Embedded Python
オブジェクトアクセス編

InterSystems®
Creative data technology

この資料の主な目標

この資料では、InterSystems IRIS 2021.2 以降に追加された Embedded Pythonを使用して、IRIS の永続クラスの定義方法、オブジェクト操作方法、メソッドの記述と実行方法を習得し、1つのデータに対してSQLでもオブジェクトでもアクセスできる「マルチモデルデータベース」の特徴をご理解いただきます。

- IRISのクラス定義作成練習（VSCode）
- PythonからIRIS内クラスに対するインスタンス操作練習
- language=pythonのメソッド記述と実行練習
  - 引数・戻り値のデータタイプについて
  - ObjectScriptのシステムクラスによくある%Statusの扱い

用語のふりかえり

IRISのネームスペースとデータベースについて

ネームスペースは、仮想の作業環境で使用したいデータベースを指定する論理定義

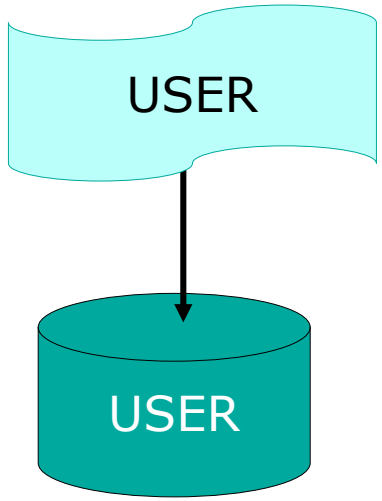
データベースは、データやロジック、クラスやテーブルのスキーマを格納する場所

ネームスペースを特定する

↓

データベースが特定できる

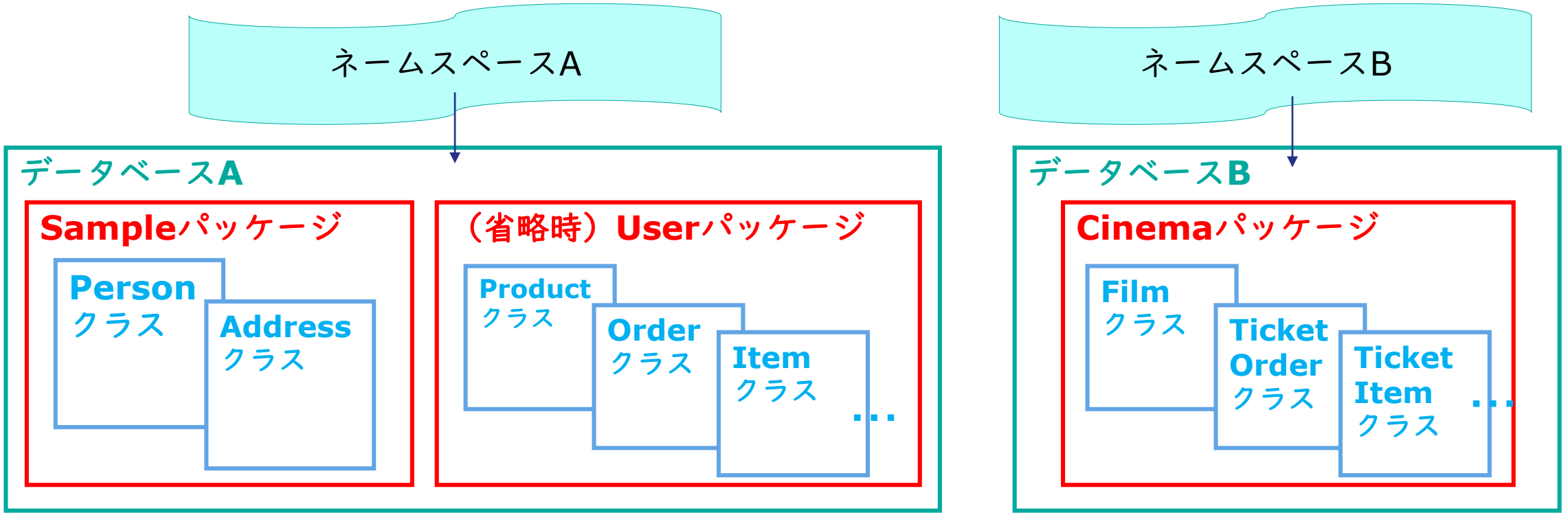
使用したい情報をすべて利用できる



IRISにクラスを作成する場合のルール

クラスは、**パッケージ名.クラス名** で表現します。

パッケージ名は省略できますが、**省略した場合は、システムデフォルトで設定されている User が使用されます**（User.クラス名）。



IRISのクラス定義
（インスタンスを永続化する？しない？を選択できます）

最も簡単なクラスは、メソッドだけを定義するメソッドのコンテナとしたクラスです。

- プロパティを持たないクラスでロジック（メソッド）だけを定義します。
- アプリケーションロジックのユーティリティコードや、ストアドプロシージャ用ロジックを定義するクラスとして使用します。

