クラス定義の構文とキーワードのリファレンス

Version 2024.1
2024-06-03

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

クラス定義の構文とキーワードのリファレンス
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

1

クラス定義の構文とキーワードの概要

このリファレンスは、クラス定義で使用する構文とキーワードの公式な説明です。特定のクラス・メンバに適用するすべての構造とキーワードは、目次を参照してください。

クラス定義は正式には ObjectScript の一部ではありません。むしろ、クラス定義の特定の部分内で ObjectScript を使用することができます (特に、メソッド定義内。ここでは、その他の実装言語を使用することもできます)。

ここでは、このリファレンスで説明されているクラス定義の構造について、簡単に説明します。

1.1 クラス定義

クラス定義ではクラスを定義します。クラス定義は、クラス・メンバ (プロパティやメソッドなど)、キーワードと呼ばれるその他の項目、およびそれぞれの対応する値から成り、これらによってクラスの振る舞いの詳細を指定します。また、InterSystems IRIS では、トリガ、クエリ、インデックスなど、通常クラスには定義しない項目をクラスに置くことができます。

さまざまなクラス・メンバの定義に使用される言語 (ObjectScript、Python、SQL、その他メンバの実装に使用されるコードを除く) は、クラス定義言語 (CDL) と呼ばれることもあります。

このリファレンスの "クラス定義の構文とキーワード" を参照してください。

関連項目 :

- クラス・プログラミングの基本的な考え方
- クラス定義の基本的なコンテンツ
- クラスの定義
- クラス定義とタイプ
- クラスの制限
- クラス定義

1.2 外部キー定義

外部キーは、参照整合性制約を定義します。外部キー制約を持つテーブルを変更する際に、外部キー制約が確認されます。外部キー定義は、永続クラスに追加できます。他のクラスでは意味がありません。クラスを接続するリレーションシップ・プロパティを定義することで、参照整合性を強制することもできます。

このリファレンスの ″外部キーの構文とキーワード″ を参照してください。

関連項目：

- 外部キーの使用法
- ″テーブル間のリレーションシップ″ の ″外部キーの定義″
- ″スタジオの使用法″ の ″クラスへの SQL トリガと外部キーの追加″

1.3 インデックス定義

インデックスとは永続クラスにより管理される構造のことです。永続クラスはクエリおよびその他の関数の最適化に使用することが意図されています。これらのインデックスは、オブジェクトベースの操作と同様、データベースに対して SQL ベースの INSERT、UPDATE、DELETE 演算が実行された場合にも、自動的に維持されます。SQL クエリ・プロセッサは、SQL クエリを作成、実行するときに、利用可能なインデックスを活用します。インデックス定義は永続クラスに追加できます。他のクラスでは意味がありません。

このリファレンスの ″インデックスの構文とキーワード″ を参照してください。

関連項目：

- インデックスの定義と構築
- ″スタジオの使用法″ の ″クラスへのインデックスの追加″

1.4 メソッド定義

InterSystems IRIS は、2 種類のメソッドとしてインスタンス・メソッドとクラス・メソッドをサポートしています。インスタンス・メソッドは、クラスの特定のインスタンスから呼び出され、そのインスタンスに関連する何らかのアクションを実行します。このタイプのメソッドはオブジェクト・クラスでのみ有用です。クラス・メソッドは、そのクラスのインスタンスがメモリ内にあるかどうかにかかわらず、呼び出すことができます。このタイプのメソッドは、他の言語では静的メソッドと呼ばれています。

このリファレンスの ″メソッドの構文とキーワード″ を参照してください。

関連項目：

- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- ″スタジオの使用法″ の ″クラスへのメソッドの追加″

1.5 パラメータ定義

パラメータ定義では、特定のクラスのすべてのオブジェクトで使用できる定数値を定義します。このクラス・パラメータの値は、クラス定義を作成するときに (またはコンパイル前の任意の時点で) 設定できます。既定では、各パラメータの値は NULL 文字列になりますが、パラメータ定義の一環として NULL 以外の値を指定することもできます。コンパイル時に、パラメータの値はクラスのすべてのインスタンスに対して構築されます。わずかな例外はありますが、この値を実行時に変更することはできません。

このリファレンスの ″パラメータの構文とキーワード″ を参照してください。

関連項目:

- クラス・パラメータの定義と参照
- ″スタジオの使用法″ の ″クラスへのクラス・パラメータの追加″

1.6 投影定義

クラス投影は、クラス・コンパイラの動作を拡張する方法を提供します。投影のメカニズムは、Java 投影で使用されます。これが、投影という用語の語源です。

プロジェクション定義では、クラス定義がコンパイルされたとき、または削除されたときに、クラス・コンパイラが指定された操作を実行するように指示します。プロジェクションは、(%Projection.AbstractProjection クラスから派生した) プロジェクション・クラスの名前を定義します。プロジェクション・クラスは (クラス定義の削除か、クラスの再コンパイルのいずれかによる)、クラスのコンパイル終了時と、クラス定義の削除時に呼び出されるメソッドを実装します。

このリファレンスの ″投影の構文とキーワード″ を参照してください。

関連項目:

- クラス・プロジェクションの定義
- ″スタジオの使用法″ の ″クラスへのプロジェクションの追加″

1.7 プロパティ定義

プロパティには、クラスのインスタンスに関係する情報が含まれます。プロパティ定義はオブジェクト・クラスに追加できます。他のクラスでは意味がありません。

このリファレンスの ″プロパティの構文とキーワード″ を参照してください。

関連項目:

- リテラル・プロパティの定義と使用
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- コレクションを使用した作業

1.8 クエリ定義

クラス・クエリは、クラスが使用できる SQL クエリを定義し、クエリのためのコンテナとして使用するクラスを指定します。必須ではありませんが、クラス・クエリは永続クラスに定義することが普通です。そのクラスの保存済みデータを対象としてクエリを実行するためです。ただし、クラス・クエリは、どのようなクラスにでも定義できます。

このリファレンスの ″クエリの構文とキーワード″ を参照してください。

関連項目:

- クラス・クエリの定義と使用
- ″スタジオの使用法″ の ″クラスへのクエリの追加″

1.9 トリガ定義

トリガは、特定のイベントが InterSystems SQL で発生するときに実行されるコード・セグメントです。InterSystems IRIS は INSERT コマンド、UPDATE コマンド、DELETE コマンドの実行をベースにしたトリガをサポートします。トリガ定義により、指定されたコードは関連するコマンドが実行される直前、または直後に実行されます。各イベントは、実行順序が指定されていれば複数のトリガを持つことができます。トリガ定義は永続クラスに追加できます。他のクラスでは意味がありません。

このリファレンスの ″トリガの構文とキーワード″ を参照してください。

関連項目 :

- トリガの使用法
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- ″スタジオの使用法″ の ″クラスへの SQL トリガと外部キーの追加″

1.10 XData ブロック

XData ブロックは、クラス内に定義されたデータの名前付きユニットです。通常、クラスのメソッドで使用します。これらには、多数の用途があります。

このリファレンスの ″XData ブロックの構文とキーワード″ を参照してください。

関連項目 :

- XData ブロックの定義と使用
- ″スタジオの使用法″ の ″クラスへの XData ブロックの追加″

1.11 ストレージ定義

%Persistent クラスは、データベースでのオブジェクトの保存と取得に使用する高レベルのインタフェースを提供します。オブジェクトの保存とロードの実際の作業は、ストレージ・クラスによって実行されます。このクラスは、ストレージ・インタフェースで使用する一連のキーワード、値、追加パラメータを収めたストレージ定義で指定します。

このリファレンスの ″Storage キーワード″ を参照してください。

関連項目 :

- ストレージ定義とストレージ・クラス
- ストレージ
- ″スタジオの使用法″ の ″クラスへのストレージ定義の追加″

最上位レベル・クラスの構文とキーワード

このリファレンスは、クラス全体に適用する構文とキーワード、またはそのメンバに既定の動作を指定する構文とキーワードについて説明します。後半のリファレンス・セクションでは、特定のクラス・メンバに適用されるキーワードについて説明しています。

クラス定義に関する一般情報へのリンクは、"クラス定義" を参照してください。

特定のクラス・メンバに適用するすべての構造とキーワードは、目次を参照してください。

最上位レベル・クラスの構文

クラス定義の構造を説明します。クラス定義に関する一般情報へのリンクは、"クラス定義" を参照してください。

概要

InterSystems IRIS では、プロパティ、メソッド、パラメータ (他のクラス言語では定数と呼ぶ) など、広く使用するクラス要素をクラスに置くことができます。また、トリガ、クエリ、インデックスなど、クラスでは通常定義されない項目を含めることもできます。

クラス定義ではクラスを定義します。クラス定義は、クラス・メンバ (プロパティやメソッドなど)、キーワードと呼ばれるその他の項目、およびそれぞれの対応する値から成り、これらによってクラスの振る舞いの詳細を指定します。

詳細

クラス定義の構造は以下のとおりです。

```
Import import_ppackage_list
Include include_code
IncludeGenerator include_generator_code

/// description
Class package.shortclassname Extends superclass_list [ class_keyword_list ]
{
Class_members
  [Parameter]
  [Property]
  [Method]
  [Foreign Key]
  [Index]
  [Projection]
  [Query]
  [Storage]
  [Trigger]
  [XData Block]
}
```

以下は、この指定の説明です。

- import_package_list (オプション) は、クラスのインポート元にするパッケージの名前を指定します。これは、コンパイラが短いクラス名を解決する方法に影響を与えます。"パッケージのインポート" を参照してください。

  このオプションは、指定する場合には、単一パッケージの名前、または括弧で囲んだ複数パッケージのコンマ区切りリストのいずれかにします。

  import_package_list が NULL の場合は、`Import` 行をクラス定義の先頭に追加しないでください。

  注釈　　クラスがパッケージをインポートする場合、そのクラスは User パッケージを自動的にインポートしません。

  インポートされたパッケージは、すべてのスーパークラスから継承されます。サブクラスが 1 つまたは複数のインポート・パッケージを指定する場合、それらはスーパークラスによって定義されているインポート・パッケージに追加されます。

- include_code (オプション) は、このクラスをコンパイルするときに使用する InterSystems IRIS インクルード (.inc) ファイルを指定します。

  このオプションは、指定する場合には、単一インクルード・ファイルの名前 (.inc 拡張子は指定しない)、または括弧で囲んだ複数インクルード・ファイルのコンマ区切りリストのいずれかにします。

  include_code が NULL の場合は、クラス定義の先頭の `Include` 行は省略します。

  インクルード・ファイルの概要については、"インターシステムズ・プログラミング・ツールの索引" の "インクルード・ファイル" を参照してください。

インクルード・ファイルは、すべてのスーパークラスから継承されます。サブクラスが 1 つまたは複数のインクルード・ファイルを指定する場合、それらはスーパークラスによって定義されているインクルード・ファイルに追加されます。

- include_generator_code (オプション) は、このクラスのジェネレータ・メソッドをコンパイルするときに使用する InterSystems IRIS インクルード (.inc) ファイルを指定します。ジェネレータ・メソッドの詳細は、"メソッド・ジェネレータとトリガ・ジェネレータの定義" を参照してください。

  一般的なコメントは、前の項目を参照してください。

  include_generator_code が NULL の場合は、`IncludeGenerator` 行をクラス定義の先頭に追加しないでください。

- description (オプション) は、クラス・リファレンスでの表示を意図しています。description は複数行で構成することができます。また、HTML フォーマット・タグや、`<class>`、`<method>` などの追加のタグを組み込むこともできます。制限および詳細は、"クラス・ドキュメントの作成" を参照してください。description は既定ではブランクです。

- package (必須) は、有効なパッケージ名です。これについては、"パッケージ名" を参照してください。

- shortclassname (必須) は、有効なクラス名です。package および shortclassname の両方により、完全なクラス名が構成されます。これは長さ制限があります。"名前付け規約" を参照してください。

- superclass_list (オプション) は、このクラスの継承元のクラスを指定します。このオプションは、指定する場合には、単一クラスの名前 (.cls 拡張子は指定しない)、または括弧で囲んだ複数クラスのコンマ区切りリストのいずれかにします。

  最初のクラスは、プライマリ・スーパークラス と見なされます。後から追加したクラスは、セカンダリ・スーパークラスです。詳細は、"Inheritance" を参照してください。

  superclass_list が NULL の場合、単語 `Extends` をクラス定義から省きます。

- class_keyword_list (オプション) は、キーワードのコンマ区切りリストであり、(ほとんどの場合) このクラス定義のコードをコンパイラが生成する方法に影響を与えます。

  すべてのキーワードは、"最上位レベル・クラスの構文とキーワード" を参照してください。

  このリストを省略する場合は角括弧も省略します。

- Class_members は、クラス・メンバのゼロ以上の定義です。

  概要は、"クラス・メンバの種類" を参照してください。クラス・メンバの詳細は、このリファレンスで以下のセクションを参照してください。

  - 外部キーの構文とキーワード
  - インデックスの構文とキーワード
  - メソッドの構文とキーワード
  - パラメータの構文とキーワード
  - 投影の構文とキーワード
  - プロパティの構文とキーワード
  - クエリの構文とキーワード
  - Storage キーワード
  - トリガの構文とキーワード
  - XData ブロックの構文とキーワード

関連項目

- クラス・プログラミングの基本的な考え方

Abstract (クラス・キーワード)

Abstract クラスであるかどうかを指定します。

使用法

クラスを Abstract としてマークするには、以下の構文を使用します。

```
Class MyApp.MyClass [ Abstract ]
{ //class members }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を配置します。

詳細

オブジェクト・クラスが Abstract の場合、そのクラスのインスタンスは作成できません。

Abstract クラスは、1 つ以上の具象 (非抽象) クラスのスーパークラスとなります。アプリケーションには、すべての Person に共通の属性、メソッド、およびプロパティを含む抽象 Person クラスを含めることができます。非抽象 Employee および Customer クラスは、Person から継承し、それぞれに固有の追加のメソッドとプロパティを含むことができます。

Abstract クラスには、シグニチャのみを持ち、コードのない抽象メソッドを含めることもできます。サブクラスは、これらのメソッドを継承します。開発者は、そのサブクラスに固有のメソッドのコードを提供する必要があります。

サブクラスへの影響

このキーワードは継承されません。

既定値

このキーワードを省略すると、クラスは Abstract ではなくなります。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

ClassType (クラス・キーワード)

このクラスのタイプ (または、振る舞い) を指定します。

使用法

クラスのタイプを指定するには (必要な場合)、以下の構文を使用します。

```
Class MyApp.MyClass [ ClassType = classtype ]
{ //class members }
```

classtype は以下のいずれかになります。

- `datatype` ― このクラスはデータ型クラスであり、リテラル値を表すために使用されます。
- `dynamic` ― このクラスは、%DynamicAbstractObject、%DynamicArray、%DynamicObject などのダイナミック・クラスです。
- `index` ― このクラスは、インデックス・インタフェースを定義する特殊クラスである、インデックス・クラスです。詳細は、クラス・リファレンスの "%Library.FunctionalIndex" を参照してください。
- `persistent` ― このクラスは、データをデータベースに保存することを表します。
- `serial` ― このクラスは、データが他の永続オブジェクトに (シリアル化された状態で) 保存されることを表します。
- `stream` ― このクラスは、ストリーミング・データを表します。
- `view` ― このクラスは、SQL ビューを定義するために使用されます ("ViewQuery" キーワードを参照してください)。
- 空の文字列は、このクラスが特定のタイプを持たないことを表します。Abstract クラスは、一般にはクラス・タイプを指定しません。

このキーワードを指定しない場合、クラス・タイプは、プライマリ・スーパークラスがあればそれから継承されます。

ClassType は、%RegisteredObject、%SerialObject、%Persistent、%DynamicAbstractObject などのシステム・クラス、およびデータ型クラスに指定されるので、それらのクラスをサブクラス化する場合には、一般にこのキーワードを指定する必要はありません。

詳細

このキーワードは、このクラスの用途を指定します。クラス・コンパイラは、ClassType キーワードを使用してクラスをコンパイルする方法を決定します。例えば、ClassType が `persistent` の場合、クラス・コンパイラは、ストレージ・コンパイラを実行してクラスの永続コードを生成します。明示的に定義されていない限り、ClassType の値は既定値であるか、プライマリ・スーパークラスから継承されます。

永続クラスの場合は、明示的な ClassType 文が必要なのは、標準の永続性動作がオーバーライドされるときのみです。クラス定義にこのような文が含まれている場合、その理由は、開発者によってその文が指定されたか、そのクラスが生成されたコードが旧バージョンの InterSystems IRIS を使用して開発されたかのどちらかです。

サブクラスへの影響

このキーワードは、プライマリ・スーパークラスから継承されます。サブクラスは、キーワードの値をオーバーライドできます。

既定値

このキーワードを省略した場合、クラス・タイプは、プライマリ・スーパークラスがあればそれから継承されます。

注釈　　シャード・クラスのクラスタイプは、`persistent` 以外の値を取ることはできません。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

ClientDataType (クラス・キーワード)

このデータ型がクライアント・テクノロジに投影されるときに使用するクライアント・データ型を指定します。データ型クラスのみに適用されます。

使用法

このデータ型がクライアント・テクノロジに投影されるときに使用するクライアント・データ型を指定するには、以下の構文を使用します。

```
Class MyApp.MyString [ ClientDataType = clienttype ]
{ //class members }
```

clienttype は以下のいずれかになります。

- BIGINT
- BINARY
- BINARYSTREAM
- BOOLEAN
- CHARACTERSTREAM
- CURRENCY
- DATE
- DECIMAL
- DOUBLE
- FDATE
- FTIMESTAMP
- HANDLE
- INTEGER
- LIST
- LONGVARCHAR
- NUMERIC
- STATUS
- TIME
- TIMESTAMP
- VARCHAR (既定値)

詳細

このキーワードは、このクラスがクライアント・テクノロジに投影されるときに使用するクライアント・データ型を指定します。すべてのデータ型クラスは、クライアント・データ型を指定する必要があります。

サブクラスへの影響

このキーワードは、プライマリ・スーパークラスから継承されます。サブクラスは、キーワードの値をオーバーライドできます。

既定値

既定のクライアント・データ型は VARCHAR です。

関連項目

- クラス定義
- データ型クラスの定義
- コンパイラ・キーワードの概要

ClientName (クラス・キーワード)

このクラスのクライアント・プロジェクションで使用される既定のクラス名をオーバーライド可能にします。

使用法

クライアントに投影されるときのクラスの既定の名前をオーバーライドするには、以下の構文を使用します。

```
Class MyApp.MyClass [ ClientName = clientclassname ]
{ //class members }
```

clientclassname は、クラス名の代わりにクライアント名として使用される、引用符で囲まれていない文字列です。

詳細

このキーワードにより、(InterSystems IRIS Java バインディングを使用するときなど) クラスをクライアントに投影するときに、クラスに別名を定義できます。

サブクラスへの影響

このキーワードは継承されません。

既定値

このキーワードを省略すると、クライアントでは実際のクラス名が使用されます。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

CompileAfter (クラス・キーワード)

他の (指定した) クラスの後にこのクラスをコンパイルするように指定します。DependsOn とは対照的に、このキーワードは、他のクラスが実行可能である必要はありません。

使用法

クラス・コンパイラが他のクラスの後にこのクラスをコンパイルするように指定するには、以下の構文を使用します。

```
Class MyApp.MyClass [ CompileAfter = classlist ]
{ //class members }
```

classlist は以下のいずれかになります。

- クラス名。以下に例を示します。

  ```
  [ CompileAfter = MyApp.Class1 ]
  ```

- 括弧で囲んだ、クラス名のコンマ区切りリスト。以下に例を示します。

  ```
  [ CompileAfter = (MyApp.Class1,MyApp.Class2,MyApp.Class3) ]
  ```

詳細

このキーワードは、クラス・コンパイラが指定されたクラスをコンパイルした後にこのクラスをコンパイルするように指定します。このキーワードが、このクラスをコンパイルする前に、指定されたクラスが実行可能であるかどうかを確認することはありません。また、CompileAfter キーワードは、System キーワードの共通値を持つクラスのみに影響を与えます。

注釈　他のクラスが実行可能になった後にクラスがコンパイルされるようにするには、代わりに DependsOn キーワードを使用します。例えば、クラス A がクラス B で定義されているクラス・クエリを参照する場合、クラス A がコンパイル可能になるにはまずクラス B が実行可能でなければなりません。この場合、DependsOn キーワードを使用します。

このキーワードは、実行時動作には影響を与えません。

サブクラスへの影響

このキーワードは、すべてのスーパークラスから継承されます。サブクラスでこのキーワードの値が指定されている場合、その値は、サブクラスをコンパイルする前にコンパイルする必要がある追加のクラスを指定します。

既定値

既定では、このキーワードは指定されません。

関連項目

- System キーワード
- DependsOn キーワード
- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

DdlAllowed (クラス・キーワード)

DDL 文をクラス定義の変更または削除に使用できるかどうかを指定します。永続クラスのみに適用されます。

使用法

DDL を使用してクラスを変更できるようにするには、以下の構文を使用します。

```
Class MyApp.Person Extends %Persistent [ DdlAllowed ]
{ //class members }
```

そのように指定しない場合は、このキーワードを省略するか、または以下の構文を使用します。

```
Class MyApp.Person Extends %Persistent [ Not DdlAllowed ]
{ //class members }
```

詳細

このキーワードは、DDL 文 (DROP TABLE、ALTER TABLE、DROP INDEX など) を、クラス定義の変更または削除に使用できるかどうかを指定します。

一般に、SQL ユーザが DDL 文を使用してクラスを変更できるようにすることは望ましくありません。

サブクラスへの影響

このキーワードは継承されません。

既定値

このキーワードを省略すると、DDL 文を使用してクラス定義に影響を与えることはできません。

メモ

DDL CREATE TABLE 文を実行してクラスを作成した場合、そのクラスでは、DdlAllowed キーワードの初期値は True に設定されます。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

DependsOn (クラス・キーワード)

コンパイラが他の (指定した) クラスを実行可能にした後にこのクラスをコンパイルするように指定します。

使用法

他のクラスが実行可能になった後に、クラス・コンパイラがこのクラスをコンパイルするように指定するには、以下の構文を使用します。

```
Class MyApp.MyClass [ DependsOn = classlist ]
{ //class members }
```

classlist は以下のいずれかになります。

- クラス名。以下に例を示します。

  ```
  [ DependsOn = MyApp.Class1 ]
  ```

- 括弧で囲んだ、クラス名のコンマ区切りリスト。以下に例を示します。

  ```
  [ DependsOn = (MyApp.Class1,MyApp.Class2,...) ]
  ```

詳細

このキーワードは、クラス・コンパイラが指定されたクラスを実行可能にした後にこのクラスをコンパイルするように指定します。例えば、クラス A がクラス B で定義されているクラス・クエリを参照する場合、クラス A がコンパイル可能になるにはまずクラス B が実行可能でなければなりません。この場合、クラス A はクラス B に依存しています。

このキーワードは、このクラスのコンパイルがメソッド・ジェネレータ・ロジックで他のクラスを使用する場合に役立ちます。クラスに他のクラスを呼び出す初期値式が含まれている場合にも、このキーワードは役立ちます。

このキーワードは、実行時動作には影響を与えません。

注釈　DependsOn キーワードは、System キーワードの共通値を持つクラスのみに影響を与えます。

また、クラスに `DependsOn=ClassA` がある場合に `CompileAfter=ClassA` も指定すると冗長になります。"CompileAfter" キーワードを参照してください。

サブクラスへの影響

このキーワードは、すべてのスーパークラスから継承されます。サブクラスでキーワードの値が指定されている場合、その値は、サブクラスをコンパイルする前に実行可能にする必要がある追加のクラスを指定します。

既定値

既定では、このキーワードは指定されません。

関連項目

- System キーワード
- CompileAfter キーワード
- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

Deprecated (クラス・キーワード)

このクラスを非推奨として指定します。このキーワードはクラス・コンパイラでは無視され、単にクラスが非推奨であることを人間が読める形で示します。

使用法

クラスを非推奨としてマークするには、以下の構文を使用します。

```
Class MyApp.MyClass [ Deprecated ]
{ //class members }
```

そうしない場合は、このキーワードを省略するか、キーワードの直前に単語 Not を配置します。

関連項目

- クラス定義

Final (クラス・キーワード)

クラスが Final である (サブクラスを持てない) かどうかを指定します。

使用法

クラスを Final として指定するには、以下の構文を使用します。

```
Class MyApp.Exam As %Persistent [ Final ]  { //class members }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 Not を配置します。

詳細

クラスが final である場合、サブクラスを持つことはできません。

また、クラスが Final クラスの場合、クラス・コンパイラは特定のコード生成の最適化を利用できます (Final クラスのインスタンスは、多様型として使用できないため)。

既定値

このキーワードを省略すると、クラス定義は Final ではなくなります。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

GeneratedBy (クラス・キーワード)

別のクラス内のコードで生成されたため編集してはならないクラスであることを示します。

使用法

以下の構文は、このクラスが別のクラスのコードで生成されたことを示します。

```
Class MyApp.MyClass [ GeneratedBy = MyApp.Generator.cls ] { //class members }
```

MyApp.Generator は、完全修飾されたクラス名です。

詳細

このキーワードを指定した場合、編集してはならないことを示すため、スタジオではクラスがグレーの背景色で表示されます。

サブクラスへの影響

このキーワードは継承されません。

既定値

このキーワードを省略すると、スタジオはクラスを通常どおり表示します。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

Hidden (クラス・キーワード)

クラスが Hidden である (クラス・リファレンスにリストされない) かどうかを指定します。

使用法

クラスを Hidden とするには、以下の構文を使用します。

```
Class MyApp.Person [ Hidden ]  { //class members }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を配置します。

詳細

クラスが hidden である場合、それはクラス・リファレンスにも、InterSystems スタジオ・インスペクタの [ワークスペース] ウィンドウにもリストされません。また、Visual Studio Code 用の InterSystems ObjectScript 拡張機能を使用する際に ObjectScript エクスプローラのペインにもリストされません (ただし、**[開く]** ダイアログ・ボックスでクラスの名前を入力すれば、引き続きクラスをスタジオで開くことができます。また、VS Code – ObjectScript を使用している場合にクラスをローカル・ワークスペースに保存済みの場合も、クラスを開くことが可能です)。

サブクラスへの影響

このキーワードは継承されません。

既定値

このキーワードを省略すると、クラスは Hidden ではなくなります。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

Inheritance (クラス・キーワード)

このクラスのスーパークラスの継承順序を指定します。

使用法

このクラスのスーパークラスの継承順序を指定するには、以下の構文を使用します。

```
Class MyApp.MyClass Extends (MySuperClass1, MySuperClass2) [ Inheritance = inheritancedirection ] {
//class members }
```

inheritancedirection は `left` または `right` です。

または、このキーワードを省略します。その場合、InterSystems IRIS は既定の継承の向き (`left`) を使用します。

詳細

Inheritance キーワードで、多重継承によるクラスの継承順序を指定します。inheritancedirection の値に `left` を指定すると左から右への継承、`right` を指定すると右から左への継承になります。

例えば、説明の中のクラス定義で、値が `left` の場合、MySuperClass1 と MySuperClass2 との間でメンバ定義の競合があると MySuperClass1 が優先されることを示します。一方、`right` の場合は、MySuperClass2 が優先されることを示します。

重要　　一番左側にあるスーパークラスは、継承順序にかかわらず、常にプライマリ・スーパークラスになります。

サブクラスへの影響

このキーワードは継承されません。

既定値

このキーワードを省略すると、継承順序は `left` になります。

関連項目

- クラス定義
- クラスの定義
- ″クラスの定義″ の ″多重継承″
- コンパイラ・キーワードの概要

Language (クラス・キーワード)

このクラスのメソッドを実装するのに使用する既定の言語を指定します。

使用法

このクラスにメソッドを実装するために使用する既定の言語を指定するには、以下の構文を使用します。

