

Embedded Python

オブジェクトアクセス編

演習補足資料

(IRIS 2022.1 ベース)

V1.0



目次

1.	はじめに	3
2.	ObjectScript エクステンションの使い方	4
(1)	ObjectScript エクステンションのインストール	4
(2)	IRIS へ接続する	5
3.	Embedded Python の使い方	10
(1)	クラス定義の作成	10
(2)	計算プロパティ：Age の設定	12
(3)	データの登録	14
(4)	データを確認する（管理ポータル）	17
(5)	SQL でアクセス	19
(6)	インスタンスメソッドの作成と実行	22
(7)	クラスメソッドの作成と実行	23
(8)	エラー発生時の動作	25

図表目次

図 1	VSCode：InterSystems ObjectScript Extension Pack の選択	4
図 2	VSCode：Workspace の settings.json の作成	5
図 3	settings.json：接続先設定	6
図 4	IRIS Web サーバポート番号の確認（管理ポータルの概要ページ）	7
図 5	IRIS 接続時のパスワード入力欄	9
図 6	接続情報の更新／管理ポータルやクラスリファレンスへのリンク	9
図 7	FS.Person クラスの新規作成（Person.cls の作成）	10
図 8	Python シェルへの切り替え	15
図 9	管理ポータル：SQL メニューを開く／ネームスペースの切り替え方	17
図 10	管理ポータル SQL メニュー：FS.Person テーブルの確認	18
図 11	管理ポータル：SQL メニュー データの確認	18
図 12	ご参考：CREATE FUNCTION で定義した関数	24

1. はじめに

この資料では、VSCode を使って InterSystems IRIS または InterSystems IRIS for Health（以降、IRIS）で開発を行うために必要な ObjectScript エクステンションの使い方について解説します。

また、ObjectScript エクステンションを使用して永続クラスを作成し、Embedded Python を使用してインスタンス生成、データベースへの保存、データの確認、SQL 実行方法を解説します。

具体的には以下の流れで解説します。


- ・ ObjectScript エクステンションの使い方について
 - (1) ObjectScript エクステンションのインストール
 - (2) IRIS へ接続する
- ・ Embedded Python の使い方
 - (1) クラス定義の作成
 - (2) 計算プロパティ Age の設定
 - (3) データの登録
 - (4) データを確認する（管理ポータル）
 - (5) SQL でアクセス
 - (6) インスタンスの作成と実行
 - (7) クラスメソッドの作成と実行
 - (8) エラー発生時の動作
 - (9) 補足