メソッドの他に**データ要素であるプロパティ**を定義するクラスもあります。

- **データベースに格納する機能を持つクラス（永続クラス＝%Persistentを継承）**
  - 演習で作成します。
- **データベースに格納しないクラス**（メモリ上にインスタンスは作成できるけど、データベースには保存できないクラス＝%RegisteredObjectを継承）
- 永続クラスのプロパティにオブジェクト参照を使用して埋め込むことが前提のクラス（埋め込みクラス＝ %SerialObjectを継承）

メソッドは、特定の動作を実行するためのコードです。

- **メソッドコード中で使用する引数や変数は、プライベートスコープです。**

メソッドの記述には、ObjectScript または **Python** を使用します（Language属性で指定します）。

クラス定義文

Class パッケージ名.クラス名 Extends スーパークラス

```
Class FS.Person Extends %Persistent
```

- 特に何も継承する必要がないときは、Extends以降は不要です。
- 多重継承する場合は、スーパークラスをカンマで区切って指定し、Extends以降を括弧で括ります。
  - 同じ名称の定義が存在する場合は、左に書いたクラスが優先されます。

```
Class FS.Person Extends (%Persistent,%Populate)
```

プロパティ定義文

プロパティは、オブジェクトのデータ要素が**どのような値であるかを定義する設定項目**です。

定義文は以下の通りです。

Property **Name As %String**;

%Stringの正式名称は **%Library.String** クラスです。%Libraryパッケージ以下クラスは非常によく利用するクラスであるため、**Library.** を省略できます。

- **As の後ろにデータタイプ**やプロパティを表現するための**クラス名を指定できます。**
  - データタイプの例：%String、%Integer、%Dateなど
  - オブジェクト参照を行う場合は、参照先クラス名をAsの後ろの指定します。
  - %Stringに独自のチェック項目をつけたしたユーザ定義のデータタイプも指定できます。その場合は、作成したデータタイプクラス名を指定します。
- （ご参考）リスト、リレーションシップ定義など、このほかにもオブジェクトよりの定義も実装できます。

作成する永続クラス：FS.Person

永続クラスを作成し、単一の値を格納するプロパティを定義します。
（Embedded Python SQLアクセス編で作成したテーブルを、クラスとして作成する流れを体験します）

DOB（誕生日）から年齢を算出できるので、Ageプロパティを
計算プロパティとして定義します。

計算プロパティとは？

- アクセス時に計算（処理）結果を返すプロパティの定義
- IRISには、値を保持しないタイプと保持するタイプを選択できます。
  - 演習では**値を保持しないCalculated**キーワードを使用します。
- 値を保持する場合、SqlComputeOnChangeキーワードを利用して、以下の用途で処理を追加できます。
  - 他のプロパティ値が変更されたときに処理させる
  - 新規保存時、更新保存時の時だけ処理させる

FS.Person
Name As %String DOB As %Date Age As %Integer
AgeComputation() Print()

Ageプロパティの定義（計算プロパティ）

指定するキーワードは以下の通りです。

- **Calculated**
  年に1度、年齢が変わるので、アクセスされたときに処理が動くように、値を保持しない常時計算タイプの**Calculated**キーワードを使用します。
- **SqlComputed**
  このキーワードを指定するとAgeプロパティアクセス時に実際の処理を記述したクラスメソッド：**プロパティ名Computation()**が実行されます。

```
Property Age As %Integer [ Calculated, SqlComputed ];
```

プロパティ名Computation() の命名規則でクラスメソッドを作成します。

- 第1引数を用意しデータタイプは**%PropertyHelper**を指定します。第1引数を利用して、処理実行中に他のプロパティ値を取得することができます。 **cols**.getfield("DOB")
- *戻り値*は、Ageプロパティに設定したデータタイプを指定します。
- **Language=python** を指定します。

```
ClassMethod AgeComputation(cols As %PropertyHelper) As %Integer [ Language = python ]
{
}
```

