロパティ (例えば ^glo({sub1},{Sub2})) に基づいた IdKey を持つグローバルがあるとします。{Sub2} に対しループを開始する前に {Sub1} の値を指定している場合は、$DATA(^glo({Sub1}) を実行して前のレベルにデータがないかどうかをチェックします。 この例では、Subscript Level 1 にグローバルがないため、何をテストするのかという指示が必要になります。 Data Access 式は、^¦{L1}¦Facility となります。 次の例でも Data Access 式が必要になりますが、そちらの方が理解しやすいかもしれません。 これが分かりにくいと思う方は、次の例をご覧ください。

データアクセス / 行全体の参照: クラス例: Mapping.TwoNamespacesOneGlobal2.xml

このクラスは、前のクラスと同じデータをマッピングしています。 違うのは Subscript Level 1 です。 このクラスでは、ネームスペースの値をハードコードする代わりに、2 つ目のグローバルを設け、その中にループする必要があるネームスペース ^NS(“ Claims ” ,{NS}) を含めています。 これにより、マッピングが簡素化されるほか、クラスのマッピングを変更する代わりにグローバルを設定するだけで、新しいネームスペースを追加できるので柔軟性も増します。

データアクセス: マッピング内の ‘ Global ’ は ^NS として定義されていますが、それは、Subscript Level 1 と 2 でそのグローバルに対しループを実行するためです。 Subscript Level 3 では、^¦{NS}¦Facility({Sub1}) に切り替えます。 Next Code で別のグローバルを使用するには、それを ‘ Data Access ’ で定義する必要があります。 {L1} は ^NS グローバルで制約となっていただけで、^Facility グローバルで使用されてもいないので、単純に整列させています。 {L2} は、グローバルの拡張構文 ^¦{L2}¦Facility 内でネームスペース参照として使用されています。 このクラスでは、‘ Next Code ’ は定義されていません (前の例で必要なかったのもこのためです)。 クラスのコンパイラーが ‘ Data Access ’ を受け取り、それを使ってこのレベルで必要な $ORDER() を生成します。

Full Row Reference: これは、‘ Invalid Condition ” と似ていて、^¦{L2}¦Facility({L3}) のように IdKey のすべての部分が使用されている場合に、行を参照する目的で使用されます。 このクラスでは、‘ Full Row Reference ’ を定義しなくても大丈夫だと思いますが、定義しても損はありません。 グローバルをイテレートする前に添え字の論理値を変更する ‘ Next Code ’ がある場合は、‘ Full Row Reference ’ を定義することが必要になります。 例えば、先ほども触れましたが、RowIdSpec クラスには ‘ Next Code ’ を使うこともできたと思います。 そのアプローチをとっていたら、‘ Full Row Reference ’ は ^Mapping({L1},{L2}/100) となっていたでしょう。

Subscript の Access Type 'Global': クラス例: Mapping.TwoGlobals.xml

このクラスは、^Member と ^Provider という 2 つの別のグローバルからデータを表示しています。 前の例では、行にアクセスするのに、1 つのグローバルからはじめ、その後に別のグローバルに切り替えています。 今回は、行を含むグローバルが 2 つあり、 1 つ目のグローバルのすべての行をループしてから、もう 1 つのグローバルの行をループします。

Subscript Expression Type 'Global': 1 つ目の Subscript Level を 'Global' として定義した場合は、マップの Global Property を「*」に設定する必要があります。
このスタイルのマッピングは、Mapping.TwoNamespacesOneGlobal でも使用できたのではないかと思います。
このようなマッピングを作成するあなたは、まさにアーティストです！

{L1} の Access Type は‘ Global ’、そして‘ Next Code ’は「Membe」、「Provider」、「“ ”」を返します。 Access Type を‘ Global ’として定義すると、コンパイラーは {L1}
をグローバルの名前として使う必要があると分かるため、{L2} と {L3} は単なる添え字ということになります。
{L4} も添え字ではありますが、2
つのグローバルがプロパティを別の場所に格納するというやっかいな事実に対処するコードを持っています。

このクラスでは、マッピングの Data セクションでもう 1 つ興味深いことが見られます。 ^Member
グローバルだけをマッピングしていたなら、Subscript Level は 3 つだけ定義していたでしょう。また、Data
セクションの Node 値 は 4、5、6、7、8、および 9 になっていたと思います。 Zip Code が Node 9 または 16
にあることに対処するために、ベースノードを IdKey 4 もしくは 10 に追加し、Zip Code
に対するオフセットを取得するために Node 内で +6 を使用します。

アクセス変数: クラス例: Mapping.SpecialAccessVariable.xml

特殊アクセス変数 は、どの Subscript Level でも定義できるほか、 1 つのレベルに複数個設けることができます。
この例では、{3D1} という変数が 1 つ定義されています。 「3」は Subscript Level 3
を意味し、「1」はこのレベルで定義された最初の変数であることを意味ます。
コンパイラーはこのために一意の変数名を生成し、その定義と削除を行います。 変数は、‘ Next Code ’
を実行した後に定義されます。 この例では、Level 4 の‘ Next Code ’にある変数を使いたいと思ったので、Level 3
で既に定義しておきました。 この例では、無効な日付の値があればそれを「*」に変更し、かつループの同じ場所
に戻れるよう、ループのどの要素まで行ったのかを「覚えておくため」に {3D1} を使用しています。

Bitmaps: クラス例: Mapping.BitMapExample.xml

Bitmap インデックスは比較的新しいものですが、だからといって Cache SQL Storage
を使うアプリケーションには追加しない方がいいというわけでもありません。 Bitmap
インデックスは、単純な正の %Integer IdKey を持つクラスなら、どのクラスにでも追加できます。

‘ Type ’を「bitmap」として定義し、IdKey は添え字として含めません。 Bitmap を定義するときは、Bitmap Extent
が必要なことも覚えておきましょう。 何ら特別なものではありません。Extent は IdKey
の値のインデックスなので、添え字は必要ありません。また‘ Type ”は「bitmapextent」です。

このクラスには、Sets や Kills を使ってデータを変更する際に、Bitmap
インデックスを管理するために呼び出せるメソッドがあります。 SQL もしくは Objects
を使って変更を加えることができるなら、インデックスは自動的に管理されます。

確かに少しややこしい例も含まれています！ これらのクラスを見て、内容が理解できるでしょうか。
よく解らないという方は、Brendan@interSystems.com
までご連絡ください。頑張るアーティストの皆さんのためなら、いつでも喜んでサポートさせていただきます。

クラスの例はこちら。

マッピングに関する他の記事の内容をおさらいしたいという方は、以下のリンクをご利用ください。

グローバルをクラスにマッピングする技術 (1/3)

グローバルをクラスにマッピングする技術 (2/3)

グローバルをクラスにマッピングする技術 (3/3)

グローバルをクラスにマッピングする技術 (4/3)

#マッピング #SQL #オブジェクトデータモデル #Caché

ソースURL:
https://jp.community.intersystems.com/post/%E3%82%B0%E3%83%AD%E3%83%BC%E3%83%90%E3%83%AB%E3%82%92%E3%82%AF%E3%83%A9%E3%82%B9%E3%81%AB%E3%83%9E%E3%83%83%E3%83%94%E3%83%B3%E3%82%B0%E3%81%99%E3%82%8B%E6%8A%80%E8%A1%93-53