以降の説明では、USER ネームスペースに接続した状態での例を記述しています。

USER ネームスペース以外を利用されている場合は、利用中ネームスペース名に置き換えてご利用ください。

2. ObjectScript エクステンションの使い方

(1) ObjectScript エクステンションのインストール

VSCode をインストールしたら、Extension の追加を行うため  をクリックします。

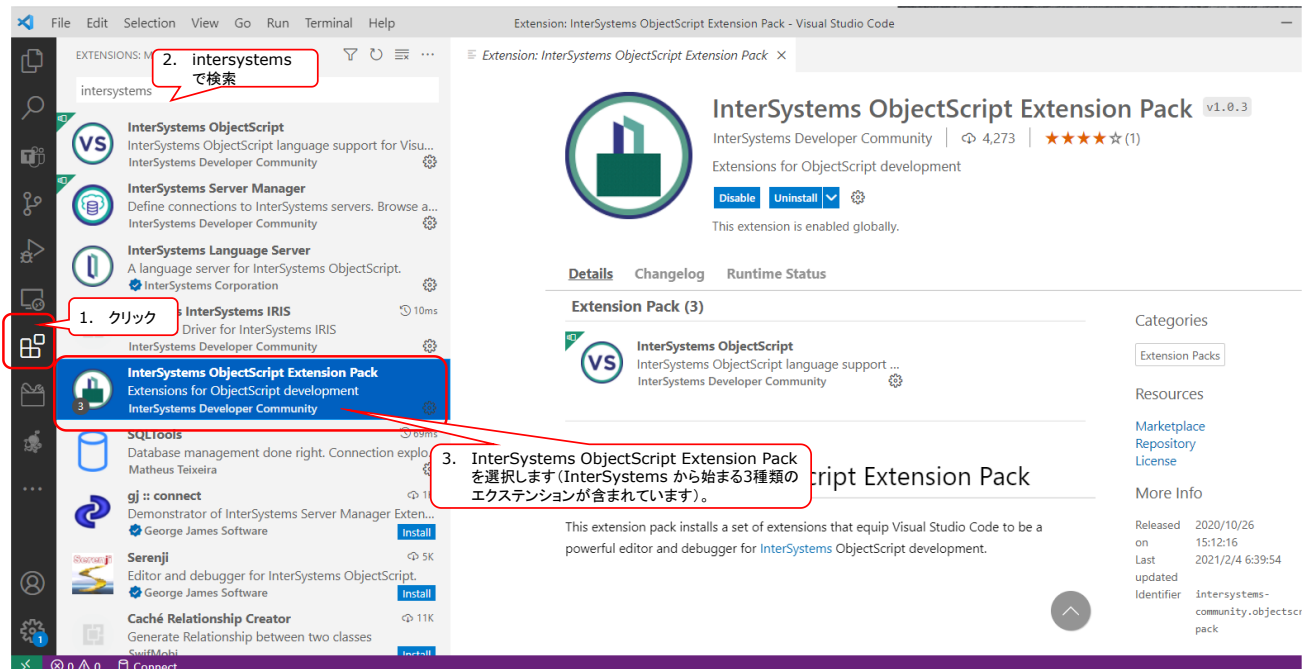
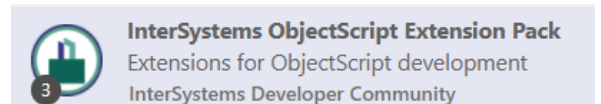



図 1 VSCode : InterSystems ObjectScript Extension Pack の選択

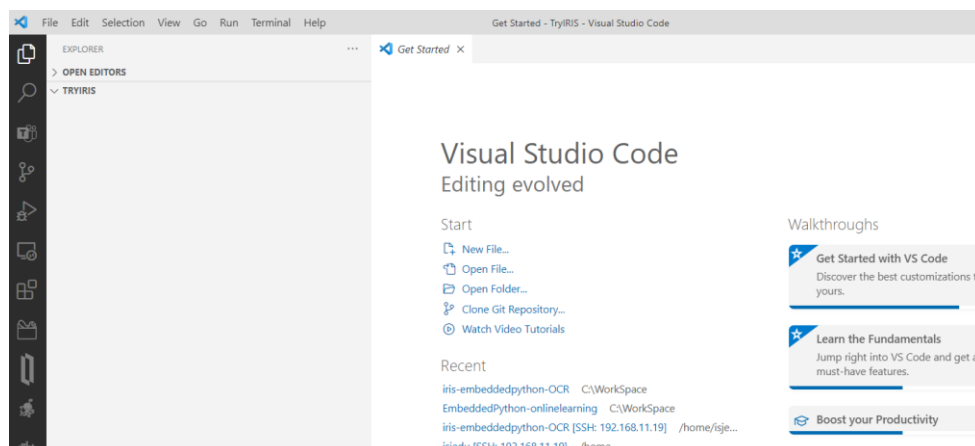
InterSystems から始まる Extension が 3 種類ありますが、「InterSystems ObjectScript Extension Pack」のインストールですべてインストールできます。



インストールが完了すると、左端に  のアイコンが表示されます。

準備が完了したら、任意のディレクトリでワークスペースを作成し、VSCode を作成したワークスペースに移動してください。

例は、C:\¥Workspace¥TryIRIS を作成し、移動した状態の図です。



(2) IRIS へ接続する

(1)で作成したワークスペースに移動した状態で、VSCode のメニューバーから **File > Preferences > Settings** を選択します。