```
Class MyApp.MyClass [ Language = language ]
{ //class members}
```

language は以下のいずれかになります。

- `objectscript` — ObjectScript (既定)
- `tsql` — Transact-SQL

または、このキーワードを省略します。その場合、InterSystems IRIS は既定の言語 (ObjectScript) を使用します。

詳細

このキーワードは、このクラスのメソッドを実装するのに使用する既定の言語を指定します。個々のメソッドでは、メソッドの Language キーワードを使用して、この値をオーバーライドできます。

サブクラスへの影響

このキーワードは継承されません。

既定値

このキーワードを省略すると、言語は ObjectScript になります。

注釈　クラス・レベルでは `Language = ispl` も `Language = python` も指定できません。これらの値はメソッドにのみ使用できます。

注釈　シャード・クラスのメソッドの既定言語は、ObjectScript 以外の言語にはできません。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

LegacyInstanceContext (クラス・キーワード)

廃止された %this 変数を、このクラス内のインスタンス・メソッドで使用できるかどうかを指定します。

使用法

クラス内のインスタンス・メソッドで %this を使用できるようにするには、以下の構文を使用します。

```
Class MyApp.MyClass [ LegacyInstanceContext ]  { //class members }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を配置します。

詳細

このキーワードが true の場合、現在は廃止された ($this に置き換えられている) %this 変数を、このクラス内のインスタンス・メソッドで使用できます。このキーワードが false の場合、インスタンス・メソッドでは %this を参照できません。

サブクラスへの影響

このキーワードは継承されません。

既定値

このキーワードを省略した場合、インスタンス・メソッドでは %this を参照できません。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

NoExtent (クラス・キーワード)

コンパイラに、オブジェクトをディスクからロードし、ディスクに保存するためのストレージ定義とメソッドを生成させないようにするかどうかを指定します。

使用法

コンパイラに、オブジェクトをディスクからロードし、ディスクに保存するためのストレージ定義とメソッドを生成させないようにするには、次の構文を使用します。

```
Class MyApp.MyClass [ NoExtent ]  { //class members }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 Not を配置します。

詳細

このキーワードが true の場合、コンパイラは、オブジェクトをディスクからロードし、ディスクに保存するためのストレージ定義とメソッドを生成しません。そのようなクラスのインスタンスは保存できません。多くの場合、そのようなクラスは %Library.Persistent から継承された標準永続インタフェースを拡張またはオーバーライドします。

サブクラスへの影響

このキーワードは継承されません。

既定値

このキーワードを省略すると、クラス・コンパイラは、オブジェクトをディスクからロードし、ディスクに保存するためのストレージ定義とメソッドを生成しません (該当する場合)。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

OdbcType (クラス・キーワード)

このデータ型が ODBC または JDBC によって表示されるときに使用するタイプを指定します。すべてのデータ型クラスは、ODBC タイプを指定する必要があります。このキーワードは、データ型クラスのみに適用されます。

使用法

このデータ型が ODBC または JDBC によって投影されるときに使用するタイプを指定するには、以下の構文を使用します。

```
Class MyApp.MyString [ ClassType = DataType, OdbcType = odbctype ]  { //class members }
```

odbctype は以下のいずれかになります。

- BIGINT
- BIT
- DATE
- DOUBLE
- INTEGER
- LONGVARBINARY
- LONGVARCHAR
- NUMERIC
- RESULTSET
- SMALLINT
- STRUCT
- TIME
- TIMESTAMP
- TINYINT
- VARBINARY
- VARCHAR (既定値)

詳細

このキーワードは、ODBC または JDBC によって表示されるときに使用するタイプを指定します。

すべてのデータ型クラスは、ODBC タイプを指定する必要があります。

サブクラスへの影響

このキーワードは、プライマリ・スーパークラスから継承されます。サブクラスは、キーワードの値をオーバーライドできます。

既定値

このキーワードを省略すると、ODBC タイプは VARCHAR になります。

関連項目

- クラス定義
- データ型クラスの定義
- コンパイラ・キーワードの概要

Owner (クラス・キーワード)

このクラスの所有者と、それに対応するテーブルを指定します。永続クラスのみに適用されます。

使用法

このクラスの所有者と、それに対応するテーブルを指定するには、以下の構文を使用します。

```
Class MyApp.Person Extends %Persistent [ Owner = "username" ]  { //class members }
```

username は InterSystems IRIS ユーザ名です。

詳細

このキーワードは、クラスの所有者と、それに対応するテーブルを指定します。

サブクラスへの影響

このキーワードは、プライマリ・スーパークラスから継承されます。サブクラスは、キーワードの値をオーバーライドできます。

既定値

このキーワードを省略すると、このクラスとそのテーブルは _SYSTEM ユーザによって所有されます。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

ProcedureBlock (クラス・キーワード)

このクラスの各 ObjectScript メソッドが、規定でプロシージャ・ブロックであるかどうかを指定します。

使用法

このクラスの ObjectScript メソッドが既定でプロシージャ・ブロックになるようにするには、このキーワードを省略するか、または以下の構文を使用します。

```
Class MyApp.MyClass [ ProcedureBlock ]  { //class members }
```

あるいは、以下の構文を使用します。

```
Class MyApp.MyClass [ Not ProcedureBlock ] { //class members }
```

詳細

このキーワードは、このクラスの ObjectScript メソッドが、既定でプロシージャ・ブロックであるかどうかを指定します。これは、メソッドで ProcedureBlock キーワードを設定することによって、メソッドごとにオーバーライドできます。

このキーワードは、他の言語で記述されたメソッドでは無視されます。

ObjectScript では、メソッドはプロシージャ・ブロックとして実装することも、しないこともできます。プロシージャ・ブロックは、変数の範囲設定を強制します。メソッドからは、呼び出し元で定義されている変数は見えません。新しいアプリケーションはプロシージャ・ブロックを使用しますが、従来のアプリケーションとの互換性を保つために、非プロシージャ・ブロックも存在します。

サブクラスへの影響

このキーワードは継承されません。

既定値

このキーワードを省略すると、このクラス内の各 ObjectScript メソッドは (個々のメソッドに対して指定をオーバーライドしていない限り) プロシージャ・ブロックになります。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

PropertyClass (クラス・キーワード)

このクラスにプロパティ・パラメータを追加します。

使用法

このクラスにプロパティ・パラメータを追加するには、以下の構文を使用します。

```
Class PropClass.MyClass Extends %RegisteredObject [ PropertyClass = PropClass.MyPropertyClass ] {
//class members }
```

propertyclasslist は以下のいずれかになります。

- 完全なクラス名 (すべてのパッケージを含む)以下に例を示します。

  ```
  [ PropertyClass = PropClass.MyPropertyClass ]
  ```

- 括弧で囲んだ、クラス名のコンマ区切りリスト。

詳細

カスタム・プロパティ・パラメータを追加する必要がある場合は、以下の手順に従います。

1. 1 つまたは複数のクラス・パラメータを定義するクラスを定義してコンパイルします。以下に例を示します。

   ```
   Class PropClass.MyPropertyClass
   {
   Parameter MYPARM As %String = "XYZ";
   }
   ```

   これらのクラス・パラメータは、次の手順ではプロパティ・パラメータになります。

2. プロパティを定義するクラスでは、PropertyClass キーワードを指定します。

サブクラスへの影響

サブクラスは、このキーワードによって追加されたカスタム動作を継承します。サブクラスでキーワードの値が指定されている場合、その値は、このクラスのプロパティのパラメータを指定する追加のクラスを指定します。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

ServerOnly (クラス・キーワード)

このクラスを Java クライアントに投影するかどうかを指定します。

使用法

クラスを Java クライアントに投影する既定の方法をオーバーライドするには、以下の構文を使用します。

```
Class Sample.NewClass1  [ ServerOnly = serveronlyvalue ] { //class members }
```

serveronlyvalue は以下のいずれかになります。

- `0` は、このクラスを投影できることを表します。
- `1` は、このクラスを投影しないことを表します。

詳細

このキーワードが 1 の場合、クラスは Java クライアントに投影されません。このキーワードが 0 の場合、クラスは投影されます。

サブクラスへの影響

このキーワードは継承されません。

既定値

このキーワードを省略すると、このクラスはスタブでない場合に投影されます (スタブである場合は投影されません)。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

Sharded (クラス・キーワード)

このクラスをシャーディングするかどうかを指定します。シャード・クラスタを含む環境内の永続クラスにのみ適用します。

使用法

クラスをシャーディング対象として定義するには、以下の構文を使用します。

```
Class MyApp.MyClass Extends %Persistent [ Sharded = 1 ]
{ //class members }
```

これ以外の場合は、このキーワードを省略します。

詳細

シャーディングはデータ・ストレージを水平方向に拡張するメカニズムです。クラスがシャーディングされる場合、そのクラスのインスタンスは、シャード・クラスタ内で定義されているすべてのデータ・ノードに分散されます。

シャード環境があり、クラスをシャーディングしないと定義する場合、そのクラスのインスタンスは最初のデータ・ノードにのみ格納されます。ただし、そのデータはすべてのノードに表示できます。

サブクラスへの影響

このキーワードは継承されます。

既定値

このキーワードを省略すると、クラスはシャーディングされません。

関連項目

- ″スケーラビリティ・ガイド″ の ″シャーディングによるデータ量に応じた水平方向の拡張″
- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

SoapBindingStyle (クラス・キーワード)

このクラスで定義されている Web メソッドが使用するバインディング・スタイルまたは SOAP 呼び出し機能を指定します。Web サービスまたは Web クライアントとして定義されているクラス内でのみ適用されます。

使用法

このクラスで定義されているいずれかの Web メソッドにより使用されるバインディング・スタイルを指定するには、以下の構文を使用します。

```
Class MyApp.MyClass [ SoapBindingStyle = soapbindingstyle ]  { //class members }
```

soapbindingstyle は以下のいずれかになります。

- `document` (既定値) ― このクラスの Web メソッドは、既定でドキュメント・スタイルのバインディングを使用します。

  このバインディング・スタイルにより、SOAP メッセージはドキュメントとしてフォーマットされ、通常はパートを 1 つのみ持ちます。

  SOAP メッセージでは、通常、`<Body>` 要素に 1 つの子要素が含まれます。`<Body>` 要素のそれぞれの子は、メッセージ・パートに対応します。

- `rpc` ― このクラスの Web メソッドは、RPC (リモート・プロシージャ呼び出し) スタイルのバインディングを既定で使用します。

  このバインディング・スタイルにより、SOAP メッセージは複数のパートを持つメッセージとしてフォーマットされます。

  SOAP メッセージでは、`<Body>` 要素に 1 つの子要素が含まれ、その名前は対応する処理名から取得されます。この要素は生成されたラッパ要素であり、これにはメソッドの引数リストの引数ごとに 1 つの子要素が含まれます。

SoapBindingStyle が `document` であり ARGUMENTSTYLE が `message` である場合、メッセージ・スタイルは RPC に非常によく似たものになります。これについては、"Web サービスおよび Web クライアントの作成" を参照してください。

重要　手動で作成した Web サービスでは、通常はこのキーワードの既定値が最適な値となります。SOAP ウィザードで WSDL から Web クライアントまたは Web サービスを生成すると、このキーワードはその WSDL に最適に設定されます。この値を変更すると、Web クライアントまたは Web サービスは機能しなくなる場合があります。

詳細

このキーワードにより、このクラスで定義された任意の Web メソッドが使用する既定のバインディング・スタイルを指定できます。これは、SOAP 本文の形式に影響します (SOAP ヘッダの形式には影響しません)。

SoapBindingStyle メソッド・キーワードまたは SoapBindingStyle クエリ・キーワードを使用して、個々のメソッドのバインディング・スタイルをオーバーライドできます。

サブクラスへの影響

このキーワードは継承されません。

既定値

既定値は `document` です。SOAP 標準 v1.1 の Chapter 7 Using SOAP for RPC では、Web メソッドで RTC スタイルのバインディングを使用するように規定しています。ただし、.NET を含むほとんどの SOAP クライアントでは、ドキュメント・スタイルのバインディングを使用しています。

WSDL との関係

SoapBindingStyle クラス・キーワードにより、WSDL の <binding>セクション内の <soap:binding> 要素の style 属性の値を指定します。例えば、SoapBindingStyle が document の場合、WSDL は次のようになります。

```
...
<binding ...>
    <soap:binding ... style="document"/>
    <operation ...>
        <soap:operation ... style="document"/>
...
```

ここで示すように、SoapBindingStyle クラス・キーワードでは、WSDL の <binding> セクション内の <soap:operation> 要素の style 属性の既定値も指定されます。この属性は、SoapBindingStyle メソッド・キーワードで詳細に制御できます。

一方、SoapBindingStyle が rpc の場合は、WSDL は次のようになります。

```
...
<binding ...>
    <soap:binding ... style="rpc"/>
    <operation ...>
        <soap:operation ... style="rpc"/>
...
```

バインディング・スタイルは、<message> 要素にも次のように影響します。

- バインディング・スタイルが document の場合、既定ではメッセージはパートを 1 つのみ持ちます。以下に例を示します。

  ```
  <message name="AddSoapIn">
      <part name="parameters" .../>
  </message>
  ```

  ARGUMENTSTYLE パラメータが message の場合、メッセージは複数のパートを持つことができます。以下に例を示します。

  ```
  <message name="AddSoapIn">
     <part name="a" .../>
     <part name="b" .../>
  </message>
  ```

- バインディング・スタイルが rpc の場合、メッセージは複数のパートを持つことができます。以下に例を示します。

  ```
  <message name="AddSoapIn">
      <part name="a" .../>
      <part name="b" .../>
  </message>
  ```

SOAP メッセージへの影響

SOAP メッセージへの主な影響は、SOAP 本文に複数のサブ要素を含められるようにするかどうかを制御することです。

以下に、RPC スタイルのバインディングおよびエンコードスタイルのメッセージを使用する Web メソッドに対して考えられる要求メッセージの本文の例を示します。

XML

```
<SOAP-ENV:Body SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
   <types:Add>
      <a href="#id1" /><b href="#id2" />
   </types:Add>
   <types:ComplexNumber id="id1" xsi:type="types:ComplexNumber">
      <Real xsi:type="s:double">10</Real>
      <Imaginary xsi:type="s:double">5</Imaginary>
   </types:ComplexNumber>
   <types:ComplexNumber id="id2" xsi:type="types:ComplexNumber">
      <Real xsi:type="s:double">17</Real>
      <Imaginary xsi:type="s:double">2</Imaginary>
   </types:ComplexNumber>
</SOAP-ENV:Body>
```

一方、リテラル・バインディングおよびエンコードスタイルのメッセージを使用する Web メソッドに対して考えられる要求メッセージの本文の例を以下に示します。

XML

```
<SOAP-ENV:Body>
   <tns:Add>
      <tns:a xsi:type="tns:ComplexNumber">
         <Real xsi:type="s:double">10</Real>
         <Imaginary xsi:type="s:double">5</Imaginary>
      </tns:a>
      <tns:b xsi:type="tns:ComplexNumber">
         <Real xsi:type="s:double">17</Real>
         <Imaginary xsi:type="s:double">2</Imaginary>
      </tns:b>
      </tns:Add>
</SOAP-ENV:Body>
```

この場合、SOAP 本文はサブ要素を 1 つ持ちます。

%XML.DataSet での使用

タイプ %XML.DataSet のオブジェクトの場合、以下のテーブルに示すように、SoapBindingStyle キーワードと SoapBodyUse キーワードのすべての組み合わせがサポートされるわけではありません。

	SoapBodyUse=literal (既定)	SoapBodyUse=encoded
SoapBindingStyle=document (既定)	サポートされる	サポートされない
SoapBindingStyle=rpc	サポートされる	サポートされる

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要
- Web サービスおよび Web クライアントの作成

SoapBodyUse (クラス・キーワード)

このクラスで定義される Web メソッドのエンコードを指定します。このキーワードは、Web サービス・クラスおよび Web クライアント・クラスにのみ適用されます。

使用法

このクラスの Web メソッドの入力および出力により使用されるエンコードを指定するには、以下の構文を使用します。

```
Class MyApp.MyClass [ SoapBodyUse = soapbodyuse ]  { //class members }
```

soapbodyuse は以下のいずれかになります。

- `literal` (既定値) ー このクラスの Web メソッドは、リテラル・データを既定で使用します。つまり、SOAP メッセージの `<Body>` の中の XML は、WSDL で提供されるスキーマと一致します。
- `encoded` ー このクラスの Web メソッドは、SOAP でエンコードされたデータを既定で使用します。つまり、SOAP メッセージの `<Body>` の中の XML では、以下の仕様に従い、使用される SOAP のバージョンに応じて SOAP エンコードを使用します。
  - SOAP 1.1 (https://www.w3.org/TR/2000/NOTE-SOAP-20000508/)
  - SOAP 1.2 (https://www.w3.org/TR/soap12-part2/)

重要　手動で作成した Web サービスでは、通常はこのキーワードの既定値が最適な値となります。SOAP ウィザードで WSDL から Web クライアントまたは Web サービスを生成すると、このキーワードはその WSDL に最適に設定されます。この値を変更すると、Web クライアントまたは Web サービスは機能しなくなる場合があります。

詳細

このキーワードにより、このクラスで定義された任意の Web メソッドが使用する既定のエンコードを指定します。これは、このトピックのサブセクションでも説明しますが、このクラスの ELEMENTQUALIFIED および XMLELEMENT パラメータの既定値も制御します。

SoapBodyUse メソッド・キーワードまたは SoapBodyUse クエリ・キーワードを使用して、個々のメソッドのキーワードをオーバーライドできます。

サブクラスへの影響

このキーワードは継承されません。

既定値

既定値は `literal` です。SOAP 標準 v1.1 (Chapter 5) では、Web メソッドで SOAP エンコードを使用するように規定しています。ただし、.NET を含むほとんどの SOAP では、リテラル・スタイルを使用しています。

WSDL との関係

SoapBodyUse キーワードにより、WSDL の <binding> セクション内の <soap:body> 要素の use 属性の値を指定します。例えば、SoapBodyUse が literal の場合、WSDL は次のようになります。

```
...
<binding name="MyServiceNameSoap"
 ...
    <soap:binding ...
    <operation name="Add">
        <soap:operation ...>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>
...
```

一方、SoapBodyUse が encoded の場合は、WSDL は次のようになります。

```
...
<binding name="MyServiceNameSoap" ...
    <soap:binding ...
    <operation name="Add">
        <soap:operation .../>
        <input>
            <soap:body use="encoded" namespace="http://www.mynamespace.org"
                       encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </input>
        <output>
            <soap:body use="encoded" namespace="http://www.mynamespace.org"
                       encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </output>

    </operation>
</binding>
...
```

SOAP 1.2 では、encodingStyle 属性は次のようになります。

```
encodingStyle="http://www.w3.org/2003/05/soap-encoding"
```

SoapBodyUse キーワードでは、各 Web メソッドに対する <message> 要素の <part> 要素の内容も決定します。

- SoapBodyUse が literal の場合、各 <part> 要素には、element 属性が含まれます。以下に例を示します。

  ```
  <part name="parameters" element="s0:Add"/>
  ```

  別の例を示します。

  ```
  <part name="b" element="s0:b"/>
  ```

- SoapBodyUse が encoded の場合、各 <part> 要素には、element 属性でなく、type 属性が含まれます。以下に例を示します。

  ```
  <part name="a" type="s0:ComplexNumber"/>
  ```

SoapBodyUse は、ELEMENTQUALIFIED および XMLELEMENT パラメータの既定値も制御します。これも、WSDL に影響します。

SOAP メッセージへの影響

ドキュメント・スタイルのメッセージを使用する Web メソッドでは、Web サービスは次のような応答メッセージを送信します。

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
                   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
                   xmlns:s='http://www.w3.org/2001/XMLSchema'>
   <SOAP-ENV:Body>
      <AddResponse ...>
...
```

一方、エンコードスタイルのメッセージを使用する Web サービスでは、応答メッセージは次のようになります。

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
                   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
                   xmlns:s='http://www.w3.org/2001/XMLSchema'
                   xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'
                   xmlns:tns='http://www.mynamespace.org'
                   xmlns:types='http://www.mynamespace.org'>
   <SOAP-ENV:Body SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
      <types:AddResponse>
...
```

Web サービスまたは Web クライアントのパラメータの既定への影響

ELEMENTQUALIFIED パラメータの既定値は、SoapBodyUse キーワードによって異なります。

SoapBodyUse の値	ELEMENTQUALIFIED の既定値	メモ
literal	1	elementFormDefault="qualified"
encoded	0	elementFormDefault="unqualified"

XMLELEMENT パラメータの既定値も、SoapBodyUse キーワードによって異なります。

SoapBodyUse の値	XMLELEMENT の既定値	メモ
literal	1	メッセージ・パートには element 属性があります。
encoded	0	メッセージ・パートには type 属性があります。

ELEMENTQUALIFIED パラメータと XMLELEMENT パラメータの詳細は、"オブジェクトの XML への投影" を参照してください。

%XML.DataSet での使用

タイプ %XML.DataSet のオブジェクトの場合、SoapBindingStyle キーワードと SoapBodyUse キーワードのすべての組み合わせがサポートされるわけではありません。詳細は、SoapBindingStyle クラス・キーワードのエントリを参照してください。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要
- Web サービスおよび Web クライアントの作成

SqlCategory (クラス・キーワード)

SQL での計算に使用する型を指定します。データ型クラスのみに適用されます。

使用法

SQL での計算に使用する型を指定するには、以下の構文を使用します。

```
Class MyApp.MyString [ ClassType = DataType, SQLCategory = STRING ]  { //class members }
```

sqlcategory は以下のいずれかになります。

- DATE
- DOUBLE
- FMDATE
- FMTIMESTAMP
- INTEGER
- MVDATE
- NAME
- NUMERIC
- STRING (既定値)
- TIME
- TIMESTAMP

詳細

このキーワードは、SQL 計算でこのクラスに使用する型を指定します。

すべてのデータ型クラスは、SQL カテゴリを指定する必要があります。

新規のデータ型クラスを生成する場合、生成しようとしているデータ型に最適な SQL カテゴリ値を使用してください。既存のデータ型クラスをサブクラスにして、その SQL カテゴリを継承する方法もあります。

サブクラスへの影響

このキーワードは、プライマリ・スーパークラスから継承されます。サブクラスは、キーワードの値をオーバーライドできます。

既定値

既定の SQL カテゴリは STRING です。

関連項目

- クラス定義
- データ型クラスの定義
- コンパイラ・キーワードの概要

SqlRowIdName (クラス・キーワード)

このクラスの ID 列の、既定の SQL フィールド名をオーバーライドします。永続クラスのみに適用されます。

使用法

このクラスの ID 列の、既定の SQL フィールド名をオーバーライドするには、以下の構文を使用します。

```
Class MyApp.MyClass [ SqlRowIdName = MyId ]  { //class members }
```

MyId は SQL 識別子です。

詳細

このキーワードは、ID 列に使用される既定の SQL フィールド名をオーバーライドします。

永続クラスを SQL テーブルに投影するとき、各オブジェクトのオブジェクト識別値は SQL 列、つまり行 ID 列として投影されます。既定の行 ID 列は **ID** と呼ばれます。クラスに、他に **ID** という名前のフィールドがある場合は、**ID1** などの名前を使用します。SqlRowIdName キーワードにより、行 ID 列の名前を直接設定できます。

サブクラスへの影響

このキーワードは、プライマリ・スーパークラスから継承されます。サブクラスは、キーワードの値をオーバーライドできます。

既定値

このキーワードを省略すると、このクラスの ID 列の SQL フィールド名は `ID` になります。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

SqlRowIdPrivate (クラス・キーワード)

ODBC および JDBC に投影されるときに、このクラスの ID 列を非表示フィールドにするかどうかを指定します。永続クラスのみに適用されます。

使用法

テーブルを ODBC および JDBC に投影するときに ID 列を非表示にするには、以下の構文を使用します。

```
Class MyApp.MyClass [ SqlRowIdPrivate ]  { //class members }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を配置します。

詳細

このキーワードが true の場合、テーブルが ODBC および JDBC に投影されるときに、ID 列は非表示フィールドになります。

永続クラスを SQL テーブルに投影するとき、各オブジェクトのオブジェクト識別値は SQL 列、つまり行 ID 列として投影されます。SqlRowIdPrivate キーワードにより、行 ID 列を ODBC ベースまたは JDBC ベースのクエリで非表示にするかどうかを指定できます。行 ID 列が非表示の場合は、以下のようになります。

- さまざまなカタログ・クエリにより列としてレポートされません。
- `SELECT *` クエリに含められません。

クエリが SELECT 節内で明示的に列をリストする場合、ODBC または JDBC クライアントはこの列を選択できます。(定義上は、UPDATE 文または INSERT 文の行 ID 列は、行 ID の値を変更したり直接設定したりできないため、使用できません。)

通常、このキーワードは、従来のリレーショナル・データを使用し、レポート・ツールによって行 ID 列を表示しないようにするために使用します。

サブクラスへの影響

このキーワードは継承されません。

既定値

このキーワードを省略すると、テーブルが ODBC および JDBC に投影されるときに、ID 列は通常どおり表示されます。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

SqlTableName (クラス・キーワード)

このクラスを投影する SQL テーブルの名前を指定します。永続クラスのみに適用されます。

使用法

このクラスを投影する SQL テーブルの既定の名前をオーバーライドするには、以下の構文を使用します。

```
Class MyApp.Person Extends %Persistent [ SqlTableName = DBTable ]  { //class members }
```

DBTable は有効な SQL 識別子です。

スキーマまたはパッケージの名前は、SqlTableName で指定されたテーブルの名前の先頭に付加されたままであることに注意してください。上の例で、テーブルは MyApp.DBTable として識別されます。

詳細

このキーワードは、このクラスを投影する SQL テーブルの名前を指定します。既定では、SQL テーブル名はクラス名と同じです。

通常、クラス名が SQL の予約語である場合 (珍しいことではありません) や、SQL テーブルにクラス名としてサポートされていない文字を使用する場合 (_ など) に、このキーワードを使用します。

サブクラスへの影響

このキーワードは継承されません。

既定値

このキーワードを省略すると、クラス名が SQL テーブル名として使用されます。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

StorageStrategy (クラス・キーワード)

このクラスの永続性を制御するストレージ定義を指定します。永続クラスおよびシリアル・クラスのみに適用されます。

使用法

クラスが使用するストレージ定義を指定するには、以下のような構文を使用します。

```
Class MyApp.MyClass Extends %Persistent [ StorageStrategy = MyStorage ]
{ //class members }
```

MyStorage は、このクラス内のストレージ定義の名前です。

詳細

このキーワードは、このクラスが使用するストレージ構造を定義するために使用するストレージ定義を指定します。

通常は、このキーワードやストレージ構造について注意を払う必要はありません。クラス・コンパイラは自動的に Default という名前のストレージ構造を定義し、(必要に応じて新規のフィールドを追加して) これを維持します。1 つのクラスに対し、複数のストレージ定義を生成できます。この場合、このキーワードはクラス・コンパイラがどのストレージ定義を使用するべきかを指定します。

サブクラスへの影響

このキーワードは、プライマリ・スーパークラスから継承されます。サブクラスは、キーワードの値をオーバーライドできます。

既定値

このキーワードを省略すると、このクラスの永続性は、`Default` という既定のストレージ定義により定義されます。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

System (クラス・キーワード)

このクラスのコンパイル順序に影響を与えます。

使用法

クラスのコンパイル順序に影響を与えるには、以下のような構文を使用します。

```
Class MyApp.Person Extends %Persistent [ System = n ]
{ //class members }
```

n は 0 ～ 4 の整数であり、各クラスは、割り当てられた正の値が小さい順にコンパイルされます。0 の値が割り当てられたクラスは最後にコンパイルされます。

詳細

このキーワードによってクラスのグループが形成され、各グループには異なる値と優先度が関連付けられ、完全クラス・コンパイル・プロセスは各優先度レベルについて実行されてから、次の優先度レベルに進みます。優先度が高い順に、以下のレベルがあります。

- 1
- 2
- 3
- 4
- 0 (既定値)

クラス・コンパイルは次の 2 つの手順から成ります。

1. グローバルを解決します。
2. ルーチンをコンパイルします。

同じ System キーワード値を持つすべてのクラスでは、それぞれのグローバルが解決されてからルーチンがコンパイルされます。それぞれのレベルが異なるクラスでは、優先度の高いクラスでグローバル解決とルーチン・コンパイルの両方が完了してから、優先度の低いクラスでグローバルが解決されます。

CompileAfter キーワードと DependsOn キーワードは、共通の System 値を持つクラス内で機能して、グローバル解決の順序を決定します。共通の System 値を持つすべてのクラスでグローバルが解決された後に、これらすべてのクラスでルーチンがコンパイルされます。