Ageプロパティの定義（計算プロパティ）コードの中身

IRISのデータタイプ：%Dateは、内部形式の数値で表現され、起源日（0）は1840年12月31日であるため、Pythonの先発グレゴリオ暦の数値表現を合わせるため、以下計算式を指定します。

- （本日の日付　-　1840年12月31日　-　誕生日）// 365
- コードの中でクラスの別プロパティ値を取得するには、**cols.getfield(”プロパティ名”)**を使用します。

```
((datetime.date.today().toordinal() - datetime.date(1840,12,31).toordinal() - cols.getfield("DOB")) // 365
```

```
ClassMethod AgeComputation(cols As %PropertyHelper) As %Integer [ Language = python ]
{
    if cols.getfield("DOB")=="":
        return ""
    import datetime
    today=datetime.date.today().toordinal()
    iris0=datetime.date(1840,12,31).toordinal()
    return ((today-iris0-cols.getfield("DOB"))//365)
}
```

演習

1. 永続クラス FS.Person を作成します。
2. プロパティを定義します。
3. Ageプロパティにアクセスした時に実行されるクラスメソッド AgeComputation()を作成します。
4. コンパイルします。

手順詳細は演習補足[VSCode-EmbeddedPython-Object編-演習補足.pdf] をご参照ください。

クラスを利用したプログラミング

クラスの操作には、 **iris モジュール**に含まれる **iris.cls()** を使用します。

- iris モジュールは、IRISにログイン後に起動する Python シェルを使用した場合、インポート済のため明示的に import iris を行わなくても利用できます。

IRISにログインし、Pythonのシェルを起動する方法は以下の通りです。

- IRISのログイン
  - Windowsはターミナルを起動します。
  - Windows以外は、 **iris session インスタンス名** でログインします。
    インスタンス名が IRIS の場合は、 iris session iris でログインできます。
- IRISログイン後のPythonシェル起動方法

```
do ##class(%SYS.Python).Shell()
```

  - do は戻り値を持たないプロシージャ、メソッドを実行するときに使用する ObjectScript のコマンドです。
  - ##class()は ObjectScript でクラスを操作する際に使用する構文で、カッコ内にクラス名を記入します。以降はメソッドを指定します（指定のクラス名、メソッド名は大小文字を区別します）。

永続クラスを操作するメソッド使用方法

インスタンス生成 **_New()**、インスタンス保存 **_Save()**、インスタンスオープン**_OpenId()**、などのクラスメソッドがスーパークラスから提供されます。

利用方法は以下の通りです。

- 新規インスタンス生成

```
変数=iris.cls("パッケージ名.クラス名")._New()
```

- 既存オブジェクトオープン

```
変数=iris.cls("パッケージ名.クラス名")._OpenId(ID番号)
```

- クラスメソッドの実行

```
iris.cls("パッケージ名.クラス名").メソッド名()
```

- インスタンスメソッドの実行（変数objにインスタンスが設定されている場合の例）

```
obj.メソッド名()
```

- オブジェクトのメモリからの開放（変数objにインスタンスが設定されている場合の例）

```
obj="" または del obj
```

永続メソッド

永続クラスに用意されている主なメソッドは以下の通りです。

- メソッドの戻り値にデータタイプ：%Statusが設定されている場合、成功時は 1、失敗時はエラーステータスが返ります。

メソッド	目的	成功時	失敗時
_New()	メモリー上に新規オブジェクトを作成	object	""
_Save()	オブジェクトを保存	1	エラーステータス
_Id()	object IDを返す	object ID	""
_OpenId(*id*)	保存済みオブジェクトを取得し、メモリーに展開する	object	""

ご参考：永続メソッド　その他

メソッド	目的	成功時	失敗時
_DeleteId(*id*)	保存済みオブジェクトを削除	1	エラーステータス
_DeleteExtent()	全ての保存済みオブジェクト（とその関連オブジェクト）を削除	1	エラーステータス
_KillExtent() (開発中のみ使用)	全ての保存済みオブジェクトを削除	1	""
_ClassName(1)	オブジェクトの完全クラス名を表示	クラス名	

