記事
Toshihiko Minamoto · 2021年7月1日 5m read

便利な自動生成メソッド

クラスのコンパイル時に、定義済みのプロパティ、クエリ、またはインデックスごとに対応する複数のメソッドが自動的に生成されます。 これらのメソッドは非常に便利です。 この記事では、その一部について説明します。

プロパティ

「Property」というプロパティを定義したとしましょう。 次のメソッドが自動的に利用できるようになります（太字のPropertyは変化する部分でプロパティ名になります）。

```
ClassMethod PropertyGetStored(id)
```

このメソッドは、データ型プロパティの場合は論理値を返し、オブジェクトプロパティの場合はIDを返します。これはクラスのデータグローバルへの、ラップされたグローバル参照であり、特異なプロパティ値を取得する上でもっとも手早い方法です。 このメソッドは、保存されたプロパティでのみ利用できます。

```
Method PropertyGet()
```

プロパティgetterである。 再定義可能です。

```
Method PropertySet(val) As %Status
```

プロパティsetterである。 再定義可能です。

オブジェクトプロパティ

オブジェクトプロパティの場合、IDとOIDアクセスに関連するいくつかの追加メソッドを利用できるようになります。

```
Method PropertySetObjectId(id)
```

このメソッドはIDでプロパティ値を設定するため、オブジェクトを開いてプロパティ値として設定する必要はありません。

```
Method PropertyGetObjectId()
```

このメソッドは、プロパティ値のIDを返します。

```
Method PropertySetObject(oid)
```

このメソッドは、OIDでプロパティ値を設定します。

```
Method PropertyGetObject()
```

このメソッドは、プロパティ値のOIDを返します。

データ型プロパティ

データ型プロパティの場合、異なる形式間で変換するためのほかのメソッドがいくつか利用できるようになります。

```
ClassMethod PropertyDisplayToLogical(val)
ClassMethod PropertyLogicalToDisplay(val)
ClassMethod PropertyOdbcToLogical(val)
ClassMethod PropertyLogicalToOdbc(val)
ClassMethod PropertyXSDToLogical(val)
ClassMethod PropertyLogicalToXSD(val)

ClassMethod PropertyIsValid(val) As %Status
```

valが有効なプロパティ値であるかどうかをチェックします。

```
ClassMethod PropertyNormalize(val)
```

正規化された論理値を返します。

注意事項

- 関係はプロパティであり、これらのメソッドで取得/設定できます。
- 入力値（val）は、形式変換メソッドを除き、必ず論理値です。

インデックス

「Index」というインデックスの場合、次のメソッドを自動的に利用できるようになります。

```
ClassMethod IndexExists(val) As %Boolean
```

このvalを持つオブジェクトが存在するかどうかに応じて、1または0を返します。valはインデックス付きのプロパティの論理値です。

一意のインデックス

一意のインデックスの場合、追加メソッドを利用できるようになります。

```
ClassMethod IndexExists(val, Output id) As %Boolean
```

このvalを持つオブジェクトが存在するかどうかに応じて、1または0を返します。valはインデックス付きのプロパティの論理値です。 また、オブジェクトIDがある場合は、第2引数として返します。

```
ClassMethod IndexDelete(val, concurrency = -1) As %Status
```

インデックスの値がvalのエントリを削除します。

```
ClassMethod IndexOpen(val, concurrency, sc As %Status)
```

インデックスの値がvalの既存のオブジェクトを返します。

注意事項:

a）インデックスは複数のプロパティに基づいている可能性があるため、メソッドシグネチャは、入力として複数の値を持つように変更されます。例として、次のインデックスを見てみましょう。

```
Index MyIndex On (Prop1, Prop2);
```

この場合、IndexExistsメソッドには次のシグネチャがあります。

```
ClassMethod IndexExists(val1, val2) As %Boolean
```

val1はProp1値に対応し、val2はProp2値に対応します。 その他のメソッドも同じロジックに従います。

b）Cachéは、IDフィールド（RowID）にインデックスを作成するIDKEYインデックスを生成します。
ユーザーが再定義することが可能で、複数のプロパティを含むこともできます。
たとえば、クラスにプロパティが定義されているかをチェックするには、次を実行します。

```
Write ##class(%Dictionary.PropertyDefinition).IDKEYExists(class, property)
```

c）すべてのインデックスメソッドは、論理値をチェックします。

d) ドキュメント

クエリ

「Query」というクエリの場合（単純なSQLクエリまたはカスタムクラスクエリのどちらでも構いません。これに関する記事をご覧ください）、Funcメソッドが生成されます。

```
ClassMethod QueryFunc(Arg1, Arg2) As %SQL.StatementResult
```

このメソッドは、クエリの反復処理に使用される%SQL.StatementResultを返します。 たとえば、SamplesネームスペースのSample.Personクラスには、1つのパラメーターを受け入れるByNameクエリがあり、
次のコードを使って、オブジェクトコンテキストから呼び出すことができます。

```
Set ResultSet=##class(Sample.Person).ByNameFunc("A")
While ResultSet.%Next() { Write ResultSet.Name,! }
```

また、GitHubには、これらのメソッドを実演するデモクラスがあります。

#Code Snippet #オブジェクトデータモデル #ベストプラクティス #Caché #InterSystems IRIS

ソースURL:
https://jp.community.intersystems.com/post/%E4%BE%BF%E5%88%A9%E3%81%AA%E8%87%AA%E5%8B%95%E7%94%9F%E6%88%90%E3%83%A1%E3%82%BD%E3%83%83%E3%83%89