このため、クラス B がクラス A のメソッドをクラス B のメソッド・ジェネレータ内で (つまり B のコンパイル時に) 実行する必要がある場合、A の優先度は B の優先度より高い必要があります。これは、A の System キーワードの値は、B の System キーワード値より小さい 0 以外の整数である必要があるということです。この動作を実現するうえで、CompileAfter や DependsOn は無関係です。

サブクラスへの影響

このキーワードは継承されません。

既定値

既定値は 0 (ゼロ) です。

関連項目

- クラス定義

- クラスの定義
- コンパイラ・キーワードの概要

ViewQuery (クラス・キーワード)

このクラスに対する SQL クエリを指定します。ビュー定義クラスのみに適用されます。

使用法

このクラスの SQL クエリを指定するには、以下の構文を使用します。

```
ViewQuery = { statement }
```

ここで statement は、中括弧で囲まれた SQL SELECT 文です。

詳細

SQL ビューを (DDL CREATE VIEW 文または管理ポータルを使用して) 定義する際、システムはビュー定義を保持するために、自動的にクラス定義を生成します。このクラス定義では、ClassType は `view` であり、ViewQuery はビューの基になる SQL 文と等しくなります。

この機能は内部的なものであり、ユーザはビュー・クラスを生成したり、ViewQuery キーワードを変更したりすることはできません。その代わり、ビューの管理には通常の機能 (DDL または管理ポータル) を使用してください。

すべての非ビュー・クラスで、このキーワードは無視されます。

既定値

既定値は、空の文字列です。

関連項目

- クラス定義
- クラスの定義
- コンパイラ・キーワードの概要

外部キーの構文とキーワード

このリファレンスでは、永続クラスで定義できる外部キーに適用する構文とキーワードを説明します。キーワード（クラス属性とも呼ばれる）は、一般にコンパイラに影響を与えます。

外部キーの定義に関する一般情報へのリンクは、"外部キー定義" を参照してください。

特定のクラス・メンバに適用するすべての構造とキーワードは、目次を参照してください。

クラス定義での外部キーの構文

外部キー定義の構造を説明します。外部キーの定義に関する一般情報へのリンクは、"外部キー定義" を参照してください。

概要

外部キーは、参照整合性制約を定義します。外部キー制約を持つテーブルを変更する際に、外部キー制約が確認されます。

外部キー定義は、永続クラスに追加できます。他のクラスでは意味がありません。

クラスを接続するリレーションシップ・プロパティを定義することで、参照整合性を強制することもできます。

詳細

外部キー定義の構造は以下のとおりです。

```
/// description
ForeignKey name(key_props) References referenced_class(ref_index) [ keyword_list ];
```

以下は、この指定の説明です。

- description (オプション) は、クラス・リファレンスでの表示を意図しています。description は既定ではブランクです。"クラス・ドキュメントの作成" を参照してください。
- name (必須) は外部キーの名前です。これは、有効なクラス・メンバ名で、かつ他のクラス・メンバ名と重複しないものである必要があります。
- key_props (必須) は、この外部キーにより制約されるプロパティ (複数可) を指定します。特にこのプロパティは、外部テーブル内の参照される値と一致している必要があります。

  これはプロパティ名のコンマ区切りリストです。

  それらのプロパティは、外部キーを定義する同じクラスに存在する必要があります。
- referenced_class (必須) は、外部テーブルを指定します (つまり、外部キーが指すクラス)。
- ref_index (オプション) は、referenced_class 内の一意のインデックス名を指定します。

  ref_index を省略すると、システムは referenced_class 内の IDKEY インデックスを使用します。
- keyword_list (オプション) は、さらに外部キーを定義するキーワードのコンマ区切りリストです。

  すべてのキーワードは、"外部キーの構文とキーワード" を参照してください。

  このリストを省略する場合は角括弧も省略します。

例

```
ForeignKey EmpKey(EmpId) References MyApp.Employee(EmpIDX) [ OnDelete = cascade ];
```

この例では、`EmpIDX` がクラス `MyApp.Employee` 内の一意のインデックス名です。

関連項目

- 外部キーの使用法
- クラスの制限

Internal (外部キーのキーワード)

この外部キーが Internal (クラス・ドキュメントに表示されない) であるかどうかを指定します。

使用法

この外部キー定義を Internal としてマークするには、以下の構文を使用します。

```
ForeignKey keyname(key_props) References pkg.class(ref_index) [ Internal ];
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

Internal クラスのメンバは、クラス・ドキュメントには表示されません。このキーワードは、ユーザに対してクラスは表示するが、そのすべてのメンバを表示する必要がない場合に役立ちます。

既定値

このキーワードを省略すると、この外部キー定義はクラス・ドキュメントに表示されます。

関連項目

- 外部キー定義
- 外部キーの使用法
- コンパイラ・キーワードの概要

NoCheck (外部キーのキーワード)

InterSystems IRIS がこの外部キー制約をチェックするかどうかを指定します。

使用法

InterSystems IRIS がこの外部キーにより定義されている制約をチェックしないようにするには、以下の構文を使用します。

```
ForeignKey keyname(key_props) References pkg.class(ref_index) [ NoCheck ];
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 Not を指定します。

詳細

NoCheck キーワードは、外部キー制約のチェックを抑制します (つまり、外部キー制約のチェックが行われないように指定します)。

既定値

このキーワードを省略すると、InterSystems IRIS は外部キー制約をチェックします。

関連項目

- 外部キー定義
- 外部キーの使用法
- コンパイラ・キーワードの概要

OnDelete (外部キーのキーワード)

外部テーブルで削除されたレコードが現在のテーブルのレコードで参照されている場合に、現在のテーブルで外部キーが実行するアクションを指定します。

使用法

外部テーブルの参照されているレコードが削除されたときに、現在のテーブルで実行するアクションを指定するには、以下の構文を使用します。

```
ForeignKey keyname(key_props) References pkg.class(ref_index) [ OnDelete = ondelete ];
```

ondelete は以下のいずれかになります。

- `noaction` (既定値) – 外部テーブルの参照されているレコードの削除を試みた場合、失敗します。
- `cascade` – 外部テーブルの参照されているレコードを削除した場合、そのテーブル内の参照元レコードも削除されます。
- `setdefault` – 外部テーブルの参照されているレコードを削除した場合、参照元レコードの外部テーブルへの参照はその既定値に設定されます。
- `setnull` – 外部テーブルの参照されているレコードを削除した場合、参照元レコードの外部テーブルへの参照は NULL に設定されます。

説明

外部テーブル内の行を削除する場合、その外部テーブルに外部キー制約が設定されているすべての参照元テーブルで、削除する行を参照している行があるかどうかがチェックされます。このような参照が見つかった場合、OnDelete アクションは有効になります。

既定値

既定値は `noaction` です。

関連項目

- 外部キー定義
- 外部キーの使用法
- コンパイラ・キーワードの概要
- ″InterSystems SQL リファレンス″ の ″外部キーの定義″

OnUpdate (外部キーのキーワード)

外部テーブルのレコードのキー値が更新され、そのレコードが現在のテーブルのレコードで参照されている場合に、現在のテーブルでこの外部キーが実行するアクションを指定します。

使用法

外部テーブルの参照されているレコードのキー値が更新されたときに、現在のテーブルで実行するアクションを指定するには、以下の構文を使用します。

```
ForeignKey keyname(key_props) References pkg.class(ref_index) [ OnUpdate = onupdate ];
```

onupdate は以下のいずれかになります。

- `noaction` (既定値) ― 外部テーブルの参照されているレコードのキー値の更新を試みた場合、失敗します。
- `cascade` ― 外部テーブルの参照されているレコードのキー値を更新した場合、参照元レコードの外部テーブルへの参照も更新されます。
- `setdefault` ― 外部テーブルで参照先レコードのキー値を更新すると、参照元レコードでのその外部テーブルへの参照はその既定値に設定されます。
- `setnull` ― 外部テーブルの参照されているレコードのキー値を更新した場合、参照元レコードの外部テーブルへの参照は NULL に設定されます。

詳細

外部テーブル内の行のキー値を更新する場合、その外部テーブルに外部キー制約が設定されているすべての参照元テーブルで、更新する行を参照している行があるかどうかがチェックされます。このような参照が見つかった場合、OnUpdate アクションは有効になります。

既定値

既定値は `noaction` です。

関連項目

- 外部キー定義
- 外部キーの使用法
- コンパイラ・キーワードの概要
- ″InterSystems SQL リファレンス″ の ″外部キーの定義″

SqlName (外部キーのキーワード)

外部キーの SQL エイリアスを指定します。

使用法

この外部キーの既定の SQL 名をオーバーライドするには、以下の構文を使用します。

```
ForeignKey keyname(key_props) References pkg.class(ref_index) [ SqlName = alternate_name ];
```

alternate_name は SQL 識別子です。

詳細

このキーワードにより、SQL で参照する際の、この外部キーの別名を定義できます。

既定値

このキーワードを省略すると、外部キーの SQL 名は、外部キー定義で指定された keyname になります。

関連項目

- 外部キー定義
- 外部キーの使用法
- コンパイラ・キーワードの概要

インデックスの構文とキーワード

このリファレンスでは、永続クラスで定義できるインデックスに適用する構文およびキーワードを説明します。キーワード (クラス属性とも呼ばれる) は、一般にコンパイラに影響を与えます。

インデックス定義に関する一般情報へのリンクは、"インデックス定義" を参照してください。

特定のクラス・メンバに適用するすべての構造とキーワードは、目次を参照してください。

クラス定義でのインデックスの構文

インデックス定義の構造を説明します。インデックス定義に関する一般情報へのリンクは、"インデックス定義" を参照してください。

概要

インデックスとは、永続クラスで管理される構造のことです。永続クラスは、クエリおよびその他の関数の最適化に使用することが意図されています。これらのインデックスは、オブジェクトベースの操作と同様、データベースに対して SQL ベースの INSERT、UPDATE、DELETE 演算が実行された場合にも、自動的に維持されます。SQL クエリ・プロセッサは、SQL クエリを作成、実行するときに、利用可能なインデックスを活用します。

インデックス定義は永続クラスに追加できます。他のクラスでは意味がありません。

詳細

インデックス定義の構造は以下のとおりです。

```
/// description
Index name On property_expression_list [ keyword_list ];
```

以下は、この指定の説明です。

- description (オプション) は、クラス・リファレンスでの表示を意図しています。description は既定ではブランクです。"クラス・ドキュメントの作成" を参照してください。
- name (必須) はインデックスの名前です。名前はプロパティの名前付け規約に従い、このクラスまたはテーブル内で一意である必要があります。

  この名前は、レポート作成、インデックス構築、インデックス削除をはじめとするデータベース管理に使用します。プロパティの名前付け規約については、"クラス・メンバ" を参照してください。クラス定義のインデックス名と対応する SQL インデックス名との関係は、"InterSystems SQL リファレンス" の "CREATE INDEX" を参照してください。

  Tipヒント インデックスをプロパティと容易に区別できるよう、名前付け規約に従うと便利です。例えば、すべてのインデックス名の末尾に `IDX` を含めることができます。

- property_expression_list (必須) は、インデックスが基づくプロパティ (複数可) を指定します。さらにこれには各プロパティの照合仕様を含めることもできます。このオプションは、単一のプロパティ式、または括弧で囲んだ複数のプロパティ式のコンマ区切りリストのいずれかにします。

  特定のプロパティ式の構成要素は以下のとおりです。

  - インデックスの作成対象のプロパティの名前。
  - オプションの `(ELEMENTS)` 式または `(KEYS)` 式。これは、コレクションのサブ値に対するインデックス作成手段となります。
  - オプションの照合式。

  詳細は、"インデックスの定義と作成" を参照してください。

- keyword_list (オプション) は、インデックスを詳しく定義したキーワードのコンマ区切りリストです。

  すべてのキーワードは、"インデックスの構文とキーワード" を参照してください。

  このリストを省略する場合は角括弧も省略します。

例えば、以下のクラス定義は、2 つのプロパティとそれぞれのプロパティに基づくインデックスを定義しています。

Class Definition

```
Class MyApp.Student Extends %Persistent
{

Property Name As %String;

Property GPA As %Double;

Index NameIDX On Name;

Index GPAIDX On GPA;
}
```

関連項目

- インデックスの定義と作成
- クラスの制限

Abstract (インデックス・キーワード)

インデックスが抽象かどうかを指定します。

使用法

インデックスを抽象と指定するには、以下の構文を使用します。

```
Index name [ Abstract ];
```

name はインデックスの名前です。インデックスをプロパティと容易に区別できるよう、名前付け規約に従うと便利です。例えば、すべてのインデックス名の末尾に `IDX` を含めることができます。

注釈　シャード・テーブルを作成する際に、抽象シャード・キー・インデックスが自動的に生成されるため、インデックスを定義する必要がなくなります。

詳細

抽象インデックスは、シャード・テーブルでのみ使用することを意図されています。抽象インデックスにはデータが含まれていないため、ストレージは使用されません (インデックス・グローバルの使用なし)。シャード・テーブルには抽象インデックスがちょうど 1 つ設定され、これをシャード・キー・インデックスと呼びます。シャード・キー・インデックスの目的は、行が存在するシャードを指定するキーとして機能することです。

インデックスが抽象として定義されている場合、メソッドや SQL から、そのインデックスにアクセスすることも、そのインデックスを使用することもできません。そのインデックスを一意とマーキングしようとしたり、プライマリ・キー内で使用しようとすると、それらの制約は無視されます。

IdKey インデックスを抽象として定義することはできません。そのようにすると、クラス・コンパイル・エラーが発生します。

このキーワードを既存のインデックスで使用して、それを抽象にすることができます。これによってそのインデックス内の既存のデータが削除されることはありません。

既定値

Abstract キーワードの既定値は、False です。

関連項目

- インデックス定義
- インデックスの定義と作成
- コンパイラ・キーワードの概要

Condition (インデックス・キーワード)

条件付きインデックスを定義し、インデックスに含めるレコードの条件を指定します。

使用法

このキーワードは、既存のアプリケーションを InterSystems SQL に移行するためのものであり、新規のアプリケーションでは使用しません。

関連項目

- インデックス定義
- インデックスの定義と作成
- コンパイラ・キーワードの概要

CoshardWith (インデックス・キーワード)

このクラスとコシャードされるクラスの名前を指定します。

使用法

シャード・クラスが完全に実装されるまで、オブジェクト側からではなく、SQL からシャード・テーブルを作成することをお勧めします。ただし、シャード・テーブルを作成することによって生成したクラスを確認する場合、以下のようなコードを見ることもできます。

```
/// ShardKey index for Sharded table, auto-generated by DDL CREATE TABLE statement
Index ShardKey On DeptNum [ Abstract, CoshardWith = User.Department, ShardKey, SqlName = %ShardKey ];
```

この例では、現在のクラスは User.Department クラスとコシャードされます。

関連項目

- ″スケーラビリティ・ガイド″ の ″シャーディングによるデータ量に応じた InterSystems IRIS の水平方向の拡張″
- インデックス定義
- インデックスの定義と作成
- コンパイラ・キーワードの概要

Data (インデックス・キーワード)

プロパティのリストを指定します。そのプロパティの値は、このインデックスに保存されます。

使用法

インデックス内にプロパティの値を格納するには、以下の構文を使用します。

```
Index name On property_expression_list [ Data = stored_property_list ];
```

以下はその説明です。

- name はインデックスの名前です。インデックスをプロパティと容易に区別できるよう、名前付け規約に従うと便利です。例えば、すべてのインデックス名の末尾に `IDX` を含めることができます。
- stored_property_list は、単一のプロパティ名、または括弧で囲んだ複数のプロパティのコンマ区切りリストのいずれかにします。

詳細

このキーワードは、プロパティのリストを指定します。そのプロパティの値は、このインデックスに保存されます。

このキーワードはビットマップ・インデックスと一緒に使用することはできません。

詳細は、インデックスに関するドキュメントを参照してください。

既定値

このキーワードを省略した場合、インデックス内にプロパティの値は格納されません。

例

```
Index NameIDX On Name [ Data = Name ];
Index ZipIDX On ZipCode [ Data = (City,State) ];
```

関連項目

- インデックス定義
- インデックスの定義と作成
- コンパイラ・キーワードの概要

Extent (インデックス・キーワード)

エクステント・インデックスを定義します。

使用法

これをエクステント・インデックスとして指定するには、以下の構文を使用します。

```
Index name [ Extent ];
```

name はインデックスの名前です。インデックスをプロパティと容易に区別できるよう、名前付け規約に従うと便利です。例えば、すべてのインデックス名の末尾に `IDX` を含めることができます。

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

注釈　ビットマップ・インデックスを使用する場合は、エクステント・インデックスは自動的に追加されるので、定義する必要はありません。

詳細

エクステント・インデックスは、どのオブジェクト・インスタンスがサブクラスに属するのかを追跡するために使用します。

既定値

Extent キーワードの既定値は、False です。

関連項目

- インデックス定義
- インデックスの定義と作成
- コンパイラ・キーワードの概要

IdKey (インデックス・キーワード)

このインデックスがテーブルのオブジェクト識別値を定義するかどうかを指定します。

使用法

このテーブルのオブジェクト識別値を、このインデックスが基づくプロパティから構成することを指定するには、以下の構文を使用します。

```
Index name On property_expression_list [ IdKey ];
```

以下はその説明です。

- name はインデックスの名前です。インデックスをプロパティと容易に区別できるよう、名前付け規約に従うと便利です。例えば、すべてのインデックス名の末尾に IDX を含めることができます。
- property_expression_list は、単一のプロパティ名、または括弧で囲んだ複数のプロパティのコンマ区切りリストのいずれかにします。

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 Not を指定します。

詳細

このキーワードは、このインデックスが基づくプロパティを、このオブジェクトのオブジェクト識別値の作成に使用することを指定します。

重要　IDKEY インデックスによって使用されるどのプロパティの値においても、そのプロパティが永続クラスのインスタンスへの有効な参照でない限り、連続する 2 つの垂直バー (||) は使用しないでください。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、予測できない動作が起こる可能性があります。

オブジェクト識別値は、永続オブジェクト・インスタンスを一意に特定するために使用します。IdKey を使用するオブジェクトがいったん保存された後は、IdKey を作成したプロパティの値は変更できません。

IdKey インデックスは、一意のインデックスと同様の振る舞いをします。つまり、このインデックスで使用するプロパティ (またはプロパティの組み合わせ) の場合は、InterSystems IRIS では必ず一意となります。このインデックス定義で Unique キーワードを True に指定することは、可能ですが不必要です。

IdKey インデックスについては、InterSystems IRIS が、オブジェクトのオープン、存在確認、削除に使用できるメソッドを生成します。"メソッドのオープン、存在確認、および削除" を参照してください。

既定値

IdKey キーワードの既定値は、False です。

関連項目

- インデックス定義
- インデックスの定義と作成
- コンパイラ・キーワードの概要
- メソッドのオープン、存在確認、および削除

Internal (インデックス・キーワード)

このインデックス定義が Internal (クラス・ドキュメントに表示されない) であるかどうかを指定します。

使用法

このインデックス定義を Internal として指定するには、以下の構文を使用します。

```
Index name On property_expression_list [ Internal ];
```

以下はその説明です。

- name はインデックスの名前です。インデックスをプロパティと容易に区別できるよう、名前付け規約に従うと便利です。例えば、すべてのインデックス名の末尾に `IDX` を含めることができます。
- property_expression_list は、単一のプロパティ名、または括弧で囲んだ複数のプロパティのコンマ区切りリストのいずれかにします。

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

Internal クラスのメンバは、クラス・ドキュメントには表示されません。このキーワードは、ユーザに対してクラスは表示するが、そのすべてのメンバを表示する必要がない場合に役立ちます。

既定値

このキーワードを省略すると、このインデックスはクラス・ドキュメントに表示されます。

関連項目

- インデックス定義
- インデックスの定義と作成
- コンパイラ・キーワードの概要

PrimaryKey (インデックス・キーワード)

このインデックスがテーブルの主キーを定義するかどうかを指定します。

使用法

このテーブルの主キーを、このインデックスが基づくプロパティにより構成することを指定するには、以下の構文を使用します。

```
Index name On property_expression_list [ PrimaryKey ];
```

以下はその説明です。

- name はインデックスの名前です。インデックスをプロパティと容易に区別できるよう、名前付け規約に従うと便利です。例えば、すべてのインデックス名の末尾に `IDX` を含めることができます。
- property_expression_list は、単一のプロパティ名、または括弧で囲んだ複数のプロパティのコンマ区切りリストのいずれかにします。

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

このキーワードは、このインデックスが SQL 経由で、このクラス (テーブル) の主キーとしてレポートされることを指定します。

PrimaryKey インデックスは、一意のインデックスと同様の振る舞いをします。つまり、このインデックスで使用するプロパティ (またはプロパティの組み合わせ) の場合は、InterSystems IRIS では必ず一意となります。このインデックス定義で Unique キーワードを True に指定することは、可能ですが不必要です。

主キーのインデックスについては、InterSystems IRIS が、オブジェクトのオープン、存在確認、削除に使用できるメソッドを生成します。"メソッドのオープン、存在確認、および削除" を参照してください。

例

Class Member

```
Index EmpIDX On EmployeeID [ PrimaryKey] ;
```

既定値

このキーワードを省略すると、このテーブルの主キーは、このインデックスが基づくプロパティにより構成されません。

関連項目

- インデックス定義
- インデックスの定義と作成
- コンパイラ・キーワードの概要
- メソッドのオープン、存在確認、および削除

ShardKey (インデックス・キーワード)

このクラスに対するシャード・キーを指定します。

使用法

シャード・クラスが完全に実装されるまで、オブジェクト側からではなく、SQL からシャード・テーブルを作成することをお勧めします。ただし、シャード・テーブルを作成することによって生成したクラスを確認する場合、以下のようなコードを見ることもできます。

```
/// ShardKey index for Sharded table, auto-generated by DDL CREATE TABLE statement
Index ShardKey On DeptNum [ Abstract, CoshardWith = User.Department, ShardKey, SqlName = %ShardKey ];
```

この例で、DeptNum プロパティは、現在のクラスのシャード・キーです。

関連項目

- ″スケーラビリティ・ガイド″ の ″シャーディングによるデータ量に応じた InterSystems IRIS の水平方向の拡張″
- インデックス定義
- インデックスの定義と作成
- コンパイラ・キーワードの概要

SqlName (インデックス・キーワード)

インデックスの SQL エイリアスを指定します。

使用法

SQL で参照する際のこのインデックスの既定の名前をオーバーライドするには、以下の構文を使用します。

```
Index name On property_expression_list [ SqlName = sqlindexname];
```

以下はその説明です。

- name はインデックスの名前です。インデックスをプロパティと容易に区別できるよう、名前付け規約に従うと便利です。例えば、すべてのインデックス名の末尾に `IDX` を含めることができます。
- property_expression_list は、単一のプロパティ名、または括弧で囲んだ複数のプロパティのコンマ区切りリストのいずれかにします。
- sqlindexname は SQL 識別子です。

詳細

このキーワードにより、SQL で参照する際の、このインデックスの別名を定義できます。

既定値

このキーワードを省略すると、インデックスの SQL 名は、インデックス定義で指定された indexname になります。

関連項目

- インデックス定義
- インデックスの定義と作成
- コンパイラ・キーワードの概要

Type (インデックス・キーワード)

インデックスのタイプを指定します。

使用法

インデックスのタイプを指定するには、以下の構文を使用します。

```
Index name On property_expression_list [ Type = indextype ];
```

以下はその説明です。

- name はインデックスの名前です。インデックスをプロパティと容易に区別できるよう、名前付け規約に従うと便利です。例えば、すべてのインデックス名の末尾に `IDX` を含めることができます。
- property_expression_list は、単一のプロパティ名、または括弧で囲んだ複数のプロパティのコンマ区切りリストのいずれかにします。
- indextype は、以下のいずれかです。
  - `bitmap` ― ビットマップ・インデックス
  - `bitslice` ― ビットスライス・インデックス
  - `collatedkey` ― そのプロパティの照合を使用する、特定のプロパティに対するインデックス
  - `columnar` ― 列指向インデックス
  - `index` ― 標準インデックス (既定)
  - `key` ― 非推奨

詳細

このキーワードは、インデックスのタイプ、つまりインデックスを標準のインデックスとして実装するか、いくつかの特殊インデックスの 1 つとして実装するかを指定します。

ビットマップ・インデックスは、unique にすることはできません。

既定値

このキーワードを省略すると、インデックスは標準インデックスになります。

関連項目

- インデックス定義
- インデックスの定義と作成
- コンパイラ・キーワードの概要

Unique (インデックス・キーワード)

インデックスを一意とするかどうかを指定します。

使用法

InterSystems IRIS がこのインデックスが基づくプロパティを一意とするように指定するには、以下の構文を使用します。

```
Index name On property_expression_list [ Unique ];
```

以下はその説明です。

- name はインデックスの名前です。インデックスをプロパティと容易に区別できるよう、名前付け規約に従うと便利です。例えば、すべてのインデックス名の末尾に `IDX` を含めることができます。
- property_expression_list は、単一のプロパティ名、または括弧で囲んだ複数のプロパティのコンマ区切りリストのいずれかにします。

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

Unique キーワードがある場合、このインデックスが一意であることを示します。

一意のインデックスによりインデックス付けされたプロパティは、そのインデックスを定義するクラス (テーブル) のエクステント (すべてのオブジェクトのセット) 内で、一意の値を持つように制約されます (同じ照合値を持つインスタンスは存在しません)。

さらに、一意のインデックスをビットマップ・インデックスにすることもできません。

一意のインデックスについては、InterSystems IRIS が、オブジェクトのオープン、存在確認、削除に使用できるメソッドを生成します。"メソッドのオープン、存在確認、および削除" を参照してください。

例

Class Member

```
Index SSNIdx On SSN [ Unique ] ;
```

既定値

このキーワードを省略すると、InterSystems IRIS はこのインデックスが基づくプロパティを一意にしません。

関連項目

- インデックス定義
- インデックスの定義と作成
- コンパイラ・キーワードの概要
- メソッドのオープン、存在確認、および削除

メソッドの構文とキーワード

このリファレンスでは、メソッドに適用する構文とキーワードについて説明しています。キーワード (クラス属性とも呼ばれる) は、一般にコンパイラに影響を与えます。

メソッド定義に関する一般情報へのリンクは、"メソッド定義" を参照してください。

特定のクラス・メンバに適用するすべての構造とキーワードは、目次を参照してください。

クラス定義でのメソッドの構文

メソッド定義の構造を説明します。メソッド定義に関する一般情報へのリンクは、"メソッド定義" を参照してください。

概要

ほとんどの場合、メソッド定義は、メソッドの実行時動作を定義します。InterSystems IRIS はメソッド・ジェネレータもサポートします。これは実行時に使用するコードを生成する特殊なメソッドです。

詳細

メソッド定義の構造は以下のとおりです。

```
/// description
Method name(formal_spec) As returnclass [ keyword_list ]
{ implementation }
```

または (クラス・メソッドでは) 以下のとおりです。

```
/// description
ClassMethod name(formal_spec) As returnclass [ keyword_list ]
{ implementation }
```

または (クライアント・メソッドでは) 以下のとおりです。

```
/// description
ClientMethod name(formal_spec) As returnclass [ keyword_list ]
{ implementation }
```

以下は、この指定の説明です。

- description (オプション) は、クラス・リファレンスでの表示を意図しています。description は既定ではブランクです。"クラス・ドキュメントの作成" を参照してください。
- name (必須) はメソッドの名前です。これは、有効なクラス・メンバ名で、かつ他のクラス・メンバ名と重複しないものである必要があります。
- formal_spec (オプション) は、メソッドとやり取りする引数のリストを指定します。

  その形式仕様は、メソッドの引数とそのタイプ、呼び出しタイプ (ByRef、Output、または ByVal)、オプションの既定値のリストです。Output の呼び出しタイプは、参照によって渡される引数を示すために使用されますが、受信する値は名目上使用されません。
- returnclass (オプション) は、このメソッドから返される値のタイプ (ある場合) を指定します。returnclass を省略する場合は、単語 As も省略します。
- keyword_list (オプション) は、さらにメソッドを定義するキーワードのコンマ区切りリストです。