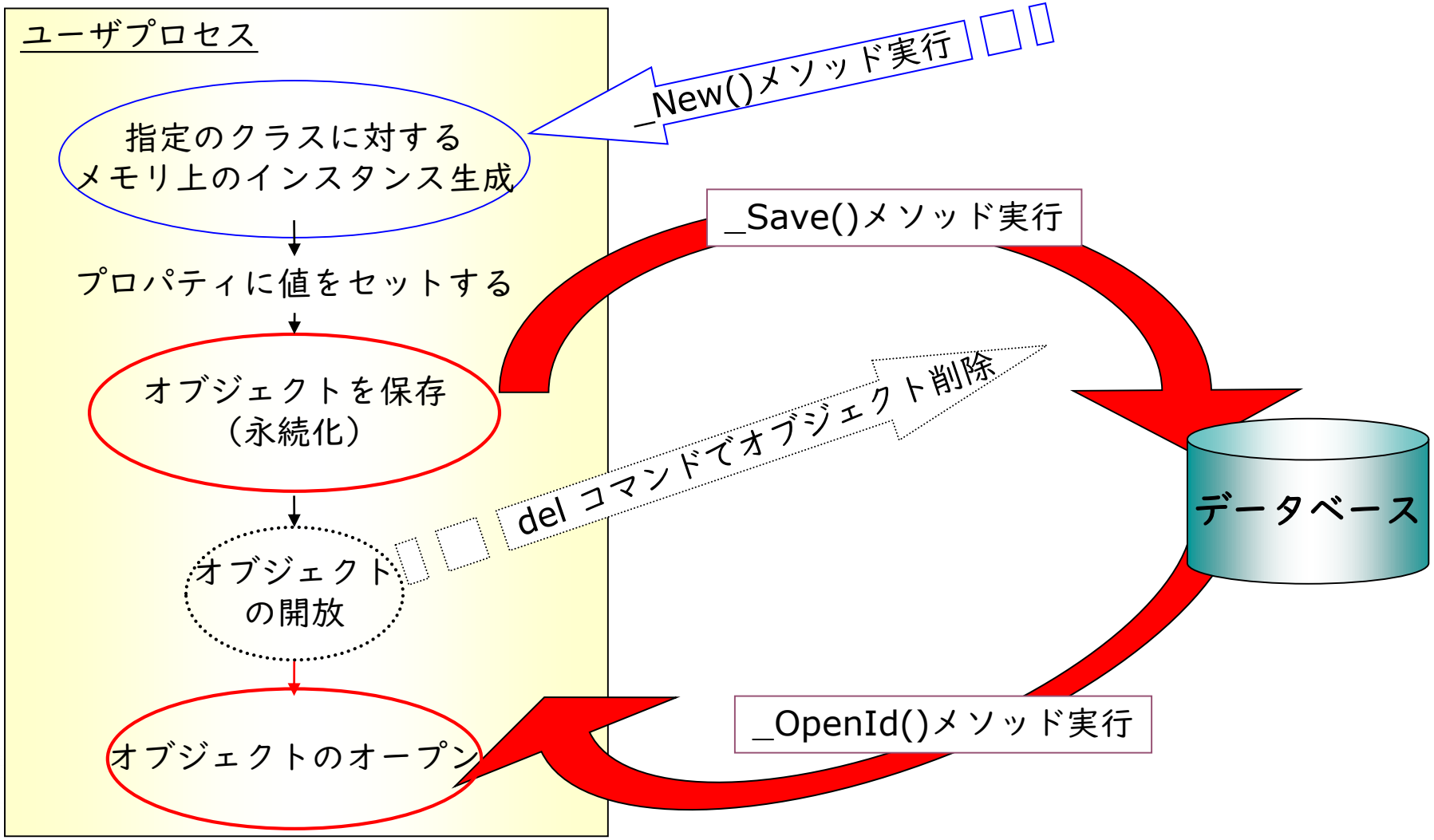
ご参考：日付の操作

データタイプ：%Dateが指定されているIRISのプロパティにPythonからアクセスする場合、以下メソッドを利用すると便利です。

```
#表示形式から内部形式
dob=iris.system.SQL.TODATE(”1999-12-31”,”YYYY-MM-DD”)
#内部形式から表示形式
irisdob=iris.system.SQL.TOCHAR(58073,”YYYY-MM-DD”)
```

