記事
Toshihiko Minamoto · 2020年11月18日 5m read

クラス、テーブル、グローバルとその仕組み

クラス、テーブル、グローバルとその仕組み

InterSystems IRIS を技術的知識を持つ人々に説明する際、私はいつもコアとしてマルチモデル
DBMSであることから始めます。

個人的には、それが（DBMSとして）メインの長所であると考えています。
また、データが格納されるのは一度だけです。 ユーザーは単に使用するアクセス API を選択するだけです。

- データのサマリをソートしたいですか？SQL を使用してください！
- 1 つのレコードを手広く操作したいですか？オブジェクトを使用してください！
- あなたが知っているキーに対して、1 つの値にアクセスしたりセットしたいですか？
  グローバルを使用してください！

これは短く簡潔なメッセージで、一見すると素晴らしく聞こえます。しかし、実際には intersystems IRIS
を使い始めるたユーザーには クラス、テーブル、グローバルはそれぞれどのように関連しているのだろうか？
互いにどのような存在なのだろうか？
データは実際にどのように格納されているのだろうか？といった疑問が生じます。

この記事では、これらの疑問に答えながら実際の動きを説明するつもりです。

パート 1. モデルに対する偏見。

データを処理するユーザーは多くの場合、処理対象のモデルに偏見を持っています。

開発者はオブジェクトで考えます。 このようなユーザーにとって、データベースとテーブルは CRUD（Create-Read-Update-Delete、ORM の使用が望ましい）を介して操作する箱のようなものですが、その基礎となる概念モデルはオブジェクトです（これは主に私たちのような多くのオブジェクト指向言語の開発者に当てはまります）。

一方、リレーショナル DBMS
に多くの時間を費やしているデータベース管理者は往々にしてデータをテーブルと見なしています。
この場合、オブジェクトはレコードの単なるラッパー扱いです。

また、InterSystems IRIS
では永続クラスはデータをグローバルに格納するテーブルでもあるため、いくつかの説明が必要になります。

パート 2. 具体例

次のような Point クラスを作成したとします。

```
Class try.Point Extends %Persistent [DDLAllowed]
{
    Property X;
    Property Y;
}
```

次のように DDL/SQL を使用して同じクラスを作成することもできます。

```
CREATE Table try.Point (
    X VARCHAR(50),
    Y VARCHAR(50))
```

コンパイル後、新しいクラスがグローバルにネイティブに格納されているデータをカラム（またはオブジェクト指向のユーザーの場合はプロパティ）にマッピングするストレージ構造を自動生成します。

```
Storage Default
{
<Data name="PointDefaultData">
    <Value name="1">
        <Value>%%CLASSNAME</Value>
    </Value>
    <Value name="2">
        <Value>X</Value>
    </Value>
    <Value name="3">
        <Value>Y</Value>
    </Value>
</Data>
<DataLocation>^try.PointD</DataLocation>
<DefaultData>PointDefaultData</DefaultData>
<IdLocation>^try.PointD</IdLocation>
<IndexLocation>^try.PointI</IndexLocation>
<StreamLocation>^try.PointS</StreamLocation>
<Type>%Library.CacheStorage</Type>
}
```

ここでは何が起きているのでしょうか？

下から順番に説明します（**太字**の単語が重要です。残りは無視してください）。

- Type - 生成されたストレージタイプ。この場合は永続オブジェクトのデフォルトストレージです。
- StreamLocation - ストリームを格納するグローバルです。
- IndexLocation - インデックス用のグローバルです。
- IdLocation - ID の自動インクリメントカウンターを格納するグローバルです。
- **DefaultData** - グローバルの値をカラム/プロパティにマッピングするストレージの XML 要素です。
- **DataLocation** - データを格納するグローバルです。

ここでは「DefaultData」が PointDefaultData となっていますので、その構造をもう少し詳しく見てみましょう。基本的に、グローバルノードは次の構造を持っていると言われています。

- 1 - %%CLASSNAME
- 2 - X
- 3 - Y

したがって、グローバルは次のようになると予想されます。

```
^try.PointD(id) = %%CLASSNAME, X, Y
```

しかし、グローバルを出力すると空になります。ここではデータを追加していなかったためです。

```
zw ^try.PointD
```

オブジェクトを 1 つ追加しましょう。

```
set p = ##class(try.Point).%New()
set p.X = 1
set p.Y = 2
write p.%Save()
```

すると、グローバルはこのようになります。

```
zw ^try.PointD
^try.PointD=1
^try.PointD(1)=$lb("",1,2)
```

ご覧のように、期待する構造 %%CLASSNAME, X, Y はオブジェクトの X プロパティと Y プロパティに対応する $lb("",1,2) とセットになっています（%%CLASSNAME はシステムプロパティですので無視してください）。

次のように SQL を使用してレコードを追加することもできます。

```
INSERT INTO try.Point (X, Y) VALUES (3,4)
```

すると、グローバルの内容は次のようになります。

```
zw ^try.PointD
^try.PointD=2
^try.PointD(1)=$lb("",1,2)
^try.PointD(2)=$lb("",3,4)
```

つまり、オブジェクトまたは SQL
を介して追加するデータは、ストレージ定義に従ってグローバルに格納されます（補足：PointDefaultData の X と Y を置き換えることでストレージ定義を手動で変更できます。その場合に新しいデータがどうなるかを確認してください！）。

では、SQL クエリを実行したい場合はどうなるのでしょうか？

```
SELECT * FROM try.Point
```

これは ^try.PointD グローバルを反復処理し、ストレージ定義（正確にはその PointDefaultData
部分）に基づいてカラムにデータを入力する ObjectScript コードに変換されます。

今度は変更を行います。 テーブルからすべてのデータを削除しましょう。

```
DELETE FROM try.Point
```

すると、この時点でグローバルの内容は次のようになります。

```
zw ^try.PointD
^try.PointD=2
```

ここでは ID カウンターのみが残っているため、新しいオブジェクト/レコードの ID は 3 になることに注意してください。 また、クラスとテーブルは引き続き存在します。

しかし、次を実行するとどうなるでしょうか。

```
DROP TABLE try.Point
```

これはテーブルとクラスを破棄し、グローバルを削除します。

```
zw ^try.PointD
```

皆さんがこの具体例に従い、グローバル、クラス、テーブルがどのように統合され、相互に補完しているかをより深く理解できたことを願っています。 手元の仕事に適切な API を使用すれば、開発がより高速かつアジャイルになり、バグが少なくなります。

#SQL #オブジェクトデータモデル #グローバル #リレーショナルテーブル #初心者 #InterSystems IRIS

ソースURL:
https://jp.community.intersystems.com/post/%E3%82%AF%E3%83%A9%E3%82%B9%E3%80%81%E3%83%86%E3%83%BC%E3%83%96%E3%83%AB%E3%80%81%E3%82%B0%E3%83%AD%E3%83%BC%E3%83%90%E3%83%AB%E3%81%A8%E3%81%9D%E3%81%AE%E4%BB%95%E7%B5%84%E3%81%BF