  すべてのキーワードは、"メソッドの構文とキーワード" を参照してください。

  このリストを省略する場合は角括弧も省略します。
- implementation (オプション) は、メソッドの実行内容を定義するコードのゼロ以上の行です。

  使用するプログラミング言語は、クラス・レベルの Language またはメソッド・レベルの Language キーワードで指定します。

パラメータ値

formal_spec および returnclass には、クラス名の後にオプションのパラメータ値を指定できます。メソッドを SQL ストアド・プロシージャとして使用する場合、これらのパラメータ値を ODBC または JDBC クライアントに対して追加情報を提供するために使用します。その他の場合は、これらのパラメータはすべて無視されます。以下に例を示します。

Class Member

```
ClassMethod MyProc(data As %String(MAXLEN = 85)) As %Integer [ SQLProc ]
{
  Quit 22
}
```

別の例を示します。

Class Member

```
ClassMethod GetName() As %String(MAXLEN=222) [ SQLProc ]
{
  Quit "Mo"
}
```

関連項目

- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- クラスの制限

Abstract (メソッド・キーワード)

抽象メソッドであるかどうかを指定します。

使用法

このメソッドを Abstract として指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ Abstract ] {    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 Not を指定します。

詳細

抽象メソッドは、実装を持たないため、これに対して生成される実行可能なコードもありません。抽象メソッドは、メソッド・シグニチャ (または、インタフェース) を定義するためにのみ存在します。これは、1 つ以上のサブクラス内でオーバーライドし、実装できます。InterSystems IRIS クラス・ライブラリ内で定義された (実装はされていません) さまざまなコールバック・メソッドは、抽象メソッドの一例です。

既定値

このキーワードを省略すると、メソッドは Abstract ではなくなります。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

ClientName (メソッド・キーワード)

クライアント・プロジェクション内のメソッドの既定の名前をオーバーライドします。

使用法

クラスがクライアント言語に投影されるときのこのメソッドの既定の名前をオーバーライドするには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ ClientName = clientname ]
{    //implementation }
```

clientname はクライアント言語で使用する名前です。

詳細

このキーワードにより、メソッドにクライアントの言語を投影するときに、メソッドに別名を定義できます。このキーワードは、メソッド名にクライアント言語では許可されない文字が含まれている場合に特に役立ちます。

既定値

このキーワードを省略すると、メソッド名がクライアント名として使用されます。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

CodeMode (メソッド・キーワード)

このメソッドの実装方法を指定します。

使用法

メソッドの実装方法を指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ CodeMode=codemode ]
{    //implementation }
```

codemode は以下のいずれかになります。

- `call` ― このメソッドは、ルーチン呼び出しのエイリアスです (従来のコードを囲むために使用します)。
- `code` (既定) ― このメソッドは、コード行として実装されます。
- `expression` ― このメソッドは、式として実装されます。
- `objectgenerator` ― このメソッドは、メソッド・ジェネレータです。

注釈　このキーワードには、`generator` という値があります。これは、以前に使用されていたキーワードで、従来の非オブジェクト・ベースのメソッド・ジェネレータを使う必要があることを示しています。この値は、従来のバージョンとの互換性を保つ目的にのみ残されています。新しいアプリケーションでは `objectgenerator` を使用する必要があります。

詳細

このキーワードは、任意のメソッドの実装方法を指定します。

通常、メソッドは 1 行以上の行のコードを使用して実装します。これは、CodeMode の既定値の `code` で指定されます。この場合、メソッド実装は 1 行以上のコードになります。

一部の単純なメソッドは、expression メソッドとして実装できます。クラス・コンパイラは、このメソッドに対する呼び出しを、式を含むインライン・コードと置換することもあります。この場合、メソッド実装は (no Quit または Return 文とあわせて) 単純な式となります。

呼び出しメソッドは、ルーチンのラッパです。この場合、メソッド実装はルーチン名とタグ名になります。

メソッド・ジェネレータは、クラスのコンパイル時にクラス・コンパイラによって実行されるプログラムです。これは、任意のメソッドに対する実際の実装を生成します。この場合、メソッド実装はメソッド・ジェネレータ用のコードとなります。"クラスの定義と使用" の "メソッド・ジェネレータとトリガ・ジェネレータの定義" を参照してください。

既定値

CodeMode キーワードの既定値は、`code` です。

例

```
/// An expression method
Method Double(val As %Integer) As %Integer [ CodeMode = expression ]
{
  val * 2
}

/// A Method generator
Method GetClassName() As %String [ CodeMode = objectgenerator ]
{
   Do %code.WriteLine(" Quit """ _ %class.Name _ """")
   Quit $$$OK
}
```

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

Deprecated (メソッド・キーワード)

このメソッドを非推奨として指定します。このキーワードはクラス・コンパイラでは無視され、単にメソッドが非推奨であることを人間が読める形で示します。

使用法

このメソッドを非推奨として指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ Deprecated ]
{    //implementation }
```

そうしない場合は、このキーワードを省略するか、キーワードの直前に単語 `Not` を配置します。

関連項目

- メソッド定義

ExternalProcName (メソッド・キーワード)

外部データベース内のストアド・プロシージャとして使用される場合の、このメソッドの名前を指定します。メソッドがストアド・プロシージャとして投影される場合にのみ適用されます。

使用法

外部データベース内のストアド・プロシージャとして使用される場合の、メソッドの既定の名前をオーバーライドするには、以下の構文を使用します。

```
ClassMethod name(formal_spec) As returnclass [ SqlProc, ExternalProcName = MyProcedure ]
{    //implementation }
```

MyProcedure は、引用符で囲まれていない文字列です。

詳細

このキーワードにより、このメソッドを外部データベース内のストアド・プロシージャとして使用する場合の名前を定義できます。

既定値

このキーワードを省略すると、メソッド名がストアド・プロシージャ名として使用されます。

関連項目

- メソッド定義
- SqlProc キーワード
- ストアド・プロシージャの定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

Final (メソッド・キーワード)

このメソッドが Final である (サブクラス内でオーバーライドできない) かどうかを指定します。

使用法

メソッドを Final として指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ Final ]
{    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 Not を指定します。

詳細

final としてマークされるクラス・メンバは、サブクラスではオーバーライドできません。

既定値

このキーワードを省略すると、メソッドは Final ではなくなります。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

ForceGenerate (メソッド・キーワード)

メソッドをすべてのサブクラスでコンパイルする必要があるかどうかを指定します。メソッドがメソッド・ジェネレータである場合にのみ適用されます。

使用法

メソッド (メソッド・ジェネレータ) をすべてのサブクラスでコンパイルする必要があることを指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ CodeMode = ObjectGenerator, ForceGenerate ]
{    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 Not を指定します。

詳細

メソッド・ジェネレータのメソッドの場合はそのメソッドをすべてのサブクラスでコンパイルする必要があることを示します。このキーワードは、各サブクラスに必ずそのバージョンのメソッドが含まれるようにする必要があるときに役立ちます。InterSystems IRIS は、生成されるコードの外見がスーパークラスで生成されるコードと同じと見なした場合には、サブクラス内のメソッドをリコンパイルしません。このロジックでは、両方のクラスのインクルード・ファイルが同じかどうかについては考慮しません。このメソッドがインクルード・ファイルで定義されたマクロを使用する場合、それとは別のインクルード・ファイルをサブクラスで使用するとしても、InterSystems IRIS はサブクラスのメソッドをリコンパイルしません。このようなシナリオでは、メソッド・ジェネレータのために ForceGenerate を指定します。

既定値

このキーワードを省略すると、メソッドはすべてのサブクラスでコンパイルされなくなります。

関連項目

- メソッド定義
- CodeMode キーワード
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

GenerateAfter (メソッド・キーワード)

このメソッドをいつ生成するかを指定します。メソッドがメソッド・ジェネレータである場合にのみ適用されます。

使用法

このメソッドのジェネレータが他のメソッドの生成後に呼び出されるように指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ CodeMode = ObjectGenerator, GenerateAfter = methodlist ]
{    //implementation }
```

methodlist は、単一のメソッド名、または括弧で囲んだ複数のメソッド名のコンマ区切りリストのいずれかにします。

詳細

メソッド・ジェネレータのメソッドで、リストされたメソッドが生成された後にジェネレータを実行することを指定します。このキーワードは、使用するメソッド・ジェネレータの実行順序を管理する必要がある場合に役立ちます。

関連項目

- メソッド定義
- CodeMode キーワード
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

Internal (メソッド・キーワード)

このメソッド定義が Internal (クラス・ドキュメントに表示されない) であるかどうかを指定します。

使用法

このメソッドを Internal として指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ Internal ]
{    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

Internal クラスのメンバは、クラス・ドキュメントには表示されません。このキーワードは、ユーザに対してクラスは表示するが、そのすべてのメンバを表示する必要がない場合に役立ちます。

既定値

このキーワードを省略すると、このメソッドはクラス・ドキュメントに表示されます。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

Language (メソッド・キーワード)

このメソッドの実装に使用する言語を指定します。

使用法

このメソッドの実装に使用する言語を指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ Language = language ]
{    //implementation }
```

language は以下のいずれかになります。

- `objectscript` (既定) ー ObjectScript
- `python` ー 組み込み Python
- `tsql` ー Transact-SQL
- `ispl` ー Informix ストアド・プロシージャ言語

詳細

このキーワードは、このメソッドの実装に使用する言語を指定します。

値 `ispl` および `tsql` は、クラス・メソッドでのみサポートされます。

`ispl` の値を指定すると、メソッドの本文が単一の CREATE PROCEDURE 文に制限されます。

既定値

このキーワードを省略すると、クラス・レベルの Language キーワードで指定された言語が使用されます。

注釈　クラス・レベルでは `Language = ispl` も `Language = python` も指定できません。これらの値はメソッドにのみ使用できます。

注釈　シャード・クラスのメソッドは、ObjectScript 以外の言語を使用して実装することはできません。

例

```
Class User.Person Extends %Persistent
{

Property Name As %String;

Property Gender As %String;

/// An ObjectScript instance method that writes the name and gender of a person
Method Write() [ Language = objectscript ]
{
   write !, ..Name, " is a ", ..Gender, !
}

/// An Embedded Python instance method that prints the name and gender of a person
Method Print() [ Language = python ]
{
   print('\n' + self.Name + ' is a ' + self.Gender)
}

/// A TSQL class method that inserts a row into the Person table
ClassMethod TSQLTest() [ Language = tsql ]
{
   INSERT INTO Person (Name, Gender) VALUES ('Manon', 'Female')
}

/// An ISPL class method that creates a stored procedure named IsplSp
ClassMethod ISPLTest() [ Language = ispl ]
```

```
{
   CREATE PROCEDURE IsplSp()
      INSERT INTO Person (Name, Gender) VALUES ('Nikolai', 'Male')
   END PROCEDURE
}

}
```

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

NotInheritable (メソッド・キーワード)

このメソッドをサブクラスで継承できるかどうかを指定します。

使用法

このメソッドをサブクラスで継承できないように指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ NotInheritable ]
{    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 Not を指定します。

詳細

このキーワードは、このメソッドをサブクラスで継承できないことを指定します。

重要　　このキーワードを使用して、メンバをそのサブクラスに継承できないようにすると非常に効果的な場合もありますが、継承が中断されるので、限られた場合にのみ慎重に使用してください。

既定値

このキーワードを省略すると、このメソッドは継承可能になります。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

PlaceAfter (メソッド・キーワード)

クラスに対して生成されるルーチン内での、他のメソッドを基準としたこのメソッドの順序を指定します。

使用法

クラス・コンパイラが、クラスを生成するルーチン内で、リストされたメソッドの後にこのメソッドを置くように指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ PlaceAfter = methodlist ]
{    //implementation }
```

methodlist は、単一のメソッド名、または括弧で囲んだ複数のメソッド名のコンマ区切りリストのいずれかにします。

詳細

このキーワードは、クラス・コンパイラが、クラスを生成するルーチン内で、リストされたメソッドの後にこのメソッドを置くように指定します。このキーワードは、クラス・コンパイラが、メソッドに対するコードを生成する順番を管理する必要がある場合に使用します (一般的なケースではありません)。

既定値

このキーワードを省略すると、クラス・コンパイラはその通常のロジックを使用して、生成するルーチン内でのメソッドの順序を決定します。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

Private (メソッド・キーワード)

このメソッドが Private (このクラスまたはそのサブクラスのメソッドによってのみ呼び出せる) であるかどうかを指定します。

使用法

このメソッドを Private として指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ Private ]
{    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

プライベート・クラス・メンバは、同じクラス (またはそのサブクラス) の他のメンバによってのみ使用できます。他の言語では、この種の可視性を表すために protected という用語を、サブクラスから見えないことを表すために private という用語を使用していることがよくあるので注意してください。

このキーワードは継承されますが、サブクラス内でその値を変更可能です。

既定値

このキーワードを省略すると、このメソッドは Private ではなくなります。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

ProcedureBlock (メソッド・キーワード)

このメソッドがプロシージャ・ブロックであるかどうかを指定します。メソッドが ObjectScript で記述されている場合にのみ適用されます。

使用法

クラス定義は、クラス内のメソッドが、既定でプロシージャ・ブロックであるかどうかを指定します。既定をオーバーライドして、特定のメソッドがプロシージャ・ブロックであることを指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ ProcedureBlock ]
{    //implementation }
```

または (同様に)

```
Method name(formal_spec) As returnclass [ ProcedureBlock=1 ]
{    //implementation }
```

そのようにせずに、特定のメソッドがプロシージャ・ブロックでないことを指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ ProcedureBlock=0 ]
{    //implementation }
```

詳細

このキーワードは、ObjectScript メソッドがプロシージャ・ブロックであることを指定します。

ObjectScript では、メソッドはプロシージャ・ブロックとして実装することも、しないこともできます。プロシージャ・ブロックは、変数の範囲設定を強制します。メソッドからは、呼び出し元で定義されている変数は見えません。新しいアプリケーションはプロシージャ・ブロックを使用しますが、従来のアプリケーションとの互換性を保つために、非プロシージャ・ブロックも存在します。

既定値

このキーワードを省略すると、クラス・レベルの ProcedureBlock キーワードの値が使用されます。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

PublicList (メソッド・キーワード)

このメソッドのパブリック変数を指定します。メソッドが ObjectScript で記述されており、プロシージャ・ブロックである場合にのみ適用されます。

使用法

メソッドに対するパブリック変数のリストを指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ PublicList = variablelist ]
{    //implementation }
```

publiclist は、単一の変数名、または括弧で囲んだ複数の変数名のコンマ区切りリストのいずれかにします。

詳細

このキーワードは、メソッドが ObjectScript で記述されており、プロシージャ・ブロックである場合にのみ使用されます。ObjectScript では、パブリック・リストはパブリック変数のスコープを持つ一連の変数を指定します。パブリック変数は、パブリック・リストを定義しているメソッドから呼び出されたすべてのメソッドから見えます。

既定値

このキーワードを省略すると、メソッドはパブリック変数を持ちません。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

Requires (メソッド・キーワード)

ユーザまたはプロセスがこのメソッドを呼び出すために必要な特権のリストを指定します。

使用法

このメソッドの使用を、指定した特権を持つユーザまたはプロセスに限定することを指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ Requires = privilegelist ]
{    //implementation }
```

privilegelist は、単一の特権、または引用符で囲んだ複数の特権のコンマ区切りリストのいずれかにします。各特権は、resource:permission の形式をとります。permission は、Use、Read、または Write (あるいは 1 文字の省略形 U、R、または W) です。

1 つのリソースに複数の許可を指定するには、1 文字の省略形を使用します。

詳細

メソッドを呼び出すには、ユーザまたはプロセスは、特権のリストにあるすべての特権を持っている必要があります。指定された特権がない場合にこのメソッドを呼び出すと、<PROTECT> エラーが発生します。

メソッドがスーパークラスから Requires キーワードを継承する場合、キーワードに新しい値を設定して、必要な特権のリストに追加できます。必要な特権をこのように削除することはできません。

既定値

このキーワードを省略すると、このメソッドの呼び出しに特別な特権は不要になります。

例

以下のメソッドでは、Sales データベースの読み取り許可と、Marketing データベースへの書き込み許可が必要です (データベースへの書き込み許可を持っている場合、読み取り許可は自動的に付与されます)。

```
ClassMethod UpdateTotalSales() [ Requires = "%DB_SALES: Read, %DB_MARKETING: Write" ]
{
   set newSales = ^["SALES"]Orders
   set totalSales = ^["MARKETING"]Orders
   set totalSales = totalSales + newSales
   set ^["MARKETING"]Orders = totalSales
}
```

1 つのリソースに複数の許可を指定するには、1 文字の省略形を使用します。以下の 2 つのメソッドは機能的に同等です。

```
ClassMethod TestMethod() [ Requires = "MyResource: RW" ]
{
   write "You have permission to run this method"
}

ClassMethod TestMethodTwo() [ Requires = "MyResource: Read, MyResource: Write" ]
{
   write "You have permission to run this method"
}
```

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- "承認ガイド" の "特権および許可"

- コンパイラ・キーワードの概要

ReturnResultsets (メソッド・キーワード)

このメソッドが結果セットを返す (それにより ODBC クライアントと JDBC クライアントがそれらを取得できる) かどうかを指定します。

使用法

メソッドが少なくとも 1 つの結果セットを返すことを指定するには、以下の構文を使用します。

```
ClassMethod name(formal_spec) As returnclass [ ReturnResultsets, SqlName = CustomSets, SqlProc ]
{    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 Not を指定します。

詳細

このキーワードは、メソッドが少なくとも 1 つの結果セットを返すことを指定します。メソッドが 1 つ以上の結果セットを返す場合は、このキーワードを True に設定します。設定を行わない場合、JDBC などのデータベース・ドライバは結果セットを取得できません。

ストアド関数には、Not ReturnResultsets を指定するか、またはこのキーワードを指定しません。

既定値

このキーワードを省略する場合、データベース・ドライバは結果セットを取得できません。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- ストアド・プロシージャの定義
- ストアド関数
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

ServerOnly (メソッド・キーワード)

このメソッドを Java クライアントに投影するかどうかを指定します。

使用法

InterSystems IRIS がメソッドを Java クライアントに投影する方法をオーバーライドするには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ ServerOnly=n ]
{    //implementation }
```

n は以下のいずれかになります。

- 0 は、このメソッドを投影できることを表します。
- 1 は、このメソッドを投影しないことを表します。

詳細

このキーワードは、メソッドを Java クライアントに投影しないことを指定します。

ヒント

サーバのみで使用できるクラスのメソッドを確認するには、ターミナルで以下のユーティリティを使用します。

```
do dumpMethods^%occLGUtil("Sample.Person")
```

引数は、完全修飾されたクラス名です。このユーティリティは、メソッドがスタブかどうか、メソッドがサーバのみのメソッドかどうか、派生元のプロパティ(メソッドがプロパティから派生している場合)など、各メソッドの基本情報を示すレポートを生成します。

既定値

このキーワードを省略すると、このメソッドがスタブ・メソッドの場合は投影しません (ただし、スタブ・メソッドでない場合は投影します)。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

SoapAction (メソッド・キーワード)

このメソッドを HTTP 経由の Web メソッドとして呼び出す場合に HTTP ヘッダで使用する SOAP アクションを指定します。Web サービスまたは Web クライアントとして定義されているクラス内でのみ適用されます。

使用法

このメソッドを Web メソッドとして使用する場合に HTTP ヘッダで使用する SOAP アクションを指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ WebMethod, SoapAction = soapaction ]
{    //implementation }
```

soapaction は以下のいずれかになります。

- `"[ default ]"` ー InterSystems IRIS は、SOAP アクションの既定値である NAMESPACE/Package.Class.Method を使用します。
- `"customValue"` ー InterSystems IRIS は SOAP アクションとして `customValue` を使用します。値は、SOAP 要求の目的を示す URI である必要があります。

  カスタム値を指定する場合は、それが Web サービス内の各 Web メソッド内で一意であるか、Web メソッドごとに SoapRequestMessage キーワードを指定する (また、そのキーワードのために一意の値を使用する) 必要があります。
- `""` ー InterSystems IRIS は SOAP アクションとして空の値を使用します。これは一般的なケースではありません。

詳細

Web メソッドの SOAP アクションは、通常、SOAP 要求メッセージの転送に使用されます。例えば、ファイアウォールは、SOAP 要求メッセージを適切にフィルタ処理するためにこれを使用できます。InterSystems IRIS Web サービスのサービスは、SOAP アクションをそのメッセージ自体と組み合わせて使用して、要求メッセージの処理方法を決定します。

このキーワードにより、このメソッドを Web メソッドとして呼び出す場合に使用する HTTP SOAP アクションを指定できます。SOAP 1.1 の場合、SOAP アクションは、`SOAPAction` HTTP ヘッダとして含まれます。SOAP 1.2 の場合は、`Content-Type` HTTP ヘッダ内に含まれます。

既定値

SoapAction キーワードを省略すると、SOAP アクションは以下のような形式になります。

```
NAMESPACE/Package.Class.Method
```

NAMESPACE は Web サービスの NAMESPACE パラメータの値、Package.Class は Web サービス・クラスの名前、Method は Web メソッドの名前です。

WSDL との関係

SoapAction キーワードは、Web サービスの WSDL の `<binding>` セクションに影響します。例えば、以下の Web メソッドを考えてみます。

Class Member

```
Method Add(a as %Numeric,b as %Numeric) As %Numeric [ SoapAction = MySoapAction,WebMethod ]
{
    Quit a + b
}
```

この Web サービスでは、WSDL の `<binding>` セクションは次のようになります。

XML

```
<binding name="MyServiceNameSoap" type="s0:MyServiceNameSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="Add">
        <soap:operation soapAction="MySoapAction" style="document"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>
```

既定では、メソッドで SoapAction キーワードを指定しないと、<soap:operation> 要素が次のようになる場合があります。

XML

```
<soap:operation soapAction="http://www.mynamespace.org/ROBJDemo.BasicWS.Add" style="document"/>
```

SOAP ウィザードを使用して WSDL から InterSystems IRIS Web サービスのサービスまたはクライアントを生成する場合、その WSDL に応じてこのキーワードが設定されます。

メッセージへの影響

前述の Web メソッドでは、Web サービスは次の形式の要求メッセージを想定します (SOAP 1.1 の場合)。

```
POST /csp/gsop/ROBJDemo.BasicWS.cls HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; InterSystems IRIS;)
Host: localhost:8080
Connection: Close
Accept-Encoding: gzip
SOAPAction: MySoapAction
Content-Length: 379
Content-Type: text/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope >...
```

既定では、メソッドで SoapAction キーワードを指定しないと、SOAPAction の行が次のようになる場合があります。

```
SOAPAction: http://www.mynamespace.org/ROBJDemo.BasicWS.Add
```

SOAP 1.2 では、細部が多少異なります。この場合、Web サービスは次の形式の要求メッセージを想定します。

```
POST /csp/gsop/ROBJDemo.BasicWS.cls HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; InterSystems IRIS;)
Host: localhost:8080
Connection: Close
Accept-Encoding: gzip
Content-Length: 377
Content-Type: application/soap+xml; charset=UTF-8; action="MySoapAction"

<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope >...
```

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要
- Web サービスおよび Web クライアントの作成

SoapBindingStyle (メソッド・キーワード)

Web メソッドとして使用される場合に、このメソッドが使用するバインディング・スタイルまたは SOAP 呼び出し機能を指定します。Web サービスまたは Web クライアントとして定義されているクラス内でのみ適用されます。

使用法

メソッドが (Web メソッドとして使用される場合に) 使用する既定のバインディング・スタイルをオーバーライドするには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ WebMethod, SoapBindingStyle = soapbindingstyle ]
{    //implementation }
```

soapbindingstyle は以下のいずれかになります。

- `document` (既定) ー この Web メソッドはドキュメント・スタイルの呼び出しを使用します。

  このバインディング・スタイルにより、SOAP メッセージはドキュメントとしてフォーマットされ、通常はパートを 1 つのみ持ちます。

  SOAP メッセージでは、通常、`<Body>` 要素に 1 つの子要素が含まれます。`<Body>` 要素のそれぞれの子は、メッセージ・パートに対応します。

- `rpc` ー この Web メソッドは RPC (リモート・プロシージャ呼び出し) スタイルの呼び出しを使用します。

  このバインディング・スタイルにより、SOAP メッセージは複数のパートを持つメッセージとしてフォーマットされます。

  SOAP メッセージでは、`<Body>` 要素に 1 つの子要素が含まれ、その名前は対応する処理名から取得されます。この要素は生成されたラッパ要素であり、これにはメソッドの引数リストの引数ごとに 1 つの子要素が含まれます。

重要　手動で作成した Web サービスでは、通常はこのキーワードの既定値が最適な値となります。SOAP ウィザードで WSDL から Web クライアントまたは Web サービスを生成すると、このキーワードはその WSDL に最適に設定されます。この値を変更すると、Web クライアントまたは Web サービスは機能しなくなる場合があります。

詳細

このキーワードにより、Web メソッドが使用するバインディング・スタイルを指定できます。これは、SOAP 本文の形式に影響します (SOAP ヘッダの形式には影響しません)。

このキーワードは、指定されたメソッドに対して、SoapBindingStyle クラス・キーワードをオーバーライドします。

既定値

このキーワードを省略すると、`<soap:operation>` 要素の `style` 属性が、代わりに SoapBindingStyle クラス・キーワードの値により決定されます。

WSDL との関係

SoapBindingStyle メソッド・キーワードにより、WSDL の `<binding>` セクション内の `<soap:operation>` 要素の `style` 属性の値を指定します。例えば、SoapBindingStyle メソッド・キーワードが `document` の場合、WSDL は次のようになります。

```
...
<binding ...>
 ...
    <operation ...>
        <soap:operation ... style="document"/>
...
```

一方、SoapBindingStyle が rpc の場合は、WSDL は次のようになります。

```
...
<binding ...>
 ...
    <operation ...>
        <soap:operation ... style="rpc"/>
...
```

バインディング・スタイルは、次のように、Web メソッドの要求または応答の <message> 要素にも影響します。

- バインディング・スタイルが document の場合、既定では各メッセージはパートを 1 つのみ持ちます。以下に例を示します。

  ```
  <message name="AddSoapIn">
      <part name="parameters" .../>
  </message>
  ```

  ARGUMENTSTYLE パラメータが message の場合、メッセージは複数のパートを持つことができます。以下に例を示します。

  ```
  <message name="AddSoapIn">
     <part name="a" .../>
     <part name="b" .../>
  </message>
  ```

- バインディング・スタイルが rpc の場合、メッセージは複数のパートを持つことができます。以下に例を示します。

  ```
  <message name="AddSoapIn">
      <part name="a" .../>
      <part name="b" .../>
  </message>
  ```

SOAP メッセージへの影響

詳細は、SoapBindingStyle クラス・キーワードのエントリを参照してください。

%XML.DataSet での使用

タイプ %XML.DataSet のオブジェクトを入力または出力として使用するメソッドでこのキーワードを使用する場合は、いくつかの制約が適用されます。詳細は、SoapBindingStyle クラス・キーワードのエントリを参照してください。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要
- Web サービスおよび Web クライアントの作成

SoapBodyUse (メソッド・キーワード)

Web メソッドとして使用される場合に、このメソッドの入出力で使用されるエンコードを指定します。Web サービスまたは Web クライアントとして定義されているクラス内でのみ適用されます。