例では、Workspace に対して設定する settings.json を編集するため、「Workspace」を選択しています（図解の 2）。

IRIS の接続情報は [InterSystems Server Manager] を使用するので、フィルタ欄で「intersystems」と記入し、絞り込みます（図解の 3）。

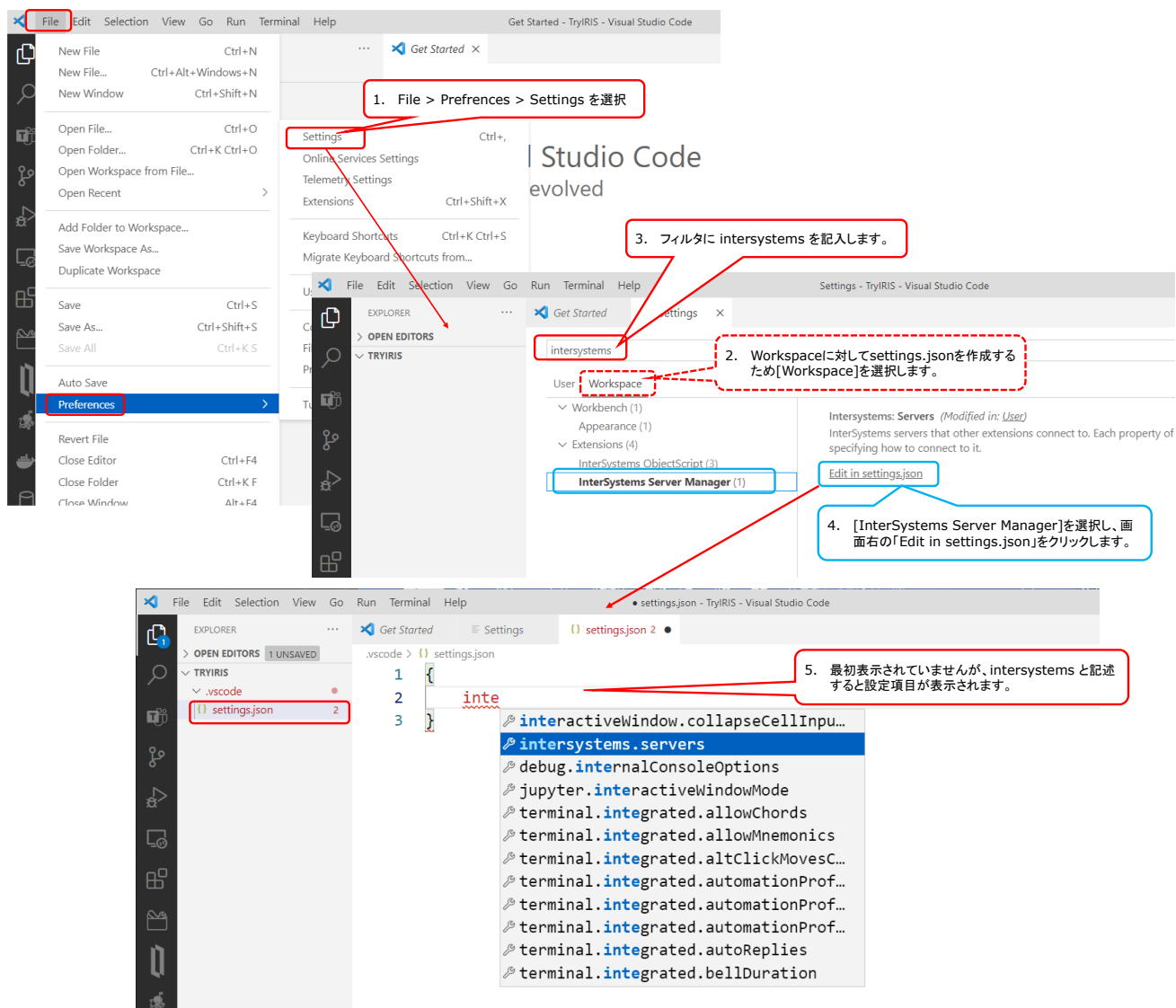


図 2 VSCode : Workspace の settings.json の作成

[InterSystems Server Manager]を選択後、[Edit in settings.json]をクリックすると、開いている Workspace に settings.json が追加されます（図の 5）。

settings.json で [intersystems.servers] と記述すると、設定サンプルが表示されます。既存のリストを修正して利用することもできますが、例では、接続情報を新規に追加する手順でご紹介します