演習：FS.Personのインスタンス生成＋保存

インスタンス生成／開放、データベースへの登録／オープンの流れ



演習

1. IRISにログインし、Pythonシェルに切り替えます。
2. FS.Personのインスタンスを生成し、Name、DOBに値を設定します。
3. Ageプロパティにアクセスし、年齢が算出できるかテストします。
4. 作成したインスタンスを保存します。
5. 管理ポータルのSQLメニューを利用して4で保存したデータを確認します。

手順詳細は演習補足[VSCode-EmbeddedPython-Object編-演習補足.pdf] をご参照ください。

永続クラス＝テーブル

インスタンスを永続化すると、システム内でユニークなID番号が付与されます。
＝テーブルのレコードID（ROWID）として利用できます。

オブジェクト指向プログラミング (OOP)	構造化クエリー言語 (SQL)
パッケージ	スキーマ
クラス	テーブル
プロパティ	カラム
メソッド	ストアド・プロシジャ
2つのクラス間のリレーションシップ	外部キー制約、組み込みJOIN
オブジェクト (メモリー上とディスク上)	行 (ディスク上)

SQLを実行する

iris.sql.exec()メソッドを利用してSQLを実行できます。

SELECTの結果は、イテレータか**enumerate()関数**を利用して取得できます。

- DOBプロパティは内部日付の値が登録されているため、Pythonで入出力する場合は、表示形式⇔内部形式の変換が必要です。

```
>>> rs=iris.sql.exec("SELECT * FROM FS.Person")
>>> print(rs.__next__())
['1', 22, 58073, '山田太郎']
>>> reco=rs.__next__()
>>> iris.system.SQL.TOCHAR(reco[2],"YYYY-MM-DD")
'2003-02-03'
```

```
>>> rs=iris.sql.exec("SELECT * FROM FS.Person")
>>> for cn,reco in enumerate(rs):
...  print(f"名前:{reco[3]}、誕生日：{iris.system.SQL.TOCHAR(reco[2],'YYYY-MM-DD')}")
...
名前:山田太郎、誕生日：1999-12-31
名前:鈴木花子、誕生日：2003-02-03
```

引数のあるSQLを実行する方法

- 引数が入る場所の置き換え文字として **?** を使用し、**iris.sql.prepare()**メソッドを使用して、SQL実行の準備をします。

```
rs=iris.sql.prepare(”INSERT INTO FS.Person (Name,DOB) VALUES(?,?)”)
```

- SQLの実行には、**execute()**メソッドを使用します。

```
dob=iris.system.SQL.TODATE(”1999-12-31”,”YYYY-MM-DD”)
rs=iris.sql.execute(”山田太郎”,dob)
```