使用法

メソッド (Web メソッドとして使用される場合) の入力および出力により使用される既定のエンコードをオーバーライドするには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ WebMethod, SoapBodyUse = soapbodyuse ]
{    //implementation }
```

soapbodyuse は以下のいずれかになります。

- `literal` (既定) ー この Web メソッドは、リテラル・データを使用します。つまり、SOAP メッセージの `<Body>` の中の XML は、WSDL で提供されるスキーマと一致します。
- `encoded` ー この Web メソッドは、SOAP でエンコードされたデータを使用します。つまり、SOAP メッセージの `<Body>` の中の XML では、以下の仕様に従い、使用される SOAP のバージョンに応じて SOAP エンコードを使用します。
  - SOAP 1.1 (https://www.w3.org/TR/2000/NOTE-SOAP-20000508/)
  - SOAP 1.2 (https://www.w3.org/TR/soap12-part2/)

重要　手動で作成した Web サービスでは、通常はこのキーワードの既定値が最適な値となります。SOAP ウィザードで WSDL から Web クライアントまたは Web サービスを生成すると、このキーワードはその WSDL に最適に設定されます。この値を変更すると、Web クライアントまたは Web サービスは機能しなくなる場合があります。

詳細

このキーワードにより、Web メソッドの入出力のエンコードを指定します。

このキーワードは、指定された Web メソッドに対して、SoapBodyUse クラス・キーワードをオーバーライドします。

既定値

このキーワードを省略すると、SoapBodyUse クラス・キーワードの値が代わりに使用されます。

WSDL との関係と SOAP メッセージへの影響

詳細は、SoapBodyUse クラス・キーワードのエントリを参照してください。

%XML.DataSet での使用

タイプ %XML.DataSet のオブジェクトを入力または出力として使用するメソッドでこのキーワードを使用する場合は、いくつかの制約が適用されます。詳細は、SoapBindingStyle クラス・キーワードのエントリを参照してください。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- コンパイラ・キーワードの概要
- Web サービスおよび Web クライアントの作成

SoapMessageName (メソッド・キーワード)

この Web メソッドに対する応答メッセージの <part> 要素の name 属性を指定します。Web サービスまたは Web クライアントとして定義されているクラス内でのみ適用されます。

使用法

応答メッセージの <part> 要素の既定名をオーバーライドするには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ WebMethod, SoapMessageName = MyResponse ]
{    //implementation }
```

soapmessagename は、XML で有効な任意の識別子です。

詳細

注釈　このキーワードは、SoapBindingStyle が (既定の) `document` に設定されている Web メソッドにのみ影響します。

このキーワードにより、応答メッセージの本文の子要素名を指定します。

既定値

このキーワードを省略すると、メッセージ名は、Web メソッド名の末尾に `Response` を追加したものになります。

Web メソッド名は、Web サービスの Web メソッド定義から取得されます。これは、そのメソッド名を変更しないと変更できません。

WSDL との関係

SoapMessageName キーワードは、Web サービスの WSDL の `<messages>` セクションと `<types>` セクションに影響します。例えば、以下の Web メソッドを考えてみます。

Class Member

```
Method Add(a as %Numeric,b as %Numeric) As %Numeric [ SoapMessageName=MyResponseMessage,WebMethod ]
{
    Quit a + b
}
```

この Web サービスでは、WSDL の `<types>` セクションと `<messages>` セクションは次のようになります。

```
<types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://www.mynamespace.org">
        <s:element name="Add">
            <s:complexType>
                <s:sequence>
                    <s:element minOccurs="0" name="a" type="s:decimal"/>
                    <s:element minOccurs="0" name="b" type="s:decimal"/>
                </s:sequence>
            </s:complexType>
        </s:element>
        <s:element name="MyResponseMessage">
            <s:complexType>
                <s:sequence>
                    <s:element name="AddResult" type="s:decimal"/>
                </s:sequence>
            </s:complexType>
        </s:element>
    </s:schema>
</types>
<message name="AddSoapIn">
    <part name="parameters" element="s0:Add"/>
</message>
<message name="AddSoapOut">
    <part name="parameters" element="s0:MyResponseMessage"/>
</message>
```

既定では、メソッドで SoapMessageName キーワードを指定しないと、AddSoapOut メッセージには、MyResponseMessage の代わりに AddResponse という名前の要素が含められます。

SoapMessageName は、応答メッセージの子要素 (AddResult など) には影響しません。

SOAP ウィザードを使用して WSDL から Web サービスまたはクライアントを生成する場合、その WSDL に応じてこのキーワードが設定されます。

SOAP メッセージへの影響

Web サービスは、次のような応答メッセージを送信します。

XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Body>
   <MyResponseMessage xmlns="http://www.mynamespace.org">
       <AddResult>42</AddResult>
   </MyResponseMessage>
</SOAP-ENV:Body>
```

既定では、メソッドで SoapMessageName キーワードを指定しないと、<MyResponseMessage> 要素は、代わりに <AddResponse> になります。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要
- Web サービスおよび Web クライアントの作成

SoapNameSpace (メソッド・キーワード)

Web メソッドが使用する XML ネームスペースを指定します。Web サービスまたは Web クライアントとして定義されているクラス内でのみ適用されます。

使用法

メソッドが (Web メソッドとして使用される場合に) 使用する既定の XML ネームスペースをオーバーライドするには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ SoapNameSpace = "soapnamespace", WebMethod ]
{    //implementation }
```

soapnamespace はネームスペース URI です。URI にコロン (:) が含まれる場合、文字列は引用符で囲む必要があります。つまり、次のように使用できます。

```
Method MyMethod() [ SoapNameSpace = "http://www.mynamespace.org", WebMethod ]
```

また、次のようにすることもできます。

```
Method MyMethod() [ SoapNameSpace = othervalue, WebMethod ]
```

ただし、次のようにはできません。

```
Method MyMethod() [ SoapNameSpace = http://www.mynamespace.org, WebMethod ]
```

重要　手動で作成した Web サービスでは、通常はこのキーワードの既定値が最適な値となります。SOAP ウィザードで WSDL から Web クライアントまたは Web サービスを生成すると、このキーワードはその WSDL に最適に設定されます。この値を変更すると、Web クライアントまたは Web サービスは機能しなくなる場合があります。

詳細

このキーワードは、この Web メソッドが使用する XML ネームスペースを指定します。詳細は、"Web サービスおよび Web クライアントの作成" を参照してください。

注釈　このキーワードは、メソッドが RPC スタイルのバインディングを使用する場合にのみ有効です。つまり、メソッド (またはそれを含むクラス) は、SoapBindingStyle = rpc とマークする必要があります。(ドキュメント・スタイルのバインディングを使用するメソッドでこのキーワードを指定する場合、WSDL は自己矛盾したものになります。)

既定値

このキーワードを省略すると、メソッドは、Web サービスまたはクライアント・クラスの NAMESPACE パラメータで指定されるネームスペースに配置されます。

WSDL との関係

InterSystems IRIS Web サービスのサービスでは、SoapNameSpace キーワードは <definitions> 要素内のネームスペース宣言に影響します。指定したネームスペース (http://www.customtypes.org など) は、ここに追加されます。以下に例を示します。

```
...
xmlns:ns2="http://www.customtypes.org"
xmlns:s0="http://www.wsns.org"
...
targetNamespace="http://www.wsns.org"
```

この例では、http://www.customtypes.org ネームスペースは、接頭語 ns2 に割り当てられます。

WSDL でも、通常どおり、Web サービスのネームスペース (`http://www.wsns.org`) を宣言します。この例では、このネームスペースは接頭語 `s0` に割り当てられ、ターゲットのネームスペースとしても使用されます。

SOAP メッセージへの影響

この SOAP メッセージは以下のようになります (読みやすいように、改行とスペースが追加されています)。

XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
                   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
                   xmlns:s='http://www.w3.org/2001/XMLSchema'
                   xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'
                   xmlns:tns='http://www.customtypes.org' >
   <SOAP-ENV:Body SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
      <tns:AddResponse>
         <AddResult>42</AddResult>
      </tns:AddResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

`<AddResponse>` 要素は、`http://www.webservicetypesns.org` ネームスペース内にあります。

一方、SoapNameSpace キーワードを指定しない場合、メッセージは以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
                   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
                   xmlns:s='http://www.w3.org/2001/XMLSchema'
                   xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'
                   xmlns:tns='http://www.wsns.org' >
   <SOAP-ENV:Body SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
      <tns:AddResponse>
         <AddResult>42</AddResult>
      </tns:AddResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

この場合、`<AddResponse>` 要素は Web サービスのネームスペース `http://www.wsns.org` に含まれます。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要
- Web サービスおよび Web クライアントの作成

SoapRequestMessage (メソッド・キーワード)

これは、複数の Web メソッドに同じ SoapAction が含まれている場合に使用します。既定のシナリオでは、このキーワードにより、要求メッセージの SOAP 本文内に最上位要素名を指定します。Web サービスまたは Web クライアントとして定義されているクラス内でのみ適用されます。

使用法

要求メッセージの SOAP 本文内に最上位要素名を指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ WebMethod, SoapAction = "MyAct",
SoapRequestMessage="MyReqMessage" ]
{    //implementation }
```

soaprequestmessage は有効な XML 識別子です。

詳細

注釈　　このキーワードは、折り返しありのドキュメント/リテラル・メッセージにのみ有効です。

折り返しありのドキュメント/リテラル・メッセージのために、要求メッセージの SOAP 本文内で最上位要素名をこのキーワードで指定します。(折り返しありのドキュメント/リテラル・メッセージが既定となります。詳細は、"Web サービスおよび Web クライアントの作成" の "メッセージ・バリエーションの例" を参照してください。)

同一 Web サービス内で複数の Web メソッドの SoapAction のために同じ値を使用する場合は、このキーワードを指定します。それ以外の場合は、通常はこのキーワードは必要ありません。

WSDL との関係

SoapRequestMessage キーワードは、Web サービスのための WSDL の <message> セクションに影響します。例えば、以下の Web メソッドを考えてみます。

Class Member

```
Method Add(a as %Numeric,b as %Numeric) As %Numeric [ SoapAction = MyAct,SoapRequestMessage=MyReqMessage,
 WebMethod ]
{
    Quit a + b
}
```

この Web サービスでは、WSDL には以下のものが含まれています。

```
<message name="AddSoapIn">
   <part name="parameters" element="s0:MyReqMessage"/>
</message>
<message name="AddSoapOut">
   <part name="parameters" element="s0:AddResponse"/>
</message>
```

これらの要素は、それに対応して <types> セクションで定義されます。

既定では、メソッドで SoapRequestMessage キーワードを指定しないと、<message> のセクションが次のようになる場合があります。

```
<message name="AddSoapIn">
   <part name="parameters" element="s0:Add"/>
</message>
<message name="AddSoapOut">
   <part name="parameters" element="s0:AddResponse"/>
</message>
```

SOAP ウィザードを使用して WSDL から InterSystems IRIS Web サービスのサービスまたはクライアントを生成する場合、その WSDL に応じてこのキーワードが設定されます。

メッセージへの影響

前述の Web メソッドの場合、Web サービスでは、次の形式の要求メッセージを想定しています。

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
 xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' xmlns:s='http://www.w3.org/2001/XMLSchema'>
  <SOAP-ENV:Body>
    <MyReqMessage xmlns="http://www.myapp.org"><a>1</a><b>2</b></MyReqMessage>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

その一方で、メソッドで SoapRequestMessage キーワードを指定しないと、メッセージが次のようになる場合があります。

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
 xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' xmlns:s='http://www.w3.org/2001/XMLSchema'>
  <SOAP-ENV:Body>
    <Add xmlns="http://www.myapp.org"><a>1</a><b>2</b></Add>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要
- Web サービスおよび Web クライアントの作成

SoapTypeNameSpace (メソッド・キーワード)

この Web メソッドが使用するタイプの XML ネームスペースを指定します。Web サービスまたは Web クライアントとして定義されているクラス内でのみ適用されます。

使用法

(メソッドが Web メソッドとして使用される場合の) タイプの既定の XML ネームスペースをオーバーライドするには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ SoapTypeNameSpace = "soapnamespace", SoapBindingStyle =
document, WebMethod ]
{    //implementation }
```

soapnamespace はネームスペース URI です。URI にコロン (:) が含まれる場合、文字列は引用符で囲む必要があります。つまり、次のように使用できます。

```
Method MyMethod() [ SoapTypeNameSpace = "http://www.mynamespace.org", SoapBindingStyle = document,
WebMethod ]
```

また、次のようにすることもできます。

```
Method MyMethod() [ SoapTypeNameSpace = othervalue, SoapBindingStyle = document, WebMethod ]
```

ただし、次のようにはできません。

```
Method MyMethod() [ SoapTypeNameSpace = http://www.mynamespace.org, SoapBindingStyle = document,
WebMethod ]
```

重要　手動で作成した Web サービスでは、通常はこのキーワードの既定値が最適な値となります。SOAP ウィザードで WSDL から Web クライアントまたは Web サービスを生成すると、このキーワードはその WSDL に最適に設定されます。この値を変更すると、Web クライアントまたは Web サービスは機能しなくなる場合があります。

詳細

このキーワードは、この Web メソッドが使用するタイプの XML ネームスペースを指定します。詳細は、"Web サービスおよび Web クライアントの作成" を参照してください。

注釈　このキーワードは、メソッドがドキュメント・スタイルのバインディングを使用する場合にのみ有効です。つまり、メソッド (またはそれを含むクラス) は、SoapBindingStyle = `document` とマークする必要があります。(RPC スタイルのバインディングを使用するメソッドでこのキーワードを指定しても意味がありません。)

既定値

このキーワードを省略すると、このメソッドのタイプは、Web サービスまたはクライアント・クラスの TYPENAMESPACE パラメータで指定されるネームスペースに配置されます。TYPENAMESPACE が指定されない場合、タイプは、Web サービスまたはクライアントの NAMESPACE パラメータで指定されるネームスペースに配置されます。

WSDL との関係

SoapTypeNameSpace キーワードは、WSDL の次のパートに影響します。

- <definitions> 要素内のネームスペース宣言。指定したネームスペース (http://www.customtypes.org など) は、ここに追加されます。以下に例を示します。

```
...
xmlns:ns2="http://www.customtypes.org"
xmlns:s0="http://www.wbns.org"
xmlns:s1="http://webservicetypesns.org"
...
targetNamespace="http://www.wbns.org"
```

この例では、http://www.customtypes.org ネームスペースは、接頭語 ns2 に割り当てられます。

WSDL では、通常どおり、次のネームスペースも宣言します。

- Web サービスのネームスペース (http://www.wsns.org)。この例では、これは接頭語 s0 に割り当てられ、Web サービスのターゲットのネームスペースとしても使用されます。
- Web サービスのタイプのネームスペース (http://www.webservicetypesns.org)。この例では、これは接頭語 s1 に割り当てられます。

  Web サービス・クラスでタイプのネームスペースが指定されない場合、このネームスペースは WSDL に含まれません。

- <types> 要素。これは、targetNamespace 属性が、SoapTypeNameSpace で指定されたネームスペースと等しい <schema> 要素を含みます。

```
<types>
...
<s:schema elementFormDefault="qualified" targetNamespace="http://www.customtypes.org">
    <s:element name="Add">
        <s:complexType>
            <s:sequence>
                <s:element minOccurs="0" name="a" type="s:decimal"/>
                <s:element minOccurs="0" name="b" type="s:decimal"/>
            </s:sequence>
        </s:complexType>
    </s:element>
    <s:element name="AddResponse">
        <s:complexType>
            <s:sequence>
                <s:element name="AddResult" type="s:decimal"/>
            </s:sequence>
        </s:complexType>
    </s:element>
</s:schema>

...
</types>
```

一方、SoapTypeNameSpace を指定しない場合、WSDL のこのパートは次のようになります。<schema> 要素の targetNamespace は、Web サービスのタイプのネームスペースです。

```
<types>
...
<s:schema elementFormDefault="qualified" targetNamespace="http://www.webservicetypesns.org">
    <s:element name="Add">
        <s:complexType>
            <s:sequence>
                <s:element minOccurs="0" name="a" type="s:decimal"/>
                <s:element minOccurs="0" name="b" type="s:decimal"/>
            </s:sequence>
        </s:complexType>
    </s:element>
    <s:element name="AddResponse">
        <s:complexType>
            <s:sequence>
                <s:element name="AddResult" type="s:decimal"/>
            </s:sequence>
        </s:complexType>
    </s:element>
</s:schema>
```

```
...
</types>
```

(また、Web サービス・クラスでタイプのネームスペースが指定されない場合、`targetNamespace` は Web サービスのネームスペースになります。)

メッセージへの影響

この SOAP メッセージは以下のようになります (読みやすいように、改行とスペースが追加されています)。

XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
                   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
                   xmlns:s='http://www.w3.org/2001/XMLSchema'>
   <SOAP-ENV:Body>
      <AddResponse xmlns="http://www.customtypes.org">
         <AddResult>3</AddResult>
      </AddResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

`<AddResponse>` 要素は、`"http://www.customtypes.org"` ネームスペース内にあります。

一方、SoapTypeNameSpace キーワードを指定しない場合、メッセージは以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
                   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
                   xmlns:s='http://www.w3.org/2001/XMLSchema'>
   <SOAP-ENV:Body>
      <AddResponse xmlns="http://www.webservicetypesns.org">
         <AddResult>3</AddResult>
      </AddResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要
- Web サービスおよび Web クライアントの作成

SqlName (メソッド・キーワード)

投影された SQL ストアド・プロシージャの既定の名前をオーバーライドします。メソッドが SLQ ストアド・プロシージャとして投影される場合にのみ適用されます。

使用法

メソッドが SLQ ストアド・プロシージャとして投影されるときに使用される既定の名前をオーバーライドするには、以下の構文を使用します。

```
ClassMethod name(formal_spec) As returnclass [ SqlProc, SqlName = sqlname ]
{    //implementation }
```

sqlname は SQL 識別子です。

詳細

このメソッドを SQL ストアド・プロシージャとして投影するときに、この名前をストアド・プロシージャの名前に使用します。

既定値

このキーワードを省略すると、InterSystems IRIS は SQL 名を以下のように決定します。

```
CLASSNAME_METHODNAME
```

この既定では大文字を使用します。ストアド・プロシージャを呼び出す際には大文字も小文字も使用できますが、SQL では大文字と小文字は区別されないためです。

つまり、以下の例では、既定の SQL 名の値は `TEST1_PROC1` となります。この既定値は SELECT 文で指定します。

Class Definition

```
Class User.Test1 Extends %Persistent
{
ClassMethod Proc1(BO,SUM) As %INTEGER [ SqlProc ]
{
   ///definition not shown
}

Query Q1(KD As %String,P1 As %String,P2 As %String) As %SqlQuery
{
  SELECT SUM(SQLUser.TEST1_PROC1(1,2)) AS Sumd
  FROM SQLUser.Test1
}
}
```

関連項目

- メソッド定義
- SqlProc キーワード
- ストアド・プロシージャの定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

SqlProc (メソッド・キーワード)

メソッドを SQL ストアド・プロシージャとして呼び出すことができるかどうかを指定します。SQL ストアド・プロシージャとして呼び出せるのはクラス・メソッドのみです (インスタンス・メソッドではありません)。

使用法

メソッドを SQL ストアド・プロシージャとして呼び出すことができると指定するには、以下の構文を使用します。

```
ClassMethod name(formal_spec) As returnclass [ SqlProc ]
{    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

このキーワードは、メソッドを SQL ストアド・プロシージャとして呼び出すことができることを指定します。SQL ストアド・プロシージャとして呼び出せるのはクラス・メソッドのみです (インスタンス・メソッドではありません)。

ストアド・プロシージャはサブクラスに継承されます。

既定値

このキーワードを省略すると、メソッドは SQL ストアド・プロシージャとして使用できません。

関連項目

- メソッド定義
- ストアド・プロシージャの定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要

WebMethod (メソッド・キーワード)

メソッドが Web メソッドであるかどうかを指定します。Web サービスまたは Web クライアントとして定義されているクラス内でのみ適用されます。

使用法

このメソッドを Web メソッドとして指定するには、以下の構文を使用します。

```
Method name(formal_spec) As returnclass [ WebMethod ]
{    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

このキーワードは、このメソッドが Web メソッドとして使用可能であり、SOAP プロトコル経由で呼び出しが可能であることを指定します。

重要　ほとんどの場合、Web メソッドは、クラス・メソッドではなくインスタンス・メソッドにする必要があります。Web メソッドの詳細および他の要件については、"Web サービスおよび Web クライアントの作成" を参照してください。

既定値

このキーワードを省略すると、メソッドは Web メソッドとして使用できません。

生成されたクラス

このキーワードをメソッドに追加してクラスをコンパイルすると、クラス・コンパイラは、もう 1 つのクラス Package.OriginalClass.MethodName を生成します。ここで、Package.OriginalClass は Web メソッドを含むクラスであり、MethodName は Web メソッドの名前です。

例えば、クラス `ROBJDemo.DocLiteralWS` で開始し、それに `Add` という名前のメソッドを追加したとします。そのメソッドに WebMethod キーワードを追加してコンパイルすると、クラス・コンパイラは、クラス `ROBJDemo.DocLiteralWS.Add` を生成します。

この生成されたクラスを変更したり、直接使用したりしないでください。これは内部で使用するために設計されたものです。

WSDL との関係

Web サービスでは、このキーワードは、生成される WSDL にも影響します。ここには、この Web メソッドを示すのに必要な追加要素が含まれるようになります。

関連項目

- メソッド定義
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- コンパイラ・キーワードの概要
- Web サービスおよび Web クライアントの作成

パラメータの構文とキーワード

このリファレンスでは、クラス・パラメータに適用する構文とキーワードについて説明しています。キーワード（クラス属性とも呼ばれる）は、一般にコンパイラに影響を与えます。

パラメータ定義に関する一般情報へのリンクは、"パラメータ定義" を参照してください。

特定のクラス・メンバに適用するすべての構造とキーワードは、目次を参照してください。

クラス定義でのパラメータの構文

パラメータ定義の構造を説明します。パラメータ定義に関する一般情報へのリンクは、"パラメータ定義" を参照してください。

概要

パラメータ定義では、特定のクラスのすべてのオブジェクトで使用できる定数値を定義します。このクラス・パラメータの値は、クラス定義を作成するときに (またはコンパイル前の任意の時点で) 設定できます。既定では、各パラメータの値は NULL 文字列になりますが、パラメータ定義の一環として NULL 以外の値を指定することもできます。コンパイル時に、パラメータの値はクラスのすべてのインスタンスに対して構築されます。わずかな例外はありますが、この値を実行時に変更することはできません。

詳細

パラメータ定義の構造は以下のとおりです。

```
/// description
Parameter name As parameter_type [ keyword_list ] = value ;
```

以下は、この指定の説明です。

- description (オプション) は、クラス・リファレンスでの表示を意図しています。description は既定ではブランクです。"クラス・ドキュメントの作成" を参照してください。
- name (必須) はパラメータの名前です。これは、有効なクラス・メンバ名で、かつ他のクラス・メンバ名と重複しないものである必要があります。
- parameter_type (オプション) は、パラメータのユーザ・インタフェース・タイプを指定し、スタジオによってインスペクタ内でのパラメータの入力検証に使用されます。

  これはクラス名ではありません。次のセクションを参照してください。ほとんどの場合、コンパイラはこのキーワードを無視します。

  parameter_type を省略する場合は、単語 As も省略します。
- value (オプション) は、パラメータの値を指定します。value を省略する場合は、等号 = も省略します。
- keyword_list (オプション) は、さらにパラメータを定義するキーワードのコンマ区切りリストです。

  すべてのキーワードは、"パラメータの構文とキーワード" を参照してください。

  このリストを省略する場合は角括弧も省略します。

パラメータに許可されるタイプ

parameter_type オプションは、以下のいずれかの値のとなります。

- BOOLEAN — True (1) または False (0)
- CLASSNAME — 有効なクラス名
- COSCODE — ObjectScript コード
- COSEXPRESSION — 有効な ObjectScript 式。

  パラメータのタイプが COSEXPRESSION である場合は、式は実行時に評価されます。

  この値は、パラメータの Type キーワードのその他ほとんどの値とは異なり、コンパイラに影響を与えます。
- COSIDENTIFIER — 有効な ObjectScript 識別子

- INTEGER ― 整数
- SQL ― SQL 文
- SQLIDENTIFIER ― 有効な SQL 識別子
- STRING ― 文字列値
- TEXT ― 複数行のテキスト値
- CONFIGVALUE ― クラス定義の範囲外で変更可能なパラメータ。この値は、パラメータの Type キーワードのその他ほとんどの値とは異なり、コンパイラに影響を与えます。パラメータのタイプが CONFIGVALUE の場合は、$SYSTEM.OBJ.UpdateConfigParam() を使用してパラメータを変更できます。例えば、以下のようにして、MyApp.MyClass クラス内のパラメータ MYPARM の値を新しく 42 に変更します。

  ```
  set sc=$system.OBJ.UpdateConfigParam("MyApp.MyClass","MYPARM",42)
  ```

  $SYSTEM.OBJ.UpdateConfigParam() は、新しいプロセスで使用される生成済みのクラス記述子に影響しますが、クラス定義には影響しません。クラスをリコンパイルすると、InterSystems IRIS により、クラス記述子が再生成され、クラス定義に記載されたようにこのパラメータの値を使用します (そのため、$SYSTEM.OBJ.UpdateConfigParam() によって行った変更は上書きされます)。

parameter_type は省略することもできます。その場合は、インスペクタによってそのパラメータに任意の値を使用できるようになります。

例

```
/// This is the name of our web service.
Parameter SERVICENAME = "SOAPDemo" ;
```

関連項目

- クラス・パラメータの定義と参照
- クラスの制限

Abstract (パラメータ・キーワード)

Abstract パラメータであるかどうかを指定します。

使用法

このパラメータを Abstract として指定するには、以下の構文を使用します。

```
Parameter name As parameter_type [ Abstract ] = value ;
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

抽象パラメータの動作は、何も定義されていないかのように単純です。しかし、ユーザは、抽象パラメータを定義することでこれを明記し、このパラメータのシグニチャをサブクラス内で定義することを強制できます。

既定値

このキーワードを省略すると、パラメータは Abstract ではなくなります。

関連項目

- パラメータ定義
- クラス・パラメータの定義と参照
- コンパイラ・キーワードの概要

Constraint (パラメータ・キーワード)

このパラメータのスタジオ内でのユーザ・インタフェースの制約を指定します。

使用法

このプロパティのユーザ・インタフェースの制約を指定するには、以下の構文を使用します。

```
Parameter name As parameter_type [ Constraint = "constraint" ]  = value ;
```

constraint はスタジオにより使用される文字列です。

詳細

スタジオ・インスペクタでは、パラメータの入力検証にこの制約の値が使用されます。その値は、クラス・コンパイラによって使用も強制もされません。

このキーワードは、Flags キーワードと連動します。例えば、Flags が ENUM に設定されている場合、Constraint は使用可能なパラメータ値のコンマで区切られたリストです。

例

```
Parameter MYPARM [ Constraint = "X,Y,Z", Flags = ENUM ] = "X";
```

関連項目

- パラメータ定義
- クラス・パラメータの定義と参照
- コンパイラ・キーワードの概要

Deprecated (パラメータ・キーワード)

このパラメータを非推奨として指定します。このキーワードはクラス・コンパイラでは無視され、単にパラメータが非推奨であることを人間が読める形で示します。

使用法

このパラメータを非推奨として指定するには、以下の構文を使用します。

```
Parameter name As parameter_type  [ Deprecated ] = value;
```

そうしない場合は、このキーワードを省略するか、キーワードの直前に単語 Not を配置します。

関連項目

- パラメータ定義

Final (パラメータ・キーワード)

このパラメータが Final である (サブクラス内でオーバーライドできない) ことを指定します。

使用法

パラメータを Final として指定するには、以下の構文を使用します。

```
Parameter name As parameter_type  [ Final ] = value;
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

final としてマークされるクラス・メンバは、サブクラスではオーバーライドできません。

既定値

このキーワードを省略すると、パラメータは Final ではなくなります。

関連項目

- パラメータ定義
- クラス・パラメータの定義と参照
- コンパイラ・キーワードの概要

Flags (パラメータ・キーワード)

このパラメータの (スタジオ内での) ユーザ・インタフェースのタイプを変更します。

使用法

このプロパティの (スタジオ内での) ユーザ・インタフェースのタイプを変更するには、以下の構文を使用します。

```
Parameter name As parameter_type  [ Flags = flags ] = value;
```

flags は以下のいずれかになります。

- ENUM ー このパラメータは、Constraint キーワードで指定された (コンマで区切られたリストの) 値のうちの 1 つです。インスペクタでは、これらの値のドロップダウン・リストが提供されます。
- LIST ー このパラメータ値は、コンマで区切られた項目のリストから成る文字列です。

EDIT、EMPTY、および SYS は使用されません。

詳細

このパラメータの (スタジオ内での) ユーザ・インタフェースのタイプを変更します。スタジオでは、インスペクタ内でのパラメータの入力検証にこのタイプが使用されます。クラス・コンパイラでは、このキーワードは無視されます。

既定値

このキーワードを省略すると、スタジオはパラメータに単一の値のみを許可します (選択のためのドロップダウン・リストは提供しません)。

関連項目

- パラメータ定義
- クラス・パラメータの定義と参照
- コンパイラ・キーワードの概要

Internal (パラメータ・キーワード)