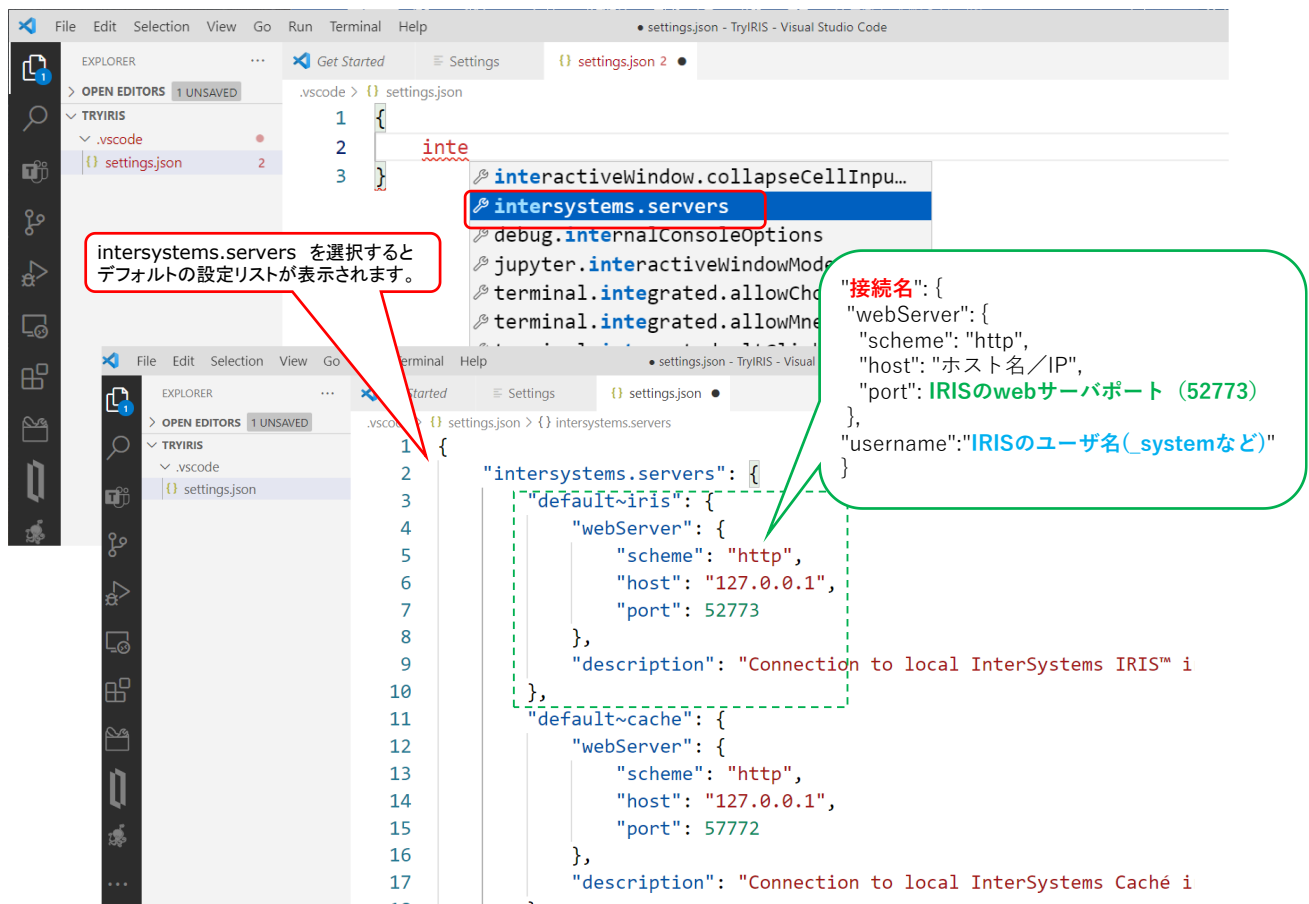


図 3 settings.json : 接続先設定

VSCoDe から IRIS へは、REST を利用してアクセスしています。

以下例は、接続名 **test** の設定で、127.0.0.1 : Web サーバポート **52773** を使用する IRIS へ **SuperUser** ユーザでアクセスします。

```

"intersystems.servers": {
  "test": {
    "webServer": {
      "scheme": "http",
      "host": "127.0.0.1",
      "port": 52773
    },
    "username": "SuperUser"
  },
}

```

補足：ウェブサーバポートや事前定義ユーザのパスワードについて

IRIS のウェブサーバポート番号は、管理ポータル「概要」ページをご覧ください。管理ポータルを開いた時のアドレスバーをご確認ください。

Web サーバポート番号の確認は以下の通りです。

Webサーバポート

概要のリンクをクリックすると、IRISの基本情報を確認できます。

概要

サーバ 68cf1580ad1e ネームスペース %SYS 変更 ユーザ SYSTEM ライセンス先 InterSystems IRIS Community インスタンス IRIS

システム概要

バージョン	IRIS for UNIX (Ubuntu Server LTS for x86-64 Containers) 2021.2 (Build 651U) Mon Jan 31 2022 17:59:03 EST
構成	/usr/irissys/iris.cpf
データベースキャッシュ(MB)	974
ルーチンキャッシュ (MB)	96
ジャーナルファイル	/usr/irissys/mgr/journal/20220307.001
スーパーサーバ・ポート	1972
ウェブサーバポート	52773
ライセンスサーバアドレス/ポート	/
ライセンス先	InterSystems IRIS Community
クラスタサポート	このシステムはクラスタの一部ではありません
ミラーリング	このシステムはミラーメンバではありません
システム開始日時	2022-03-07 11:36:49
暗号化キー識別子	利用可能ではありません。暗号化は有効になっていません。
NLSロケール	JPUW
このセッションの優先言語	日本語

VSCodeからIRISへ接続するときのREST接続に使用するウェブサーバポート番号

図 4 IRIS Web サーバポート番号の確認（管理ポータルの概要ページ）

IRIS には事前定義ユーザとして `_SYSTEM` や `SuperUser` が用意されています。

セキュリティ設定を最小でインストールした場合、また、コンテナを利用している場合、事前定義ユーザに対する初期パスワードとして `SYS`（大文字）が設定されています。¹