演習

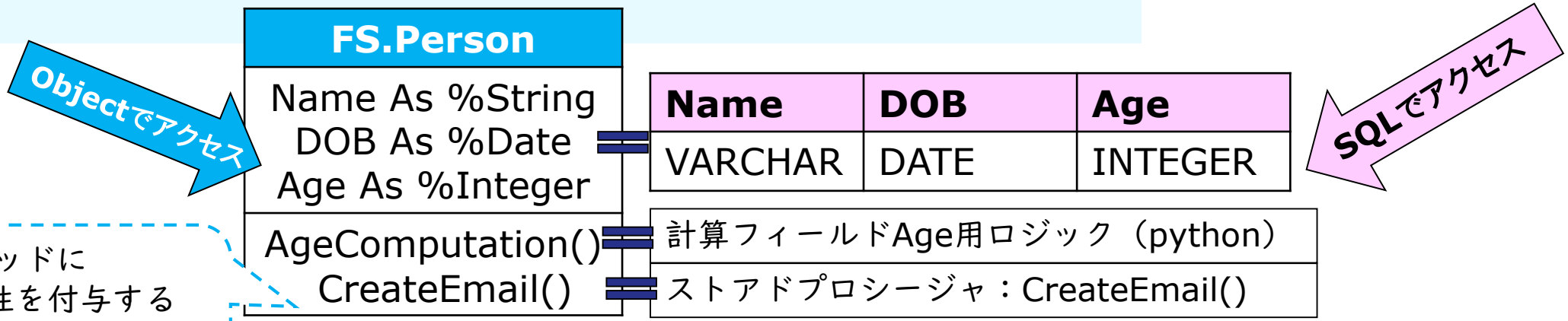
作成した永続オブジェクトに対して、SQLでアクセスしてみます。

1. iris.sql.exec()を利用する
2. iris.sql.prepare()を利用する

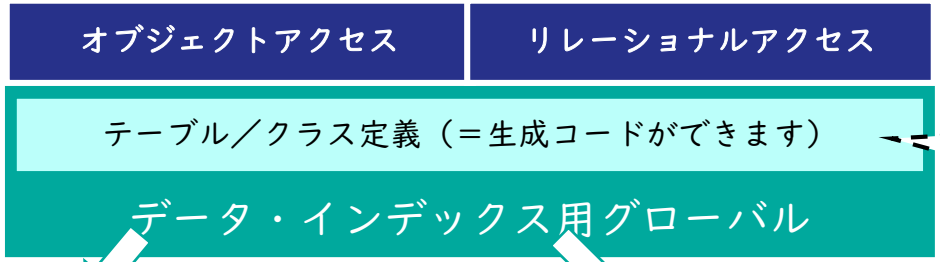
手順詳細は演習補足[VSCode-EmbeddedPython-Object編-演習補足.pdf] をご参照ください。

永続クラス定義＝テーブル定義
どちらも使えます（マルチモデルの利点）

```
ClassMethod CreateEmail(uid As %String) As %String [ Language = python, SqlProc ]
{
    if uid==None:
        return ""
    return uid+"@mail.com"
}
```



クラスメソッドに
SqlProc属性を付与する
とストアドプロシージャ
としても利用できます。



コンパイルによってユーザ
コードを含む「生成コード」
が作成されます。

```
1:      ^FS.PersonD    = 2
2:      ^FS.PersonD(1) = $lb("","山田太郎",58073)
3:      ^FS.PersonD(2) = $lb("","鈴木花子",59203)
```

```
1:      ^FS.PersonI("NameIdx"," 山田太郎",1) = ""
2:      ^FS.PersonI("NameIdx"," 鈴木花子",2) = ""
```

補足：オブジェクト／レコードとストレージ定義

FS > Person.cls > FS.Person > Default

```
1   Class FS.Person Extends %Persistent
2   {
3
4   /// 人の名前
5   Property Name As %String;
6
7   Property DOB As %Date;
8
9   Property Age As %Integer [ Calculated,
10
```

```
1:      ^FS.PersonD    = 2
2:      ^FS.PersonD(1) = $lb("","山田太郎",58073)
3:      ^FS.PersonD(2) = $lb("","鈴木花子",59203)
```

AgeはCalculated属性を付与したため、
値を保持していないことがわかります。

```
21  Storage Default
22  {
23  <Data name="PersonDefaultData">
24  <Value name="1">
25  <Value>%%CLASSNAME</Value>
26  </Value>
27  <Value name="2">
28  <Value>Name</Value>
29  </Value>
30  <Value name="3">
31  <Value>DOB</Value>
32  </Value>
33  </Data>
34  <DataLocation>^FS.PersonD</DataLocation>
35  <DefaultData>PersonDefaultData</DefaultData>
36  <IdLocation>^FS.PersonD</IdLocation>
37  <IndexLocation>^FS.PersonI</IndexLocation>
38  <StreamLocation>^FS.PersonS</StreamLocation>
39  <Type>%Storage.Persistent</Type>
40  }
```

メソッドの引数と戻り値のデータタイプ

IRISのクラス定義に用意するメソッドの引数、戻り値のデータタイプについては、IRISが提供するデータタイプ（%String、%Date、%Integerなど）やユーザ定義クラスを指定します。

IRISのメソッドに可変長引数を渡す場合、**可変長引数用の記述で引数を定義します。**

- メソッドの引数に**ドットを3つ ...** 追加することで可変長の引数を定義できます。
- language=pythonのメソッドで受けた引数は、タプルとして受け取ります。

```
>>> iris.cls("FS.Utils").KahenPython(*["これは","リスト","です"])
('これは', 'リスト', 'です')
>>> iris.cls("FS.Utils").KahenPython(*("これは","タプル","です"))
('これは', 'タプル', 'です')
>>> iris.cls("FS.Utils").KahenPython(1,2,3,4,5)
(1, 2, 3, 4, 5)
```

```
ClassMethod KahenPython(input... As %String) [ Language = python ]
{
    print(input)
}
```