このパラメータ定義が Internal (クラス・ドキュメントに表示されない) であるかどうかを指定します。

使用法

このパラメータを Internal として指定するには、以下の構文を使用します。

```
Parameter name As parameter_type  [ Internal ] = value;
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

Internal クラスのメンバは、クラス・ドキュメントには表示されません。このキーワードは、ユーザに対してクラスは表示するが、そのすべてのメンバを表示する必要がない場合に役立ちます。

既定値

このキーワードを省略すると、このパラメータはクラス・ドキュメントに表示されます。

関連項目

- パラメータ定義
- クラス・パラメータの定義と参照
- コンパイラ・キーワードの概要

投影の構文とキーワード

このリファレンスでは、クラス投影に適用する構文とキーワードについて説明しています。キーワード（クラス属性とも呼ばれる）は、一般にコンパイラに影響を与えます。

投影定義に関する一般情報へのリンクは、"投影定義" を参照してください。

特定のクラス・メンバに適用するすべての構造とキーワードは、目次を参照してください。

クラス定義での投影の構文

プロジェクション定義の構造を説明します。投影定義に関する一般情報へのリンクは、"投影定義" を参照してください。

概要

投影定義は、クラス定義がコンパイルまたは削除されたときに、指定された操作の実行をクラス・コンパイラに指示します。プロジェクションは、(%Projection.AbstractProjection クラスから派生した) プロジェクション・クラスの名前を定義します。プロジェクション・クラスは (クラス定義の削除か、クラスの再コンパイルのいずれかによる)、クラスのコンパイル終了時と、クラス定義の削除時に呼び出されるメソッドを実装します。

詳細

プロジェクション定義の構造は以下のとおりです。

```
/// description
Projection name As projection_class (parameter_list) ;
```

以下は、この指定の説明です。

- description (オプション) は、クラス・リファレンスでの表示を意図しています (ただしプロジェクションは、現在はクラス・リファレンスに表示されていないことに注意してください)。description は既定ではブランクです。"クラス・ドキュメントの作成" を参照してください。
- name (必須) はプロジェクションの名前です。これは、有効なクラス・メンバ名で、かつ他のクラス・メンバ名と重複しないものである必要があります。
- projection_class (必須) は、プロジェクション・クラスの名前である必要があります。これは、%Projection.AbstractProjection のサブクラスです。
- parameter_list (オプション) は、パラメータとその値の、コンマ区切りのリストです。指定した場合、それらは projection_class が使用するパラメータになります。

  このリストを省略する場合は、括弧も省略します。
- keyword_list (オプション) は、さらにプロジェクションを定義するキーワードのコンマ区切りリストです。

  すべてのキーワードは、"投影の構文とキーワード" を参照してください。

  このリストを省略する場合は角括弧も省略します。

関連項目

- クラス・プロジェクションの定義
- クラスの制限

Internal (投影キーワード)

このプロジェクション定義が Internal (クラス・ドキュメントに表示されない) であるかどうかを指定します。クラス・ドキュメントは、現在はプロジェクションをまったく表示しません。

使用法

このプロジェクションを Internal として指定するには、以下の構文を使用します。

```
Projection projectionname As class [ Internal ];
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 Not を配置します。

詳細

Internal クラスのメンバは、クラス・ドキュメントには表示されません。このキーワードは、ユーザに対してクラスは表示するが、そのすべてのメンバを表示する必要がない場合に役立ちます。

クラス・ドキュメントは、現在はプロジェクションをまったく表示しません。

既定値

このキーワードを省略すると、プロジェクションは Internal ではなくなります。

関連項目

- 投影定義
- クラス・プロジェクションの定義
- コンパイラ・キーワードの概要

プロパティの構文とキーワード

このリファレンスでは、オブジェクト・クラス内で定義できるプロパティに適用する構文とキーワードを説明します。キーワード (クラス属性とも呼ばれる) は、一般にコンパイラに影響を与えます。

プロパティ定義に関する一般情報へのリンクは、"プロパティ定義" を参照してください。

特定のクラス・メンバに適用するすべての構造とキーワードは、目次を参照してください。

クラス定義でのプロパティの構文

プロパティ定義の構造を説明します。リレーションシップはプロパティです。プロパティ定義に関する一般情報へのリンクは、"プロパティ定義" を参照してください。

概要

プロパティには、クラスのインスタンスに関係する情報が記述されます。プロパティ定義はオブジェクト・クラスに追加できます。他のクラスでは意味がありません。

詳細

プロパティ定義の構造は以下のとおりです。

```
/// description
Property name As classname (parameter_list) [ keyword_list ] ;
```

または (リスト・プロパティでは) 以下のとおりです。

```
/// description
Property name As List Of classname (parameter_list) [ keyword_list ] ;
```

または (配列プロパティでは) 以下のとおりです。

```
/// description
Property name As Array Of classname (parameter_list) [ keyword_list ] ;
```

または (リレーションシップ・プロパティでは) 以下のとおりです。

```
/// description
Relationship name As classname [ keyword_list ] ;
```

以下は、この指定の説明です。

- description (オプション) は、クラス・リファレンスでの表示を意図しています。description は既定ではブランクです。"クラス・ドキュメントの作成" を参照してください。
- name (必須) はプロパティの名前です。これは、有効なクラス・メンバ名で、かつ他のクラス・メンバ名と重複しないものである必要があります。
- classname (オプション) は、このプロパティが基づくクラスの名前です。
- parameter_list (オプション) は、パラメータとその値の、コンマ区切りのリストです。指定する場合、それらは classname が使用するパラメータ、またはすべてのプロパティが使用できるパラメータのいずれかにする必要があります。

  このリストを省略する場合は、括弧も省略します。
- keyword_list (リレーションシップ・プロパティでは必須、その他のプロパティではオプション) は、さらにプロパティを定義するキーワードのコンマ区切りリストです。

  すべてのキーワードは、"プロパティの構文とキーワード" を参照してください。

  このリストを省略する場合は角括弧も省略します。

注釈　　リレーションシップは、シャード・クラスではサポートされていません。

例

```
/// Person's Social Security number.
Property SSN As %String(PATTERN = "3N1""-""2N1""-""4N") [ Required  ];
```

関連項目

- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- クラスの制限

Aliases (プロパティ・キーワード)

オブジェクト・アクセスにより使用するための、このプロパティの追加名を指定します。

使用法

プロパティの追加名を指定するには、以下の構文を使用します。

```
Property name As classname [ Aliases=othernames ];
```

othernames は、中括弧で囲まれた、有効なプロパティ名のコンマ区切りリストです。

詳細

Aliases キーワードを指定する場合、コンパイラは元のプロパティが指しているのと同じ基本データを指す、特定のエイリアスを作成します。例えば、以下のように Sample.Person の Name プロパティを再定義するとします。

```
Property Name As %String(POPSPEC = "Name()") [ Aliases = {Alternate}, Required ];
```

その場合、以下のターミナル・セッションで示すとおり、コードは Name プロパティまたは同等の Alternate プロパティのいずれかで機能します。

```
SAMPLES>set p=##class(Sample.Person).%OpenId(1)

SAMPLES>w p.Name
Fripp,Charles Z.
SAMPLES>w p.Alternate
Fripp,Charles Z.
SAMPLES>set p.Alternate="Anderson,Neville J."

SAMPLES>w p.Name
Anderson,Neville J.
```

元のプロパティと関連付けられているすべてのプロパティも、各エイリアス・プロパティに対して定義されます。そのためこの例では、AlternateIsValid() は呼び出し可能であり、NameIsValid() メソッドの場合と同じ結果を返します。さらに、プロパティ・メソッドを (例えば、カスタム NameGet() メソッドを記述するなどして) オーバーライドする場合、そのオーバーライドはエイリアスのプロパティに自動的に適用されます。

注釈　　このキーワードは、プロパティの SQL プロジェクションには影響を与えません。

既定値

既定では、このキーワードは NULL であり、プロパティにエイリアスはありません。

例

```
Property PropA As %String [ Aliases={OtherName,OtherName2} ];
```

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用

- コンパイラ・キーワードの概要

Calculated (プロパティ・キーワード)

このプロパティを含むオブジェクトがインスタンス化される際に、プロパティにメモリ内ストレージを割り当てないことを指定します。

使用法

プロパティにメモリ内ストレージを割り当てないことを指定するには、以下の構文を使用します。

```
Property name As classname [ Calculated ];
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 Not を指定します。

詳細

このキーワードは、このプロパティを含むオブジェクトがインスタンス化される際に、プロパティにメモリ内ストレージを割り当てないことを指定します。

メモリ内ストレージが必要ないプロパティを定義するときに、このキーワードを使用します。このプロパティに値を指定する方法には、次の 2 つがあります。

- プロパティのための Get (場合によっては、Set) メソッドを定義します。例えば、Age プロパティの場合は、AgeGet メソッドを提供することで、ある人物の現在の年齢を、現在日時、およびその人物の DateOfBirth プロパティの値に基づいて判断できます。"プロパティ・メソッドの使用とオーバーライド" を参照してください。
- このプロパティを計算プロパティとして定義します。これにより、SqlComputed キーワードおよび関連キーワードが使用されます。"計算プロパティの定義" を参照してください。

サブクラスは Calculated キーワードを継承しますが、それをオーバーライドできません。

SQL テーブルのインデックスに、Calculated キーワードを持つプロパティを含めるには、SqlComputed キーワードと SqlComputeCode キーワードも指定する必要があります。Calculated プロパティでインデックスを持つクラスをコンパイルする場合、このプロパティに SqlComputed キーワードと SqlComputeCode キーワードも指定しないと、システムは次のエラー・メッセージを返します。

```
ERROR #5414: Invalid index attribute: <classname>::<indexname>::<propertyname>
ERROR #5030: An error occurred while compiling class <classname>
```

既定値

Calculated キーワードの既定値は、False です。

例

```
Property Age as %Integer [ Calculated ];
```

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用

- コンパイラ・キーワードの概要

Cardinality (プロパティ・キーワード)

このリレーションシップ・プロパティのカーディナリティを指定します。リレーションシップ・プロパティには必須です。他のプロパティには使用しません。

使用法

リレーションシップ・プロパティのカーディナリティを指定するには、以下の構文を使用します。

```
Relationship relname As classname [ Cardinality = cardinality; inverse = inverse ];
```

cardinality は以下のいずれかになります。

- `one`
- `many`
- `parent`
- `children`

詳細

このキーワードは、リレーションシップ・プロパティのカーディナリティを指定します。

Cardinality キーワードは、リレーションシップ・プロパティに必須です。リレーションシップ・プロパティ以外のプロパティには無視されます。

リレーションシップの詳細は、"リレーションシップの定義と使用" を参照してください。

既定値

既定値はありません。リレーションシップを定義する場合は、Cardinality キーワードを指定する必要があります。

例

```
Relationship Chapters As Chapter [ Cardinality = many; inverse = Book ];
```

関連項目

- プロパティ定義
- Inverse キーワード
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

ClientName (プロパティ・キーワード)

このプロパティのクライアント・プロジェクションに使用するエイリアスを指定します。

使用法

クラスがクライアント言語に投影されるときのこのプロパティの既定の名前をオーバーライドするには、以下の構文を使用します。

```
Property name As classname [ ClientName = clientname ];
```

clientname はクライアント言語で使用する名前です。

詳細

このキーワードにより、プロパティにクライアントの言語を投影するときに、プロパティに別名を定義できます。このキーワードは、プロパティ名にクライアント言語では許可されない文字が含まれている場合に特に役立ちます。

既定値

このキーワードを省略すると、プロパティ名がクライアント名として使用されます。

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

Collection (プロパティ・キーワード)

コレクション・プロパティのコレクション・タイプを指定する非推奨の方法です。使用しません。

詳細

このキーワードは、"コレクションを使用した作業" で説明されている “As” 構文で置き換えられました。

関連項目

- プロパティ定義
- コレクションを使用した作業
- コンパイラ・キーワードの概要

ComputeLocalOnly (プロパティ・キーワード)

SqlComputed フィールドを、フェデレートされたテーブルおよびシャード・テーブルのローカル・サーバのみで計算するかどうかを制御します。

詳細

値が 1 の場合、SqlComputeCode は、データが存在するサーバでのみ実行されます。その結果、シャードまたはフェデレートされた環境では、計算されたデータはクエリの発行元のシステムからのみ返されます。SqlComputeCode は、他のシャードでは計算されません。

値が 0 の場合、SqlComputeCode はどのサーバでも実行できます。

既定値

既定値は 0 です。

関連項目

- SqlComputeCode
- SqlComputed
- 計算プロパティの定義

Deprecated (プロパティ・キーワード)

このプロパティを非推奨として指定します。このキーワードはクラス・コンパイラでは無視され、単にプロパティが非推奨であることを人間が読める形で示します。

使用法

このプロパティを非推奨として指定するには、以下の構文を使用します。

```
Property name As classname [ Deprecated ];
```

そうしない場合は、このキーワードを省略するか、キーワードの直前に単語 Not を配置します。

関連項目

- プロパティ定義

Final (プロパティ・キーワード)

このプロパティが Final である (サブクラス内でオーバーライドできない) ことを指定します。

使用法

プロパティを Final として指定するには、以下の構文を使用します。

```
Property name As classname [ Final ];
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

final としてマークされるクラス・メンバは、サブクラスではオーバーライドできません。

既定値

このキーワードを省略すると、プロパティは Final ではなくなります。

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

Identity (プロパティ・キーワード)

このプロパティが、対応する SQL テーブルの ID 列に対応するかどうかを指定します。永続クラスに適用されます。

使用法

このプロパティが、対応する SQL テーブルの ID 列に対応するかどうかを指定するには、以下の構文を使用します。

```
Property name As %Integer [ Identity ];
```

注釈　　プロパティのタイプは、**%BigInt**、**%Integer**、**%SmallInt**、**%TinyInt** など、任意の整数のタイプにできます。

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

永続オブジェクトの場合、このキーワードは、プロパティが、対応する SQL テーブル内の ID 列 (SQL IDENTITY キーワードでマークされている列) に対応することを指定します。このキーワードは、DDL 文を使用して作成されるテーブルの場合に特に役立ちます。DDL を使用してテーブルを作成する際は、ビットマップ・エクステント・インデックスを作成できるように、可能であれば `MINVAL=1` を指定して IDENTITY フィールドを定義するようにしてください。"CREATE TABLE" を参照してください。

既定値

このキーワードを省略すると、このプロパティは ID 列として使用されません。

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コンパイラ・キーワードの概要

InitialExpression (プロパティ・キーワード)

このプロパティの初期値を指定します。

使用法

このプロパティの初期値を指定するには、以下の構文を使用します。

```
Property name As classname [ InitialExpression = initialexpression ];
```

initialexpression は、中括弧で囲まれた、定数または ObjectScript 式です。

詳細

このキーワードは、プロパティの初期値を指定します。新規のインスタンスが生成される際に、そのクラスの %New() メソッドが、この値を割り当てます (プロパティが一時的な場合、その初期値は、インスタンスが生成される際に %New() によって実行されるコード、または、インスタンスがディスクからメモリにロードされる際に %OpenId() によって実行されるコードのいずれかによって決定されます)。

初期値式の値は、特定のプロパティ・タイプに適したものとする必要があります。

どのような複雑な式でも指定できますが、以下の制限があります。

- 初期値式は他のプロパティを参照できません。つまり、`{..otherpropertyname}` のような式は無効です。
- 初期値式はオブジェクトをインスタンス化できず、オブジェクト参照を含むことができません。
- 初期値式は、インスタンス・メソッドを呼び出すことができません (クラス・メソッドのみ)。
- 初期表現は ObjectScript で指定しなければなりません。
- 式によって実行されたコードではエラーをレポートしないでください。InterSystems IRIS には、式から返されたエラーを処理する方法は用意されていません。
- 式から実行されたコードに起因して他の処理が発生した場合、InterSystems IRIS には、その処理結果に対処する方法は用意されていません。

サブクラスは InitialExpression キーワードの値を継承し、それをオーバーライドできます。

既定値

InitialExpression キーワードの既定値は、NULL です。

例

以下の例は、ObjectScript 式の使用例です。

```
Property DateTime As %Date [ InitialExpression = {$zdateh("1966-10-28",3)} ];

Property MyString As %String [ InitialExpression = {$char(0)} ];

/// this one is initialized with the value of a parameter
Property MyProp As %String [ InitialExpression = {..#MYPARM} ];

/// this one is initialized by a class method
Property MyProp2 As %Numeric [ InitialExpression = {..Initialize()} ];
```

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業

- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

Internal (プロパティ・キーワード)

このプロパティ定義が Internal (クラス・ドキュメントに表示されない) であるかどうかを指定します。.

使用法

このプロパティを Internal として指定するには、以下の構文を使用します。

```
Property propertyname As classname [ Internal ];
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

Internal クラスのメンバは、クラス・ドキュメントには表示されません。このキーワードは、ユーザに対してクラスは表示するが、そのすべてのメンバを表示する必要がない場合に役立ちます。

既定値

このキーワードを省略すると、このプロパティはクラス・ドキュメントに表示されます。

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

Inverse (プロパティ・キーワード)

このリレーションシップの逆側を指定します。リレーションシップ・プロパティには必須です。他のプロパティには使用しません。

使用法

このリレーションシップ・プロパティの逆側である、関連するクラス内のリレーションシップ・プロパティを指定するには、以下の構文を使用します。

```
Relationship Chapters As Chapter [ Cardinality = cardinality; Inverse = inverse ];
```

inverse は、関連するクラス内のプロパティ名です。

詳細

このキーワードは、リレーションシップの逆側の名前を指定します。つまり、関連するクラス内の対応するリレーションシップ・プロパティの名前です。逆プロパティは関連するクラス内に存在し、正しい Cardinality 値を持つ必要があります。

Inverse キーワードは、リレーションシップ・プロパティに必須です。リレーションシップ・プロパティ以外のプロパティには無視されます。

詳細は、"リレーションシップの定義と使用" を参照してください。

既定値

既定値はありません。リレーションシップを定義する場合は、Inverse キーワードを指定する必要があります。

例

```
Relationship Chapters As Chapter [ Cardinality = many; inverse = Book ];
```

関連項目

- プロパティ定義
- Cardinality キーワード
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

MultiDimensional (プロパティ・キーワード)

このプロパティに多次元配列の特性を指定します。

使用法

このプロパティに多次元配列の特性を指定するには、以下の構文を使用します。

```
Property Data [ Multidimensional ];
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 Not を指定します。

詳細

多次元プロパティは、以下の点で他のプロパティとは異なります。

- InterSystems IRIS はこれに対するプロパティ・メソッドは提供しません (プロパティ・メソッドについては、"クラスの定義と使用" を参照してください)。
- オブジェクトを検証または保存するときに、無視されます。
- アプリケーションにそれを明示的に保存するコードが含まれていない場合、ディスクに保存されません。

  つまり、プロパティは自動的に Tarnsient でもあります。
- Java や他のクライアントに公開することはできません。
- SQL テーブルに格納できず、SQL テーブルでも公開されません。

多次元プロパティはあまり使用されませんが、オブジェクトの状態に関する情報を一時的に格納する場合に有用な方法を提供します。

既定値

このキーワードを省略すると、プロパティは多次元ではなくなります。

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

OnDelete (プロパティ・キーワード)

関連するオブジェクトが削除されたときに、現在のテーブルで実行するアクションを指定します。このキーワードは、Cardinality を Parent または One として指定するリレーションシップ・プロパティにのみ適用されます。その他すべてのコンテキストでの使用は無効です。

使用法

関連するオブジェクトが削除されたときに、現在のテーブルで実行するアクションを指定するには、以下の構文を使用します。

```
Relationship relname As classname [ Cardinality = cardinality, Inverse = inverse, OnDelete = ondelete
 ];
```

ondelete は以下のいずれかの値になります。この説明では、関連するレコードはリレーションシップの相手側に属するレコードまたはオブジェクトであり、参照元レコードはリレーションシップのこちら側にあるレコードまたはオブジェクトです。

- cascade ― 関連するレコードを削除した場合、このテーブル内の参照元レコードも削除されます。
- noaction ― 関連するレコードの削除を試みた場合、失敗します。
- setdefault ― 関連するレコードを削除した場合、このテーブル内の参照元レコードはその既定値に設定されます。
- setnull ― 関連するレコードを削除した場合、このテーブル内の参照元レコードは NULL に設定されます。

詳細

このキーワードは、リレーションシップの相手側でレコードが削除されたときに発生する参照アクションを定義します。

既定値

このキーワードを省略すると、以下のようになります。

- Parent である Cardinality とのリレーションシップの場合、OnDelete は cascade です。つまり、親レコードを削除する場合、既定では、関連する子レコードが削除されます。
- One である Cardinality とのリレーションシップの場合、OnDelete は noaction です。つまり、1 レコードの削除を試みた場合、既定ではそれを指すレコードが相手のテーブルにあれば失敗します。

例

```
Class MyApp.Employee Extends %Persistent {
...
Relationship Employer As MyApp.Company [ Cardinality = one, Inverse = Employees, OnDelete = cascade ];
}
```

この例は、会社 (company) と従業員 (employee) との間の一対多リレーションシップを示しています。ここに示すように、Cardinality は、会社がリレーションシップの “One” の側であることを示す一方で、OnDelete は、会社を削除した場合の従業員への効果を示します。OnDelete の値は cascade であるため、会社を削除すると、その効果がカスケードされて従業員も削除されます。

OnDelete の値が noaction (一対多のリレーションシップの既定値) である場合、その会社を雇用主とする従業員がいると、会社を削除することはできません。

OnDelete の値が setnull または setdefault の場合、会社を削除すると、従業員の雇用主が NULL に設定されます。

関連項目

- プロパティ定義
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

Private (プロパティ・キーワード)

プロパティが Private (このクラスまたはそのサブクラスのメソッドによってのみ使用可能) であるかどうかを指定します。

使用法

プロパティを Private として指定するには、以下の構文を使用します。

```
Property name As classname [ Private ];
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 Not を指定します。

詳細

プライベート・クラス・メンバは、このクラス (またはサブクラス) のメソッドによってのみ使用できます。

プライベート・プロパティは、カタログ情報 (%Library.SQLCatalog を使用してアクセスします) には表示されず、SELECT * クエリでも返されません。ただし、SQL クエリではプライベート・プロパティを明示的に参照し、使用できます。

サブクラスは Private キーワードの値を継承しますが、それをオーバーライドできません。

InterSystems IRIS では、プライベート・プロパティは常に、プロパティを定義したクラスのサブクラスに継承され、参照できます。他の言語では、これらを保護されたプロパティと呼ぶことがあります。

既定値

このキーワードを省略すると、プロパティは Private ではなくなります。

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

ReadOnly (プロパティ・キーワード)

プロパティが読み取り専用であることを指定します。読み取り専用にすると、その値の設定方法が制限されます。

使用法

プロパティを ReadOnly と指定するには、以下の構文を使用します。

```
Property name As classname [ ReadOnly ];
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

重要　　コレクション・プロパティには ReadOnly キーワードを使用しないでください。

詳細

このキーワードは、オブジェクト参照を使用してそのプロパティの値を設定できないことを指定します。次のように、オブジェクト参照を使用して読み取り専用プロパティの値を設定しようとしたとします。

```
set oref.Name = "newvalue"
```

この場合は、実行時に <CANNOT SET THIS PROPERTY> エラーが発生します。

同様に、プロパティが読み取り専用に定義されている場合、対応する SQL テーブル内のフィールドも読み取り専用に定義されます。読み取り専用のフィールドを SQL 文を使用して明示的に挿入または更新することはできません。これを行おうとすると、SQL エラー (SQLCODE = -138) になります。

読み取り専用プロパティの値は、以下の方法で指定できます。

- InitialExpression キーワードを使用する。
- SQLComputeCode キーワードを使用する。
- "プロパティ・メソッドの使用とオーバーライド" に説明されているプロパティ・メソッド内に指定する。

これらの各方法には特定の制約があることに注意してください。

メモ

プロパティが読み取り専用と必須の両方としてマークされている場合、オブジェクト・アクセスと SQL アクセスでは動作に次のような違いがあることに注意してください。

- InterSystems IRIS は、オブジェクトの保存時にプロパティを検証しません。この結果、InterSystems IRIS はそのプロパティの必須キーワードを無視します。
- InterSystems IRIS は、レコードの挿入時または更新時にプロパティの必須キーワードを考慮します。

既定値

このキーワードを省略すると、プロパティは ReadOnly ではなくなります。

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業

- オブジェクト値プロパティの定義と使用
- コンパイラ・キーワードの概要

Required (プロパティ・キーワード)

永続クラスで、プロパティ値をディスクに保存する前に、値を指定する必要があることを示します。XML 対応クラスでは、プロパティのマップ先の要素が必須であることを示します。

使用法

プロパティを Required として指定するには、以下の構文を使用します。

```
Property name As classname [ Required ];
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

永続クラスで、このキーワードはプロパティに値を指定してからでないと格納オブジェクトをディスクに保存できないことを示します。プロパティに値が指定されていない場合はエラーが発生します。

プロパティが %Stream タイプの場合、そのストリームを NULL ストリームにできません。つまり、%IsNull() メソッドが 0 を返した場合、このストリーム・プロパティは値を持っていると見なされます。

%XML.Adaptor を拡張するクラスの場合、このキーワードは対応する XML スキーマに影響します。プロパティが Required と指定されている場合、スキーマ内の対応要素に `minOccurs="0"` はなく、必須と見なされます。"オブジェクトの XML への投影" を参照してください。この場合、クラスが永続クラスでなくてもかまいません。XML スキーマの検証は、InterSystems IRIS が XML ドキュメントを読み取る際に行われます。"XML ツールの使用法" を参照してください。

サブクラスで、必要に応じてオプションのプロパティをマークできますが、その逆は実行できません。

メモ

プロパティが読み取り専用と必須の両方としてマークされている場合、オブジェクト・アクセスと SQL アクセスでは動作に次のような違いがあることに注意してください。

- InterSystems IRIS は、オブジェクトの保存時にプロパティを検証しません。この結果、InterSystems IRIS はそのプロパティの必須キーワードを無視します。
- InterSystems IRIS は、レコードの挿入時または更新時にプロパティの必須キーワードを考慮します。

既定値

このキーワードを省略すると、プロパティは Required ではなくなります。

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

ServerOnly (プロパティ・キーワード)

このプロパティを Java クライアントに投影するかどうかを指定します。

使用法

このプロパティを Java クライアントに投影するかどうかを指定するには、以下の構文を使用します。

```
Property name As classname [ ServerOnly = n ];
```

n は以下のいずれかになります。

- 0 は、このプロパティが投影されることを表します。
- 1 は、このプロパティが投影されないことを表します。

詳細

このキーワードは、プロパティを Java クライアントに投影するかどうかを指定します。

既定値

このキーワードを省略すると、プロパティは投影されます。

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

SqlColumnNumber (プロパティ・キーワード)

このプロパティの SQL 列番号を指定します。永続クラスのみに適用されます。

使用法

プロパティの SQL 列番号を指定するには、以下の構文を使用します。

```
Property name As classname [ SqlColumnNumber = 4 ];
```

n は、2 ～ 4096 の数値です。

詳細

このキーワードにより、このプロパティの SQL 列順序を明示的に設定できます。SQL 列順序は、SELECT * 文を通じてテーブルにクエリを実行する場合、または列順序を明示的に指定せずに INSERT または LOAD DATA コマンドを使用する場合に使用されます。

このプロパティ・キーワードを設定すると、クラス定義により暗示される順序を、テーブルとしてのこのクラスの SQL プロジェクションから切り離すことができます。

既定値

既定値は空文字列です。

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

SqlComputeCode (プロパティ・キーワード)

このプロパティの値を設定するコードを指定します。

使用法

プロパティの計算方法を指定するには、以下の構文を使用します。