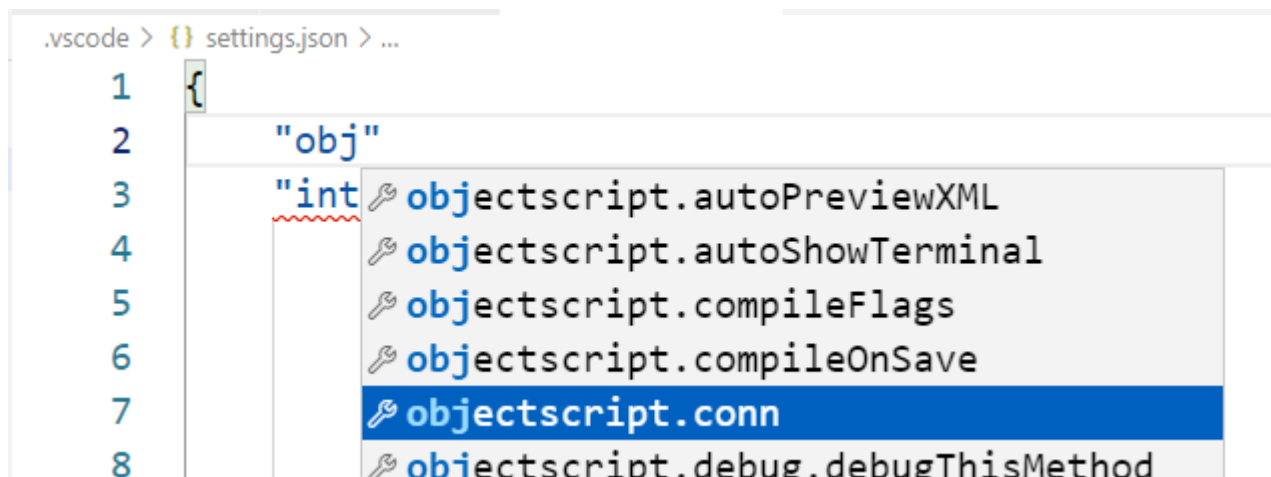
接続時に使用するユーザ名とパスワードをご確認ください。

資料の例では、事前定義ユーザ `SuperUser` を利用してアクセスする例をご紹介します。

¹ コンテナ利用時は初回アクセス時にパスワード変更画面が表示されます。

次に、作成したサーバ名（図例では test）を利用して、IRIS に接続します。

settings.json で、“objectscript.conn” を追加します。



```
"objectscript.conn": {
  "active": true,
  "server": "test",
  "ns": "USER",
},
```

上記指定では、ネームスペース：USER に接続します。他のネームスペースに接続する場合は、ネームスペース名を変更すれば切り替わります。

"server"： には、“intersystems.servers” で作成したサーバ名を指定します（例は以下）。

```
"intersystems.servers": {
  "test": {
    "webServer": {
      "scheme": "http",
      "host": "127.0.0.1",
      "port": 52773
    },
    "username": "SuperUser"
  },
},
```