Pythonのコレクション（リスト／タプル／辞書）などをデータタイプに指定する場合

IRISは、Pythonのコレクション（リスト、辞書、タプルなど）やTrue／False／None に対応するデータタイプを持っていないため、引数や戻り値のタイプを指定するときは、**%SYS.Python**を指定してください。

```
ClassMethod listtest(list As %SYS.Python) As %SYS.Python [ Language = python ]
{
    print(f"入力されたデータはのタイプは:{type(list)} です")
    if len(list)==2:
        return True
    else:
        return False
}
```

```
>>> listA=["テスト","リストです","要素は3個あります"]
>>> ret=iris.cls("FS.Utils").listtest(listA)
入力されたデータはのタイプは：<class 'list'> です
>>>
>>> if ret==True:
...  print("リストの要素は2つでした")
... else:
...  print(f"リストの要素は{len(listA)}でした")
...
リストの要素は3でした
>>>
```

演習
インスタンスメソッド／クラスメソッドの作成

FS.Personクラスにインスタンスメソッドとクラスメソッドを追加します。

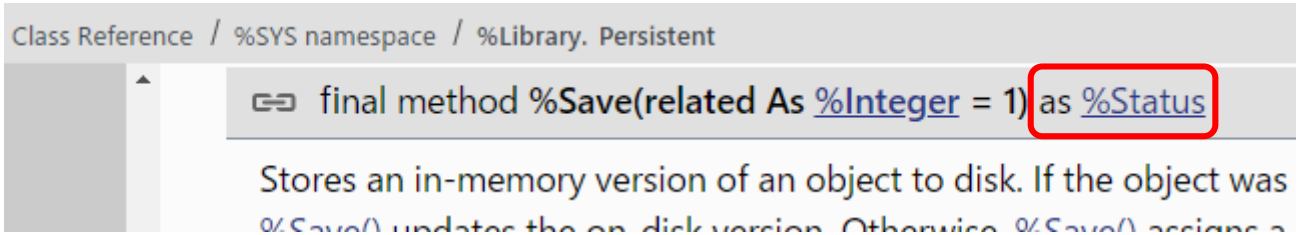
インスタンスメソッドの定義は **Method**、クラスメソッドの定義は **ClassMethod** です。

- インスタンスメソッドはインスタンスが存在している状態で実行できるメソッドであるため、SQLでは実行できません。
- クラスメソッドはインスタンスに依存せず実行できるメソッドであるため、**SqlProc属性**を用意しています。この属性を付けると、クラスメソッドとしてもストアドプロシージャとしても実行できます。
- コードをPythonで記述するために、**language属性にpythonを指定**します。

手順詳細は演習補足[VSCode-EmbeddedPython-Object編-演習補足.pdf] をご参照ください。

IRISのデータタイプ：%Statusについて

インスタンスを保存するときに使用する _Save()メソッドの戻り値は、データタイプ：%Statusが定義されています（クラスリファレンス参照）。



_Save()メソッド実行時、全プロパティに対する検証が自動的に実行されます。

検証が成功し正常に保存できた時は1を返し、失敗した場合はエラーステータスを戻り値に返すため、
_Save()メソッドの実行がエラーだったとしても、例外は発生しません。

%Statusを戻すメソッドを実行する場合、エラーステータスが戻ってきていないか確認する必要があり、Pythonでは、**iris.check_status()**メソッドを使用して確認します。

エラーステータスの場合、 **iris.check_status()**メソッド実行時に**RuntimeError**が発生します。

詳細は、演習でご確認ください。

演習補足[VSCode-EmbeddedPython-Object編-演習補足.pdf] をご参照ください。

ここまでの流れで確認できたこと

- IRIS の永続クラスの定義方法、オブジェクト操作方法を確認できました。
- 永続クラス＝テーブル であり、1つのデータに対してSQLでもオブジェクトでもどちらからでもアクセスできる「マルチモデルデータベース」の特徴を確認できました。
- IRISサーバ側でPythonを使ってプログラミングができることを確認できました。

その他情報について

コミュニティにも利用例が掲載されています。

- 1対多のリレーションシップを使った例（レシートの中身をIRISに登録する例）
- Excel のデータを IRIS グローバルに格納する方法

最新情報については、以下タグにアクセスしてください。

- #Embedded Python

より良い方法を発見した場合、エラーが出た場合、使い方が不明な場合は、ぜひコミュニティへ投稿してください！