```
Property name As classname [ SqlComputeCode = { Set {FieldName} = Expression }, SqlComputed ];
```

以下は、この指定の説明です。

- FieldName ー 定義されるプロパティの SQL フィールド名。
- Expression ー プロパティの値を指定する ObjectScript 式。

詳細

このキーワードを指定した場合 (また、SqlComputed が True の場合)、このプロパティは計算プロパティとなります。"計算プロパティの定義" を参照してください。

このキーワードの値については、以下の規則に従って、プロパティの値を設定する ObjectScript コードの行を指定します (中括弧で囲む)。

- このプロパティを参照するには、**{*}** を使用します。

  また、SqlFieldName キーワードがプロパティに対して指定されていない場合は、**{propertyname}** を使用します。この propertyname はプロパティの名前です。SqlFieldName キーワードがプロパティに対して指定されている場合は、**{sqlfieldnamevalue}** を使用します。この sqlfieldnamevalue は、そのキーワードの値です。

  SqlFieldName が役立つのは永続クラスに対してのみですが、すべてのオブジェクト・クラスに使用できます。

  ObjectScript コード内のフィールド名の詳細は、"リテラル・プロパティの SQL プロジェクションの制御" または "CREATE TRIGGER" を参照してください。

- 同様に、別のプロパティを参照するために、SqlFieldName キーワードがプロパティに対して指定されていない場合は、**{propertyname}** を使用します。この propertyname はプロパティの名前です。SqlFieldName キーワードがプロパティに対して指定されている場合は、**{sqlfieldnamevalue}** を使用します。この sqlfieldnamevalue は、そのキーワードの値です。

  {Propertyname} 内で、矢印構文または相対ドット構文を使用することはできません。例えば、**{objectprop.propA}** や **{objectprop->propA}** のような表現はサポートされていません。

- 必要に応じて、コードに複数の Set コマンドを含めることができます。等号の前後には空白文字を使用できますが、Set 文のそれぞれは全体を 1 行で記述する必要があります。
- コードでは、通常の完全構文を使用して、クラス・メソッド (インスタンス・メソッドではない)、ルーチン、またはサブルーチンを参照できます。同様に、ObjectScript 関数と演算子を使用できます。
- コードには、埋め込み SQL を含めることができます。
- コードには、以下の擬似フィールド参照変数を含めることができます。これは、クラスのコンパイル時に特定の値に変換されます。
  - {%%CLASSNAME} と {%%CLASSNAMEQ} は、ともに SQL テーブル定義を投影するクラスの名前に変換されます。{%%CLASSNAME} は引用符なし文字列を返し、{%%CLASSNAMEQ} は引用符付き文字列を返します。
  - {%%TABLENAME} はテーブルの完全修飾名に変換され、引用符付き文字列を返します。
  - {%%ID} は RowID 名に変換されます。この参照は、RowID フィールドの名前がわからないときに役立ちます。

これらの名前は、大文字と小文字を区別しません。

- 計算コードは必ず ObjectScript で記述します。Python などの他の言語によるコードを指定するには、代わりに *Property*Computation クラス・メソッドを指定します。*Property*Computation は計算するプロパティの名前です。詳細は、"計算プロパティ値" を参照してください。

  SqlComputeCode メソッドと *Property*Computation メソッドの両方を指定すると、SqlComputeCode によって *Property*Computation がオーバーライドされます。

- コードでは、..propertyname の形式および ..methodname() の形式の構文は使用できません。

以下に例を示します。

```
Property TestProp As %String [ SqlComputeCode = {set {*} = {OtherField}}, SqlComputed ];
```

別の例を示します。

```
Property FullName As %String [ SqlComputeCode = {set {*}={FirstName}_" "_{LastName}}, SqlComputed ];
```

このコードは、Do コマンドで呼び出します。

重要

- このフィールドのインデックス付けを予定している場合は、非決定的コードではなく、決定的コードを使用します。非決定的コードは、古いインデックス・キー値を確実に削除することができないため、InterSystems IRIS は非決定的コードの結果に対するインデックスを保持できません。(決定的コードは、同じ引数を渡された場合はいつでも同じ値を返します。例えば、$h を返すコードは、$h が関数の制御外で変更されているため、非決定的です。)
- SqlComputeCode で使用されるユーザ変数はすべて、使用前に New が適用されます。これにより、関連コード内の別の場所にある同名変数との競合を回避できます。

既定値

既定値は空文字列です。

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

SqlComputed (プロパティ・キーワード)

これが計算プロパティであるかどうかを示します。

使用法

このプロパティを計算プロパティとして指定するには、以下の構文を使用します。

```
Property name As classname [ SqlComputeCode = sqlcomputecode, SqlComputed ];
```

sqlcomputecode には、プロパティの計算に使用するコードを指定します (SqlComputeCode の説明を参照)。計算コードを *Property*Computation メソッドで定義する場合は SqlComputeCode を省略します。詳細は、"計算プロパティ値" を参照してください。

プロパティを計算しない場合は、このキーワードを省略するか、キーワードの直前に単語 Not を指定します。

詳細

このキーワードが True の場合 (また、このプロパティで SqlComputeCode も指定されている場合)、このプロパティは計算プロパティとなります。オプションおよびその他の詳細は、"計算プロパティの定義" を参照してください。

プロパティに SqlComputed キーワードの値がある場合、InterSystems IRIS はその値をプロパティの計算に使用します。具体的には、SqlComputeCode から新しいクラス・メソッド <property>Compute が生成されます。このメソッドは、プロパティの <property>Get メソッドから呼び出します。また、プロパティに SqlComputeOnChange キーワードも指定した場合は、<property>Compute メソッドが、指定された時刻に呼び出されます。

この機能は、<property>Get メソッドと <property>Set メソッドに実装されています。これらのメソッドのいずれかをオーバーライドすると、オーバーライドの対象とするメソッドに計算をトリガする条件がない限り、プロパティの計算は機能しなくなります。

既定値

このキーワードを省略すると、このプロパティは計算プロパティではなくなります。

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

SqlComputeOnChange (プロパティ・キーワード)

このキーワードは、プロパティをいつ再計算するかを制御します。トリガされる計算プロパティのみに適用します。

使用法

プロパティをいつ再計算するかを指定するには、以下の構文を使用します。

```
Property name As classname [ SqlComputed, SqlComputeCode=sqlcomputecode, SqlComputeOnChange =
propertynames ];
```

引数は以下のとおりです。

- sqlcomputecode には、プロパティの計算に使用するコードを指定します (SqlComputeCode の説明を参照)。計算コードを *Property*Computation メソッドで定義する場合は SqlComputeCode を省略します。詳細は、"計算プロパティ値" を参照してください。
- propertynames には、1 つのプロパティ名、または複数のプロパティ名のコンマ区切りリストのいずれかを指定します。この値には、値 %%INSERT または %%UPDATE を含めることもできます。SqlFieldname によって与えられる値ではなく、実際のプロパティ名を使用する必要があることに注意してください。

詳細

このキーワードは、トリガされる計算プロパティにのみ適用可能であり、他のプロパティでは無視されます。(トリガされる計算プロパティは、SqlComputed が True で、SqlComputeCode が指定されているが、Calculated および Transient が両方とも False のプロパティです。"計算プロパティの定義" を参照してください。)

このキーワードでは、このプロパティを再計算する条件を制御します。以下の場合に、再計算が行われます。

- 指定されているプロパティの変更
- トリガ・イベントの発生

キーワードに %%INSERT または %%UPDATE の値が指定されている場合は、INSERT 呼び出しまたは UPDATE 呼び出しが、フィールド (プロパティ) の値のイベント・トリガ計算をそれぞれ指定します。

- %%INSERT を指定すると、InterSystems IRIS は行がテーブルに挿入されたときにフィールドの値を計算します。InterSystems IRIS は、SQLComputeCode キーワードで指定されたコードを呼び出して、値を設定します。SQLComputeCode が同じフィールドを入力値として使用している場合、InterSystems IRIS はそのフィールドに対して明示的に指定された値を使用します。値が指定されていない場合、InterSystems IRIS は InitialExpression (これが指定されている場合) または NULL (InitialExpression が指定されていない場合) を使用します。
- %%UPDATE を指定すると、InterSystems IRIS は行がテーブルに挿入されたときにフィールドの値を計算し、行が更新されたときにそれを再計算します。どちらの場合も、InterSystems IRIS は、SQLComputeCode キーワードで指定されたコードを呼び出して、値を設定します。SQLComputeCode が同じフィールドを入力値として使用している場合、InterSystems IRIS はそのフィールドに対して明示的に指定された値を使用します。値が指定されていない場合、InterSystems IRIS は前のフィールド値を使用します。

イベント・トリガ計算は、検証と正規化の直前に発生します (検証と正規化の後にデータベースへの値の書き込みが続きます)。

この例では、行が挿入または更新されると、プロパティ LastUpdate が現在の日付と時刻で更新されます。

```
Property LastUpdate As %TimeStamp [ SqlComputeCode = {set {*}=$ZDATETIME($NOW(),3)}, SqlComputed,
SqlComputeOnChange = (%%INSERT, %%UPDATE) ];
```

注釈　フィールド値のイベント・トリガ計算は、プロパティ値を計算するコードによっては、プロパティに対して明示的に指定されている値をオーバーライドすることがあります。

既定値

SqlComputeOnChange キーワードの既定値は、空の文字列です。

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

SqlFieldName (プロパティ・キーワード)

SQL プロジェクションで使用するフィールド名を指定します。永続クラスに適用されます。

使用法

テーブルが SQL に投影されるときのこのプロパティの既定の名前をオーバーライドするには、以下の構文を使用します。

```
Property name As classname [ SqlFieldName = sqlfieldname ];
```

sqlfieldname は SQL 識別子です。

詳細

このキーワードは、SQL プロジェクションでプロパティを識別するために使用される列名を指定します。

既定値

このキーワードを省略すると、プロパティ名が SQL 列名として使用されます。

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

SqlListDelimiter (プロパティ・キーワード)

SQL 内でリストに使用する区切り文字を指定します。永続クラス内のリスト・プロパティに適用されます。従来のアプリケーションでのみ使用されます。

使用法

このリスト・プロパティに対して SQL 内で使用する区切り文字を指定するには、以下の構文を使用します。

```
Property Name As List Of Classname [ SqlListDelimiter = """delimiter""", SqlListType = DELIMITED ];
```

delimiter は区切り文字です。

詳細

このキーワードは、このプロパティがリストであり、SqlListType が DELIMITED または SUBNODE の場合に、このプロパティ用に SQL 内で使用する区切り文字を指定します。このキーワードは、従来のアプリケーションをサポートするために提供されています。

既定値

SqlListDelimiter キーワードの既定値は、空の文字列です。

例

```
Property Things As list Of %String [ SqlListDelimiter = """,""", SqlListType = DELIMITED ];
```

関連項目

- プロパティ定義
- コレクションを使用した作業
- コンパイラ・キーワードの概要

SqlListType (プロパティ・キーワード)

このフィールドの値を SQL のメモリ内で表現し、ディスクに格納することを指定します。永続クラス内のリスト・プロパティのみに適用されます。従来のアプリケーションでのみ使用されます。

使用法

```
Property Name As List Of Classname [ SqlListType = sqllisttype ];
```

sqllisttype は以下のいずれかになります。

- `LIST` ー リストは $List() 形式でメモリおよびディスクに格納されます。これが既定値です。
- `DELIMITED` ー リストは区切り文字列としてメモリおよびディスクに格納されます。区切り文字は、SqlListDelimiter キーワードで指定されます。
- `SUBNODE` ー リストはディスク上のサブノードに格納されます。つまり、各リスト要素は個々のグローバル・ノードに格納されます。SqlListDelimiter が指定されていない場合、フィールドのメモリ内の値は $List 形式です。SqlListDelimiter が指定されている場合、メモリ内の形式は区切り文字列です。

詳細

SqlListType は、フィールドの値を SQL のメモリ内でどのように表現するのか、および、ディスクにどのように格納するのかを制御します。

このキーワードは、従来のアプリケーションをサポートするために提供されています。

既定値

既定値は `LIST` です。

関連項目

- プロパティ定義
- コレクションを使用した作業
- コンパイラ・キーワードの概要

Transient (プロパティ・キーワード)

プロパティをデータベースに格納するかどうかを指定します。永続クラスのみに適用されます。

使用法

プロパティをデータベースに格納しないことを指定するには、以下の構文を使用します。

```
Property name As classname [ Transient ];
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 Not を指定します。

詳細

クラス・タイプが永続クラスである場合に、このプロパティをデータベースに保存しないことを指定します。

InterSystems IRIS は、一時プロパティを他のプロパティと同じように検証します。例えば、オブジェクトを保存しようとする場合、システムは一時プロパティを含め、そのすべてのプロパティを検証します。

サブクラスでは、一時的でないプロパティを一時としてマークすることは可能ですが、その逆は不可能です。

既定値

このキーワードを省略すると、プロパティは Transient ではなくなります。

関連項目

"計算プロパティの定義" を参照してください。

関連項目

- プロパティ定義
- リテラル・プロパティの定義と使用
- コレクションを使用した作業
- ストリームを使用した作業
- オブジェクト値プロパティの定義と使用
- リレーションシップの定義と使用
- コンパイラ・キーワードの概要

クエリの構文とキーワード

このリファレンスでは、クラス・クエリに適用する構文とキーワードについて説明しています。キーワード（クラス属性とも呼ばれる）は、一般にコンパイラに影響を与えます。

クエリ定義に関する一般情報へのリンクは、"クエリ定義" を参照してください。

特定のクラス・メンバに適用するすべての構造とキーワードは、目次を参照してください。

クラス定義でのクエリの構文

クエリ定義の構造を説明します。クエリ定義に関する一般情報へのリンクは、"クエリ定義" を参照してください。

概要

クラス・クエリは、クラス構造の一部である名前付きクエリであり、ダイナミック SQL を通じてアクセスできます。

クラス・クエリはどのクラス内でも定義できます。クラス・クエリは永続クラス内に含める必要はありません。

詳細

クエリ定義の構造は以下のとおりです。

```
/// description
Query name(formal_spec) As classname [ keyword_list ]
{ implementation }
```

以下は、この指定の説明です。

- description (オプション) は、クラス・リファレンスでの表示を意図しています。description は既定ではブランクです。"クラス・ドキュメントの作成" を参照してください。
- name (必須) はクエリの名前です。これは、有効なクラス・メンバ名で、かつ他のクラス・メンバ名と重複しないものである必要があります。
- formal_spec (オプション) は、クエリに渡す引数のリストを指定します。

  具体的には、これは関連するクエリ・クラスの Execute() メソッドを通してクエリに渡す引数のリストです。

  "クラス定義でのメソッドの構文" の formal_spec のコメントを参照してください。
- classname (必須) は、このクエリで使用するクエリ・クラスを指定します。

  通常、SQL ベースのクエリには **%SQLQuery**、カスタム・クエリには **%Query** を指定します。"クラス・クエリの定義と使用" を参照してください。

  注釈　　カスタム・クラス・クエリは、シャード・クラスではサポートされていません。
- keyword_list (オプション) は、さらにクエリを定義するキーワードのコンマ区切りリストです。

  すべてのキーワードは、"クエリの構文とキーワード" を参照してください。

  このリストを省略する場合は角括弧も省略します。
- implementation (オプション) は、クエリを定義するコードのゼロ以上の行です。

関連項目

- クラス・クエリの定義と使用
- メソッドの定義と呼び出し
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- クラスの制限

ClientName (クエリ・キーワード)

このクエリのクライアント・プロジェクションに使用するエイリアスです。

使用法

クライアント言語に投影されるときのクエリの既定の名前をオーバーライドするには、以下の構文を使用します。

```
Query name(formal_spec) As classname [ ClientName = clientname ] {    //implementation }
```

clientname はクライアント言語で使用する名前です。

詳細

このキーワードにより、クエリにクライアントの言語を投影するときに、クエリに別名を定義できます。このキーワードは、クエリ名にクライアント言語では許可されない文字が含まれている場合に特に役立ちます。

既定値

このキーワードを省略すると、クエリ名がクライアント名として使用されます。

関連項目

- クエリ定義
- クラス・クエリの定義と使用
- コンパイラ・キーワードの概要

Final (クエリ・キーワード)

このクエリが Final である (サブクラス内でオーバーライドできない) かどうかを指定します。

使用法

クエリを Final として指定するには、以下の構文を使用します。

```
Query name(formal_spec) As classname [ Final ] {    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

final としてマークされるクラス・メンバは、サブクラスではオーバーライドできません。

既定値

このキーワードを省略すると、クエリは Final ではなくなります。

関連項目

- クエリ定義
- クラス・クエリの定義と使用
- コンパイラ・キーワードの概要

Internal (クエリ・キーワード)

このクエリ定義が Internal (クラス・ドキュメントに表示されない) であるかどうかを指定します。

使用法

このクエリ定義を Internal として指定するには、以下の構文を使用します。

```
Query name(formal_spec) As classname [ Internal ] {    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

既定値

Internal クラスのメンバは、クラス・ドキュメントには表示されません。このキーワードは、ユーザに対してクラスは表示するが、そのすべてのメンバを表示する必要がない場合に役立ちます。

既定値

このキーワードを省略すると、このクエリはクラス・ドキュメントに表示されます。

関連項目

- クエリ定義
- クラス・クエリの定義と使用
- コンパイラ・キーワードの概要

Private (クエリ・キーワード)

クエリが Private であるかどうかを指定します。

使用法

このクエリを Private として指定するには、以下の構文を使用します。

```
Query name(formal_spec) As classname [ Private ] {    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

プライベート・クラス・メンバは、同じクラス (またはそのサブクラス) の他のメンバによってのみ使用できます。他の言語では、この種の可視性を表すために protected という用語を、サブクラスから見えないことを表すために private という用語を使用していることがよくあるので注意してください。

既定値

このキーワードを省略すると、このクエリは Private ではなくなります。

関連項目

- クエリ定義
- クラス・クエリの定義と使用
- コンパイラ・キーワードの概要

SoapBindingStyle (クエリ・キーワード)

Web メソッドとして使用される場合に、このクエリが使用するバインディング・スタイルまたは SOAP 呼び出し機能を指定します。Web サービスまたは Web クライアントとして定義されているクラス内でのみ適用されます。

使用法

クエリが (Web メソッドとして使用される場合に) 使用する既定のバインディング・スタイルをオーバーライドするには、以下の構文を使用します。

```
Query name(formal_spec) As classname [ WebMethod, SoapBindingStyle = soapbindingstyle ] {
//implementation }
```

soapbindingstyle は以下のいずれかの値になります。

- `document` ― この Web メソッドはドキュメント・スタイルの呼び出しを使用します。

  このバインディング・スタイルにより、SOAP メッセージはドキュメントとしてフォーマットされ、通常はパートを 1 つのみ持ちます。

  SOAP メッセージでは、通常、`<Body>` 要素に 1 つの子要素が含まれます。`<Body>` 要素のそれぞれの子は、メッセージ・パートに対応します。

- `rpc` ― この Web メソッドは RPC (リモート・プロシージャ呼び出し) スタイルの呼び出しを使用します。

  このバインディング・スタイルにより、SOAP メッセージは複数のパートを持つメッセージとしてフォーマットされます。

  SOAP メッセージでは、`<Body>` 要素に 1 つの子要素が含まれ、その名前は対応する処理名から取得されます。この要素は生成されたラッパ要素であり、これにはメソッドの引数リストの引数ごとに 1 つの子要素が含まれます。

重要　手動で作成した Web サービスでは、通常はこのキーワードの既定値が最適な値となります。SOAP ウィザードで WSDL から Web クライアントまたは Web サービスを生成すると、このキーワードはその WSDL に最適に設定されます。この値を変更すると、Web クライアントまたは Web サービスは機能しなくなる場合があります。

詳細

このキーワードは、このクエリが Web メソッドとして呼び出された場合の、このクエリによって使用されるバインディング・スタイルを指定します。

このキーワードは、指定されたクエリに対して、SoapBindingStyle クラス・キーワードをオーバーライドします。

既定値

このキーワードを省略すると、`<soap:operation>` 要素の `style` 属性が、代わりに SoapBindingStyle クラス・キーワードの値により決定されます。

WSDL との関係

詳細は、SoapBindingStyle メソッド・キーワードのエントリを参照してください (同じ名前のクラス・キーワードがあると、メソッド・キーワードやクエリ・キーワードの動作よりも WSDL のパートのほうに大きく影響します)。

SOAP メッセージへの影響

詳細は、SoapBindingStyle クラス・キーワードのエントリを参照してください。

関連項目

- クエリ定義

- クラス・クエリの定義と使用
- コンパイラ・キーワードの概要
- Web サービスおよび Web クライアントの作成

SoapBodyUse (クエリ・キーワード)

Web メソッドとして使用される場合に、このクエリの入出力で使用されるエンコードを指定します。Web サービスまたは Web クライアントとして定義されているクラス内でのみ適用されます。

使用法

クエリ (Web メソッドとして使用される場合) の入力および出力により使用される既定のエンコードをオーバーライドするには、以下の構文を使用します。

```
Query name(formal_spec) As classname [ WebMethod, SoapBodyUse = encoded ] {    //implementation }
```

soapbodyuse は以下のいずれかの値になります。

- `literal` ― この Web メソッドは、リテラル・データを使用します。つまり、SOAP メッセージの `<Body>` の中の XML は、WSDL で提供されるスキーマと一致します。
- `encoded` ― この Web メソッドは、SOAP でエンコードされたデータを使用します。つまり、SOAP メッセージの `<Body>` の中の XML では、以下の仕様に従い、使用される SOAP のバージョンに応じて SOAP エンコードを使用します。
  - SOAP 1.1 (https://www.w3.org/TR/2000/NOTE-SOAP-20000508/)
  - SOAP 1.2 (https://www.w3.org/TR/soap12-part2/)

重要　手動で作成した Web サービスでは、通常はこのキーワードの既定値が最適な値となります。SOAP ウィザードで WSDL から Web クライアントまたは Web サービスを生成すると、このキーワードはその WSDL に最適に設定されます。この値を変更すると、Web クライアントまたは Web サービスは機能しなくなる場合があります。

詳細

このキーワードは、このクエリが Web メソッドとして呼び出された場合の、その入出力のエンコードを指定します。

このキーワードは、指定されたクエリに対して、SoapBodyUse クラス・キーワードをオーバーライドします。

既定値

このキーワードを省略すると、SoapBodyUse クラス・キーワードの値が代わりに使用されます。

WSDL との関係と SOAP メッセージへの影響

詳細は、SoapBodyUse クラス・キーワードのエントリを参照してください。

関連項目

- クエリ定義
- クラス・クエリの定義と使用
- コンパイラ・キーワードの概要
- Web サービスおよび Web クライアントの作成

SoapNameSpace (クエリ・キーワード)

WSDL 内のバインディング操作レベルのネームスペースを指定します。Web サービスまたは Web クライアントとして定義されているクラス内でのみ適用されます。

使用法

バインディング操作レベルで既定のネームスペースを (クエリが Web メソッドとして使用される場合に) オーバーライドするには、以下の構文を使用します。

```
Query name(formal_spec) As classname [ SoapNameSpace = "soapnamespace", WebMethod ] {    //implementation
 }
```

soapnamespace はネームスペース URI です。URI にコロン (:) が含まれる場合、文字列は引用符で囲む必要があります。つまり、次のように使用できます。

```
Query MyQuery() [ SoapNameSpace = "http://www.mynamespace.org", WebMethod ]
```

また、次のようにすることもできます。

```
Query MyQuery() [ SoapNameSpace = othervalue, WebMethod ]
```

ただし、次のようにはできません。

```
Query MyQuery() [ SoapNameSpace = http://www.mynamespace.org, WebMethod ]
```

重要　手動で作成した Web サービスでは、通常はこのキーワードの既定値が最適な値となります。SOAP ウィザードで WSDL から Web クライアントまたは Web サービスを生成すると、このキーワードはその WSDL に最適に設定されます。この値を変更すると、Web クライアントまたは Web サービスは機能しなくなる場合があります。

詳細

このキーワードは、このクエリが Web メソッドとして呼び出された場合の、このクエリによって使用される XML ネームスペースを指定します。

注釈　このキーワードは、クエリが RPC スタイルのバインディングを使用する場合にのみ有効です。つまり、このクエリ (またはそれを含むクラス) を、SoapBindingStyle = `rpc` とマークする必要があります。(ドキュメント・スタイルのバインディングを使用するクエリでこのキーワードを指定する場合、WSDL は自己矛盾したものになります。)

詳細は、"Web サービスおよび Web クライアントの作成" を参照してください。

既定値

このキーワードを省略すると、Web メソッドは、Web サービスまたはクライアント・クラスの NAMESPACE パラメータで指定されるネームスペースに配置されます。

WSDL との関係と SOAP メッセージへの影響

詳細は、SoapNameSpace メソッド・キーワードのエントリを参照してください。

関連項目

- クエリ定義
- クラス・クエリの定義と使用
- コンパイラ・キーワードの概要

- Web サービスおよび Web クライアントの作成

SqlName (クエリ・キーワード)

投影された SQL ストアド・プロシージャの既定の名前をオーバーライドします。クエリが SLQ ストアド・プロシージャとして投影される場合にのみ適用されます。

使用法

クエリが SLQ ストアド・プロシージャとして投影されるときに使用される既定の名前をオーバーライドするには、以下の構文を使用します。

```
Query name(formal_spec) As classname [ SqlProc, SqlName = sqlname ] {    //implementation }
```

sqlname は SQL 識別子です。

詳細

このクエリを SQL ストアド・プロシージャとして投影するときに、この名前をストアド・プロシージャの名前に使用します。

既定値

このキーワードを省略すると、クエリ名が SQL プロシージャ名として使用されます。

関連項目

- クエリ定義
- クラス・クエリの定義と使用
- コンパイラ・キーワードの概要
- ストアド・プロシージャの定義と使用

SqlProc (クエリ・キーワード)

クエリを SQL ストアド・プロシージャとして呼び出すことができるかどうかを指定します。

使用法

クエリを SQL ストアド・プロシージャとして呼び出すことができると指定するには、以下の構文を使用します。

```
Query name(formal_spec) As classname [ SqlProc ] {    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

このキーワードは、クエリを SQL ストアド・プロシージャとして呼び出すことができるかどうかを指定します。

既定値

このキーワードを省略すると、クエリは SQL ストアド・プロシージャとして呼び出すことはできません。

関連項目

- クエリ定義
- クラス・クエリの定義と使用
- コンパイラ・キーワードの概要
- ストアド・プロシージャの定義と使用

SqlView (クエリ・キーワード)

クエリを SQL ビューとして投影するかどうかを指定します。

使用法

クエリを SQL ビューとして投影することを指定するには、以下の構文を使用します。

```
Query name(formal_spec) As classname [SqlView]  {    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

このキーワードは、InterSystems IRIS プロジェクトがこのクエリを SQL ビューとして投影するかどうかを指定します。

既定値

このキーワードを省略すると、InterSystems IRIS はこのクエリを SQL ビューとして投影しません。

関連項目

- クエリ定義
- クラス・クエリの定義と使用
- コンパイラ・キーワードの概要
- ビューの定義と使用

SqlViewName (クエリ・キーワード)

投影された SQL ビューの既定の名前をオーバーライドします。クエリが SLQ ビューとして投影される場合にのみ適用されます。

使用法

クエリが SLQ ビューとして投影されるときに使用される既定の名前をオーバーライドするには、以下の構文を使用します。

```
Query name(formal_spec) As classname [ SqlView, SqlViewName = "_Q1" ] {    //implementation }
```

sqlviewname は SQL 識別子です。

詳細

このキーワードは、このクエリから投影するビューの SQL エイリアスを指定します。

既定値

このキーワードを省略すると、SQL ビュー名がクエリ名となります。

関連項目

- クエリ定義
- クラス・クエリの定義と使用
- コンパイラ・キーワードの概要
- ビューの定義と使用

WebMethod (クエリ・キーワード)

クラス・クエリが Web メソッドであるかどうかを指定します。Web サービスまたは Web クライアントとして定義されているクラス内でのみ適用されます。

使用法

このクエリを Web メソッドとして指定するには、以下の構文を使用します。

```
Query name(formal_spec) As classname [ WebMethod ] {    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

このキーワードは、このクラス・クエリが Web メソッドであり、SOAP プロトコル経由での呼び出しが可能であるかどうかを指定します。

Web メソッドの要件については、"Web サービスおよび Web クライアントの作成" を参照してください。

既定値

このキーワードを省略すると、クエリは Web メソッドとして呼び出すことはできません。

生成されたクラス

クラス・クエリにこのキーワードを追加し、クラスをコンパイルすると、クラス・コンパイラは、次の 2 つの追加クラスを生成します。

- Package.OriginalClass.QueryName
- Package.OriginalClass.QueryName`.DS`

Package.OriginalClass は Web メソッドを含むクラス、QueryName はクラス・クエリの名前です。

例えば、クラス `ROBJDemo.QueryWS` で開始し、それに `MyQuery` という名前のクラス・クエリを追加したとします。そのクラス・クエリに WebMethod キーワードを追加してコンパイルすると、クラス・コンパイラは、次の追加クラスを生成します。

- `ROBJDemo.QueryWS.MyQuery`
- `ROBJDemo.QueryWS.MyQuery.DS`

これらの生成されたクラスを変更したり、直接使用したりしないでください。これらは内部で使用するために設計されたものです。

WSDL との関係

Web サービスでは、このキーワードは、生成される WSDL にも影響します。ここには、この Web メソッドを示すのに必要な追加要素が含まれるようになります。

関連項目

- クエリ定義
- クラス・クエリの定義と使用
- コンパイラ・キーワードの概要
- Web サービスおよび Web クライアントの作成

トリガの構文とキーワード

このリファレンスでは、永続クラス内で定義できる SQL トリガに適用する構文とキーワードを説明します。キーワード (クラス属性とも呼ばれる) は、一般にコンパイラに影響を与えます。

トリガ定義に関する一般情報へのリンクは、"トリガ定義" を参照してください。

特定のクラス・メンバに適用するすべての構造とキーワードは、目次を参照してください。

クラス定義でのトリガの構文

トリガ定義の構造を説明します。トリガ定義に関する一般情報へのリンクは、"トリガ定義" を参照してください。

概要

トリガは、特定のイベントが InterSystems SQL で発生したときに実行されるコード・セグメントです。InterSystems IRIS は INSERT コマンド、UPDATE コマンド、DELETE コマンドの実行をベースにしたトリガをサポートします。トリガ定義により、指定されたコードは関連するコマンドが実行される直前、または直後に実行されます。各イベントは、実行順序が指定されていれば複数のトリガを持つことができます。

トリガ定義は永続クラスに追加できます。他のクラスでは意味がありません。

詳細

トリガ定義の構造は以下のとおりです。

```
/// description
Trigger name [ keyword_list ]
{ implementation }
```

以下は、この指定の説明です。

- description (オプション) は、クラス・リファレンスでの表示を意図しています。description は既定ではブランクです。"クラス・ドキュメントの作成" を参照してください。
- name (必須) はトリガの名前です。これは、有効なクラス・メンバ名で、かつ他のクラス・メンバ名と重複しないものである必要があります。
- keyword_list (必須) は、さらにトリガを定義するキーワードのコンマ区切りリストです。

  すべてのキーワードは、"トリガの構文とキーワード" を参照してください。
- implementation (必須) は、トリガの起動時に実行するコードを定義したゼロ行以上のコードです。

例

Class Member

```
/// This trigger updates the LogTable after every insert
Trigger LogEvent [ Event = INSERT, Time = AFTER ]
{
    // get row id of inserted row
    NEW id
    SET id = {ID}

    // INSERT value into Log table
    &sql(INSERT INTO LogTable (TableName, IDValue) VALUES ('MyApp.Person', :id))

}
```

関連項目

- トリガの使用法
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- クラスの制限

CodeMode (トリガ・キーワード)

このトリガの実装方法を指定します。

使用法

トリガの実装方法を指定するには、以下の構文を使用します。

```
Trigger name [ Event = sqlevent, CodeMode = codemode ]  {    //implementation }
```

codemode は以下のいずれかになります。

- `code` ― このトリガは、コード行として実装されます (既定)。
- `objectgenerator` ― このトリガは、トリガ・ジェネレータです。

注釈　このキーワードには古い値 (`generator`) があり、これは互換性を保つ目的でのみ残されています。新しいアプリケーションでは `objectgenerator` を使用する必要があります。

詳細

このキーワードは、特定のトリガの実装方法を指定します。

既定では、トリガ・コードは、トリガの起動時に実行される 1 行以上のコードで構成されます。

ただし、CodeMode が `objectgenerator` の場合、トリガは実際にはトリガ・ジェネレータです。トリガ・ジェネレータは、クラス・コンパイラによって呼び出されるプログラムであり、特定のトリガに対する実際の実装を生成します。この場合、トリガ・コードが、生成されたコードを処理します。論理はメソッド・ジェネレータの論理と似ています。"メソッド・ジェネレータとトリガ・ジェネレータの定義" を参照してください。

既定値

既定値は `code` です。つまり、既定では、トリガはトリガ・ジェネレータではありません。

関連項目

- トリガ定義
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- トリガの使用法
- コンパイラ・キーワードの概要

Event (トリガ・キーワード)

このトリガを起動する SQL イベントを指定します。必須 (既定はありません)。

使用法

トリガを起動する SQL イベントを指定するには、以下の構文を使用します。

```
Trigger name [ Event = sqlevent, Time = AFTER ] {    //implementation }
```

sqlevent は以下のいずれかの値になります。

- `DELETE` ー このトリガは、SQL DELETE 操作の間に起動します。
- `INSERT` ー このトリガは、SQL INSERT 操作の間に起動します。
- `UPDATE` ー このトリガは、SQL UPDATE 操作の間に起動します。
- `INSERT/UPDATE` ー このトリガは、SQL INSERT 操作または SQL UPDATE 操作の間に起動します。
- `INSERT/DELETE` ー このトリガは、SQL の挿入操作または削除操作の間に起動します。
- `UPDATE/DELETE` ー このトリガは、SQL の更新操作または削除操作の間に起動します。
- `INSERT/UPDATE/DELETE` ー このトリガは、SQL INSERT 操作、SQL UPDATE 操作、または SQL DELETE 操作の間に起動します。

詳細

このキーワードは、トリガを起動する SQL イベントを指定します。

既定値

既定値はありません。トリガを定義する場合は、このキーワードの値を指定する必要があります。

関連項目

- トリガ定義
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- トリガの使用法
- コンパイラ・キーワードの概要

Final (トリガ・キーワード)

このトリガが Final である (サブクラス内でオーバーライドできない) かどうかを指定します。

使用法

トリガを Final として指定するには、以下の構文を使用します。

```
Trigger name [ Event = sqlevent, Final ] {    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

final としてマークされるクラス・メンバは、サブクラスではオーバーライドできません。

既定値

このキーワードを省略すると、トリガは Final ではなくなります。

関連項目

- トリガ定義
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- トリガの使用法
- コンパイラ・キーワードの概要

Foreach (トリガ・キーワード)

いつトリガが起動されるかを制御します。

使用法

トリガがいつ起動されるかを指定するには、以下の構文を使用します。

```
Trigger name [ Event = sqlevent, Foreach = foreach ]  {    //implementation }
```

foreach は以下のいずれかの値になります。

- `row` ー このトリガは、トリガ文の影響を受ける各行によって起動されます。行レベルのトリガの場合、Language キーワードを `objectscript` または `python` に設定できます。
- `row/object` ー このトリガは、トリガ文の影響を受ける各行、またはオブジェクト・アクセスによる変更によって起動します。行レベルのトリガの場合、Language キーワードを `objectscript` または `python` に設定できます。

  このオプションは、統一トリガを定義します。SQL またはオブジェクト・アクセスにより発生するデータ変更によって起動されるトリガであるため、そのように呼ばれます。対照的に、他のトリガを使用すると、オブジェクト・アクセスによる変更の発生時に同じ論理を使用するには、%OnDelete() などのコールバックを実装する必要があります。
- `statement` ー このトリガは、文全体で 1 回起動されます。文レベルのトリガの場合、Language キーワードを `objectscript` または `tsql` に設定できます。

詳細

いつトリガが起動されるかを制御します。

既定値

このキーワードを省略すると、トリガは行レベルのトリガになります。

例外

TSQL では、行レベル・トリガはサポートされていません。

Python では、文レベル・トリガはサポートされていません。

関連項目

- トリガ定義
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- トリガの使用法
- コンパイラ・キーワードの概要

Internal (トリガ・キーワード)

このトリガ定義が Internal (クラス・ドキュメントに表示されない) であるかどうかを指定します。

使用法

このトリガ定義を Internal として指定するには、以下の構文を使用します。

```
Trigger name [ Event = sqlevent, Internal ] {    //implementation }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 `Not` を指定します。

詳細

Internal クラスのメンバは、クラス・ドキュメントには表示されません。このキーワードは、ユーザに対してクラスは表示するが、そのすべてのメンバを表示する必要がない場合に役立ちます。

既定値

このキーワードを省略すると、このトリガはクラス・ドキュメントに表示されます。

関連項目

- トリガ定義
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- トリガの使用法
- コンパイラ・キーワードの概要

Language (トリガ・キーワード)

トリガを記述する言語を指定します。

使用法

トリガを記述する言語を指定するには、以下の構文を使用します。

```
Trigger NewTrigger1 [ Event = sqlevent, Language = language ] {    //implementation }
```

language は以下のいずれかの値になります。

- `objectscript` ー このトリガは、ObjectScript で記述されます (既定)。
- `python` ー このトリガは、Python で記述されます。
- `tsql` ー このトリガは、TSQL で記述されます。この値を使用する場合、トリガは文レベルのトリガである必要があります。つまり、Foreach キーワードの設定が `statement` でなければなりません。

詳細

このキーワードは、トリガを記述する言語を指定します。

既定値

このキーワードを省略すると、言語は ObjectScript になります。

関連項目

- トリガ定義
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- トリガの使用法
- コンパイラ・キーワードの概要

NewTable (トリガ・キーワード)

イベントの影響を受ける行または文の新しい値が格納される移行テーブルの名前を指定します。

使用法

新しい値を格納する移行テーブルの名前を指定するには、以下の構文を使用します。

```
Trigger name [ Event = sqlevent, OldTable = oldtable, NewTable = newtable ] {    //implementation }
```

newtable は、このネームスペース内の SQL テーブルの名前です。

詳細

各トリガは、(OldTable キーワードと NewTable キーワードで指定される) 移行テーブルを使用して、イベントの影響を受ける行または文の古い値と新しい値にアクセスします。

既定値

NewTable キーワードの既定値は、NULL です。

関連項目

- トリガ定義
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- トリガの使用法
- コンパイラ・キーワードの概要

OldTable (トリガ・キーワード)

イベントの影響を受ける行または文の古い値が格納される移行テーブルの名前を指定します。

使用法

古い値を格納する移行テーブルの名前を指定するには、以下の構文を使用します。

```
Trigger name [ Event = sqlevent, OldTable = oldtable, NewTable = newtable ] {    //implementation }
```

oldtable は、このネームスペース内の SQL テーブルの名前です。

詳細

各トリガは、(OldTable キーワードと NewTable キーワードで指定される) 移行テーブルを使用して、イベントの影響を受ける行または文の古い値と新しい値にアクセスします。

既定値

既定値は空文字列です。

関連項目

- トリガ定義
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- トリガの使用法
- コンパイラ・キーワードの概要

Order (トリガ・キーワード)

同じ EVENT や TIME に対して複数のトリガが存在する場合に、トリガを起動する順番を指定します。

使用法

EVENT および TIME が同じである他のトリガを基準として、このトリガが起動される順序を指定するには、以下の構文を使用します。

```
Trigger name [ Event = sqlevent, Order = n, Time = time ] {    //implementation }
```

ここで、n は整数です。

詳細

同じ EVENT や TIME に対して複数のトリガが存在する場合に、このキーワードはトリガを起動する順序を指定します。

既定値

既定値は 0 です。

関連項目

- トリガ定義
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- トリガの使用法
- コンパイラ・キーワードの概要

SqlName (トリガ・キーワード)

このトリガに使用する SQL 名を指定します。

使用法

このトリガの既定の SQL 名をオーバーライドするには、以下の構文を使用します。

```
Trigger name [ Event = sqlevent, SqlName = sqlname, Time = time ] {    //implementation }
```

sqlname は SQL 識別子です。

詳細

このトリガを SQL に投影するときに、この名前を SQL トリガの名前に使用します。

既定値

このキーワードを省略すると、SQL トリガの名前は、トリガ定義で指定された triggername になります。

関連項目

- トリガ定義
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- トリガの使用法
- コンパイラ・キーワードの概要

Time (トリガ・キーワード)

トリガをイベントの前に起動するか、後に起動するかを指定します。

使用法

トリガをイベントの前に起動するか、後に起動するかを指定するには、以下の構文を使用します。

```
Trigger name [ Event = sqlevent, Time = time ] {    //implementation }
```

time は以下のいずれかになります。

- AFTER ー このトリガは、イベントの後に起動されます。
- BEFORE ー このトリガは、イベントの前に起動されます。

詳細

このキーワードは、トリガをイベントの前に起動するか、後に起動するかを指定します。

既定値

既定値は BEFORE です。

関連項目

- トリガ定義
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- トリガの使用法
- コンパイラ・キーワードの概要

UpdateColumnList (トリガ・キーワード)

変更されると SQL によるトリガの起動を引き起こす 1 つ以上の列を指定します。TSQL でのみ使用可能です。

使用法

変更されるとトリガが起動される列を指定するには、以下の構文を使用します。

```
Trigger name [ Event = sqlevent, UpdateColumnList = updatecolumnlist ]  {    //implementation }
```

updatecolumnlist は、単一の列名、または括弧で囲んだ複数の列名のコンマ区切りリストのいずれかにします。

詳細

このキーワードは、変更されるとトリガが起動される 1 つ以上の列を指定します。このキーワードは、TSQL でのみ使用可能です。

関連項目

- トリガ定義
- メソッド・ジェネレータとトリガ・ジェネレータの定義
- トリガの使用法
- コンパイラ・キーワードの概要

XData の構文とキーワード

このリファレンスでは、XData ブロックに適用する構文とキーワードについて説明しています。キーワード (クラス属性とも呼ばれる) は、一般にコンパイラに影響を与えます。

XData ブロック定義に関する一般情報へのリンクは、"XData ブロック" を参照してください。

特定のクラス・メンバに適用するすべての構造とキーワードは、目次を参照してください。

クラス定義での XData ブロックの構文

XData ブロックの構造を説明します。XData ブロック定義に関する一般情報へのリンクは、"XData ブロック" を参照してください。

概要

XData ブロックは、データの名前付きユニットであり、クラス定義に置いて、通常はクラスのメソッドで使用します。一般的に、これは適格な XML ドキュメントですが、JSON や YAML など、他の形式のデータで構成することもできます。

詳細

XData ブロックの構造は以下のとおりです。

```
/// description
XData name [ keyword_list ]
{
data
}
```

以下は、この指定の説明です。

- description (オプション) は、クラス・リファレンスでの表示を意図しています。description は既定ではブランクです。"クラス・ドキュメントの作成" を参照してください。
- name (必須) は XData ブロックの名前です。これは、有効なクラス・メンバ名で、かつ他のクラス・メンバ名と重複しないものである必要があります。
- data (オプション) は XData ブロックのペイロードを格納します。XML の場合、これは先頭に XML 宣言がない、整形式のドキュメント (単一のルート要素を持つ) にする必要があります。
- keyword_list (オプション) は、さらに XData ブロックを定義するキーワードのコンマ区切りリストです。

  すべてのキーワードは、"XData の構文とキーワード" を参照してください。

  このリストを省略する場合は角括弧も省略します。

例

```
Class Demo.CoffeeMakerRESTServer Extends %CSP.REST
 {
  Parameter HandleCorsRequest = 1

  XData UrlMap [ XMLNamespace = "http://www.intersystems.com/urlmap" ]
  {
    <Routes>
      <Route Url="/test" Method="GET" Call="test"/>
      <Route Url="/coffeemakers" Method="GET" Call="GetAll" />
      <Route Url="/coffeemaker/:id" Method="GET" Call="GetCoffeeMakerInfo" />
      <Route Url="/newcoffeemaker" Method="POST" Call="NewMaker" />
      <Route Url="/coffeemaker/:id" Method="PUT" Call="EditMaker" />
      <Route Url="/coffeemaker/:id" Method="DELETE" Call="RemoveCoffeemaker"/>
   </Routes>
  }
```

関連項目

- XData ブロックの定義と使用
- クラスの制限

Internal (XData キーワード)

XData ブロックが Internal (クラス・ドキュメントに表示されない) であるかどうかを指定します。クラス・ドキュメントは、現在は XData をまったく表示しません。

使用法

この XData ブロックを Internal として指定するには、以下の構文を使用します。

```
XData name [ Internal ]  { }
```

そのように指定しない場合は、このキーワードを省略し、キーワードの直前に単語 Not を指定します。

詳細

Internal クラスのメンバは、クラス・ドキュメントには表示されません。このキーワードは、ユーザに対してクラスは表示するが、そのすべてのメンバを表示する必要がない場合に役立ちます。

クラス・ドキュメントは、現在は XData ブロックをまったく表示しません。

関連項目

- XData ブロック
- XData ブロックの定義と使用
- コンパイラ・キーワードの概要

MimeType (XData キーワード)

XData ブロックの MIME タイプを指定します。

使用法

XData ブロックの MIME タイプを指定するには、以下のような構文を使用します。

```
XData name [ MimeType = mimetype ] { }
```

mimetype は有効な MIME タイプ (より正式には、インターネット・メディア・タイプ) です。

詳細

このキーワードは、XData ブロックのコンテンツの MIME タイプを指定します。Visual Studio Code の ObjectScript 拡張機能では、次の MIME タイプの色分けを提供しています。

- application/json
- text/html
- text/javascript
- text/css
- application/sql
- text/x-java-source
- application/python
- text/x-python
- text/xml
- application/xml

その他の MIME タイプは色分けされません。

既定値

既定の MIME タイプは `text/xml` です。

関連項目

- XData ブロック
- XData ブロックの定義と使用
- コンパイラ・キーワードの概要

SchemaSpec (XData キーワード)

この XData ブロックを検証できる XML スキーマを指定します。

使用法

この XData ブロックを検証できる XML スキーマを指定するには、以下のような構文を使用します。

```
XData name [ SchemaSpec = "schemanamespaceURL schemaURL" ] { }
```

以下は、この指定の説明です。

- schemanamespaceURL は、スキーマが属するネームスペースの URI です。
- schemaURL は、スキーマ・ドキュメントの URL です。

それらの項目間にはスペース文字があります。二重引用符も使用されています。

詳細

このキーワードは、この XData ブロックを検証できる XML スキーマを指定します。

既定値

このキーワードを省略した場合、XData ブロックはその内容の検証に使用できる XML スキーマを提供しません。

例

```
XData MyXData [ SchemaSpec = "http:///www.person.com http://www.MyCompany.com/schemas/person.xsd" ]
{
}
```

関連項目

- XData ブロック
- XData ブロックの定義と使用
- コンパイラ・キーワードの概要

XMLNamespace (XData キーワード)

XData ブロックが属する XML ネームスペースを指定します。

使用法

XData ブロックが属する XML ネームスペースを指定するには、以下のような構文を使用します。

```
XData name [ XMLNamespace = "namespaceURL" ] { }
```

namespaceURL は XML ネームスペースの URI です。この項目は二重引用符で囲まれています。

詳細

このキーワードは、XData ブロックが属する XML ネームスペースを指定します。

既定値

このキーワードを省略した場合、この XData ブロックの内容はどのネームスペースにも属しません。

例

```
XData MyXData [ XMLNamespace = "http://www.mynamespace.org" ]
{
}
```

関連項目

- XData ブロック
- XData ブロックの定義と使用
- コンパイラ・キーワードの概要

Storage キーワード

このリファレンスでは、クラス定義のストレージ・セクションに適用するキーワードについて説明します。

ストレージの定義に関する一般情報へのリンクは、"ストレージ定義" を参照してください。

特定のクラス・メンバに適用される構造とキーワードの完全なリストは、目次を参照してください。

注釈　このリファレンスでは、ストレージ定義の構文を正式には説明していません。ストレージ定義の概要は、"ストレージ" を参照してください。"ストレージ定義とストレージ・クラス" も参照してください。

DataLocation (Storage キーワード)

このクラスのデータを保存する場所を指定します。

構文

```
<DataLocation>^Sample.PersonD</DataLocation>
```

値

このキーワードの値は、グローバル名です。オプションで、先頭に添え字を付けます。

概要

式は、このクラスのデータを保存する位置です。 通常は、^User.PersonD などのようなグローバル参照です。グローバル参照にも 1 つまたは複数の先頭の添え字を指定できます。例えば、^User.Data("Person") のようになります。

また、独立クラス (親子リレーションシップ内の子クラス) 内のグローバル変数名、またはローカル変数名の位置に {%%PARENT} を使用することもできます。例えば、{%%PARENT}(ChildData) のようになります。%%PARENT は、親の ID によって修飾されたデータ位置 (データ位置に親の ID の添え字を付けたもの) に評価されます。

既定値

<DataLocation> 要素の既定値は、空の文字列です。この場合、既定のデータ位置には ^MyApp.MyClassD を使用します (MyApp.MyClass はクラス名です)。

DefaultData (Storage キーワード)

既定のデータ・ストレージ定義を指定します。

構文

```
<DefaultData>MyData</DefaultData>
```

値

この要素の値は、現在のストレージ定義内の、データ・ストレージ・ノードの名前です。

概要

DATA 定義の名前を指定します。この名前は、クラス・コンパイラのデータ構造ジェネレータによって、以前にアンストアされたすべてのプロパティを置くために使用されます。プロパティが保存可能であるが、DATA 定義にリストされていない場合、プロパティがアンストアされたといいます。

新規の、一時的でないプロパティを永続クラス定義に追加して、これに対するストレージ位置を明示的に定義しない場合、<DefaultData> 要素で指定したストレージ・ノード内で、クラス・コンパイラが、自動的にプロパティのストレージ位置を決定します。

既定値

<DefaultData> 要素の既定値は、空の文字列です。

Final (Storage キーワード)

サブクラスによってストレージ定義が変更されないように指定します。

構文

```
<Final>1</Final>
```

値

この要素の値は、ブーリアン値です。

概要

サブクラスによってストレージ定義が変更されないように指定します。

既定値

〈Final〉 要素の既定値は、False です。

IdFunction (Storage キーワード)

既定のストレージを使用して永続クラスの新しい ID 値を割り当てるのに使用されるシステム関数を指定します。

構文

```
<IdFunction>increment</IdFunction>
```

値

この要素の値は、`increment` ($increment 関数を使用する場合) または `sequence` ($sequence 関数を使用する場合) のいずれかです。

説明

永続クラスが、オブジェクト ID の値の特定に IdKey を使用しない場合、この要素により ID 値を割り当てるのに使用する関数 ($increment 関数、または $sequence 関数) を指定できます。

クラスで $increment 関数を使用する場合、新しい ID は、<IdLocation> 要素で定義されているグローバルの位置に格納されている値をインクリメントすることによって作成されます。

クラスで $sequence 関数を使用する場合、これは、高速データ取り込みでより高速にできるよう、ID のブロックを予約できます。予約されたすべての ID が使用されない場合、ID 番号が連続しないことがあります。後続のデータが取り込まれるときに、これらの欠落番号は埋められることも、埋められないこともあります。副次的作用として、大きな番号の ID を持つクラスのインスタンスが必ずしも小さい番号の ID を持つクラスのインスタンスより新しいとは限らなくなります。さらに、<IdLocation> 要素で定義されたグローバルの位置の値は、現在割り当てられているどの ID にも直接関連しません。

既定値

<IdFunction> 要素の規定値は、クラス定義を使用して作成されたクラスの `increment` です。

<IdFunction> 要素の規定値は、DDL `CREATE TABLE` 文を使用して作成されたクラスの `sequence` です。

IdLocation (Storage キーワード)

ID カウンタの位置を指定します。

構文

```
<IdLocation>^Sample.PersonD</IdLocation>
```

値

この要素の値は、グローバル名です。オプションで、先頭に添え字を付けます。

概要

この要素により、オブジェクト ID 値の割り当てに使用されるカウンタを含むグローバル・ノードを指定できます。

既定では、既定のストレージを使用した永続クラス定義で、このグローバルの位置には、クラスのインスタンスに対して割り当てられた最も大きな ID が含まれます。ただし、これは、<IdFunction> 要素が `increment` に設定され、クラスで IdKey を使用していない場合にのみ適用されます。このグローバルの位置に格納された値は、<IdFunction> 要素が `sequence` に設定されている場合は意味がなく、クラスで IdKey を使用している場合、その位置に値は割り当てられません。

既定値

指定されていない場合、<IdLocation> 要素の値は、クラス・コンパイラによって生成されます。多くの場合、この値は ^MyApp.MyClassD (**MyApp.MyClass** はクラス名) ですが、さまざまな要因に基づいて変わることがあります。永続クラスのグローバル名の詳細は、"グローバル" を参照してください。

IndexLocation (Storage キーワード)

インデックスの、既定のストレージ位置を指定します。

構文

```
<IndexLocation>^Sample.PersonI</IndexLocation>
```

値

この要素の値は、グローバル名です。オプションで、先頭に添え字を付けます。

概要

この要素を使用して、このクラスのインデックスに使用するグローバルのストレージ位置を指定します。これが指定されていない場合、インデックスの位置は ^MyApp.MyClassI です (**MyApp.MyClass** はクラス名です)。

また、各インデックスのストレージを個別に指定することもできます。

既定値

<IndexLocation> 要素の既定値は、空の文字列です。

SqlRowIdName (Storage キーワード)

SQL 内の行 ID に使用する名前を指定します。

構文

```
<SqlRowIdName>IdName</SqlRowIdName>
```

値

この要素の値は、SQL 識別子です。

概要

この要素により、SQL に投影する行 ID (オブジェクト ID) 列の名前を直接指定できます。

既定値

<SqlRowIdName> 要素の既定値は、空の文字列です。

SqlRowIdProperty (Storage キーワード)

SQL RowId プロパティを指定します。

構文

```
<SqlRowIdProperty>prop</SqlRowIdProperty>
```

値

この要素の値は、SQL 識別子です。

概要

この要素は、インターシステムズ社の従来の製品から移行したクラスのみで使用します。

既定値

<SqlRowIdProperty> 要素の既定値は、空の文字列です。

SqlTableNumber (Storage キーワード)

内部の SQL テーブル番号を指定します。

構文

```
<SqlTableNumber>123</SqlTableNumber>
```

値

この要素の値は、テーブル番号です。

概要

この要素は、インターシステムズ社の従来の製品から移行したクラスのみで使用します。

既定値

<SqlTableNumber> 要素の既定値は、空の文字列です。

State (Storage キーワード)

シリアル・オブジェクトに使用するデータ定義を指定します。

構文

```
<State>state</State>
```

値

この要素の値は、このストレージ定義内のデータ定義の名前です。

概要

シリアル (埋め込み) クラスでは、このキーワードを使用して、オブジェクトのシリアル化状態 (オブジェクト・プロパティがシリアル化されたときに、どのように配置するか) を定義するデータ定義を指定します。また、これは、アンストアされたプロパティが既定の構成ジェネレータにより追加される、既定の DATA 定義でもあります。

既定値

〈State〉 要素の既定値は、空の文字列です。

StreamLocation (Storage キーワード)

ストリーム・プロパティの既定のストレージ位置を指定します。

構文

```
<StreamLocation>^Sample.PersonS</StreamLocation>
```

値

この要素の値は、グローバル名です。オプションで、先頭に添え字を付けます。

概要

この要素を使用して、永続クラスのすべてのストリーム・プロパティを保存するのに使用する、既定のグローバルのストレージ位置を指定します。このグローバルのルート位置に格納されている値は、このクラスのストリーム値が格納されるたびにインクリメントされるカウンタです。

各ストリーム・プロパティのストレージを個別に指定することもできます。"ストリーム・プロパティの宣言" を参照してください。

既定値

指定されていない場合、<StreamLocation> 要素の値は、クラス・コンパイラによって生成されます。多くの場合、この値は ^MyApp.MyClassS (**MyApp.MyClass** はクラス名) ですが、さまざまな要因に基づいて変わることがあります。永続クラスのグローバル名の詳細は、"グローバル" を参照してください。

Type (Storage キーワード)

永続性を提供するために使用するストレージ・クラスです。

構文

```
<Type>%Storage.Persistent</Type>
```

値

この要素の値は、クラス名です。

概要

この要素は、このクラスに対して永続性を提供するストレージ・クラスを指定します。

%Storage.Persistent クラスは、既定のストレージ・クラスで、既定のストレージ構造を提供します。

%Storage.SQL クラスは、従来のデータ構造をマップするために使用します。

シリアル (埋め込み) クラスでは、このキーワードは %Storage.Serial で設定する必要があります (新規クラス・ウィザードにより、自動的に設定されます)。

既定値

〈Type〉 要素の既定値は、%Storage.Persistent です。