作成が完了したら保存します。

VSCode の ObjectScript エクステンションのロゴをクリックします。

画面上部にパスワード入力欄が表示されるので SuperUser ユーザに対するパスワードを入力します。

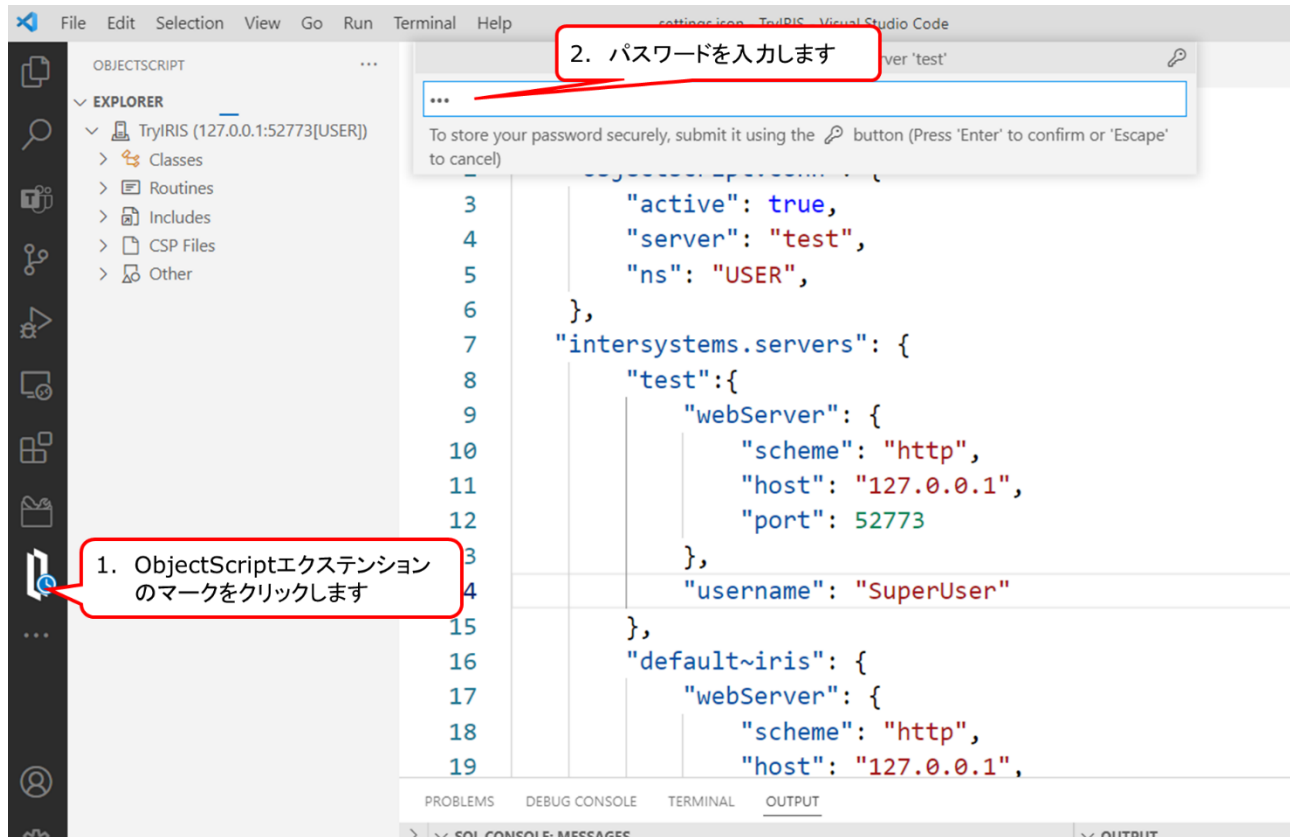


図 5 IRIS 接続時のパスワード入力欄

VSCode の画面下部のバーに



のような表示が現れます。

127.0.0.1:52773[USER] の部分をクリックすると、画面上部に以下のメニューが表示されます。

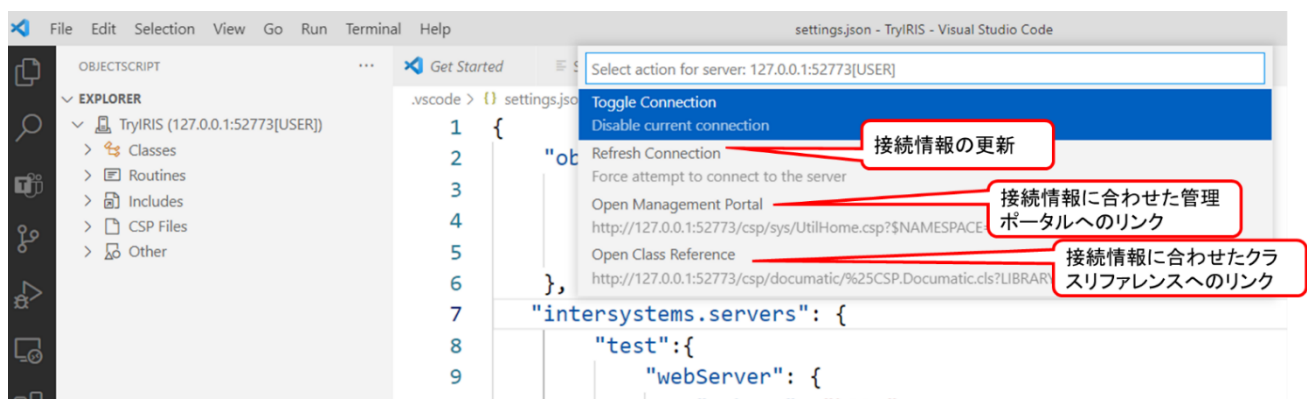


図 6 接続情報の更新／管理ポータルやクラスリファレンスへのリンク

settings.json を修正した場合などは、[Refresh Connection]をクリックすることで、最新の接続情報を使用することができます。

3. Embedded Python の使い方

(1) クラス定義の作成

次に、IRIS にクラス定義を作成します。

IRIS のクラス定義はパッケージ名が必須であるため、最初にパッケージ名のフォルダを用意します。

次にクラス名と同じ名前のファイルを作成します。

クラスを作成する場合、拡張子 **.cls** を使用します。

ファイル名 **Person.cls** を作成します。

図例は以下の通りです。

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

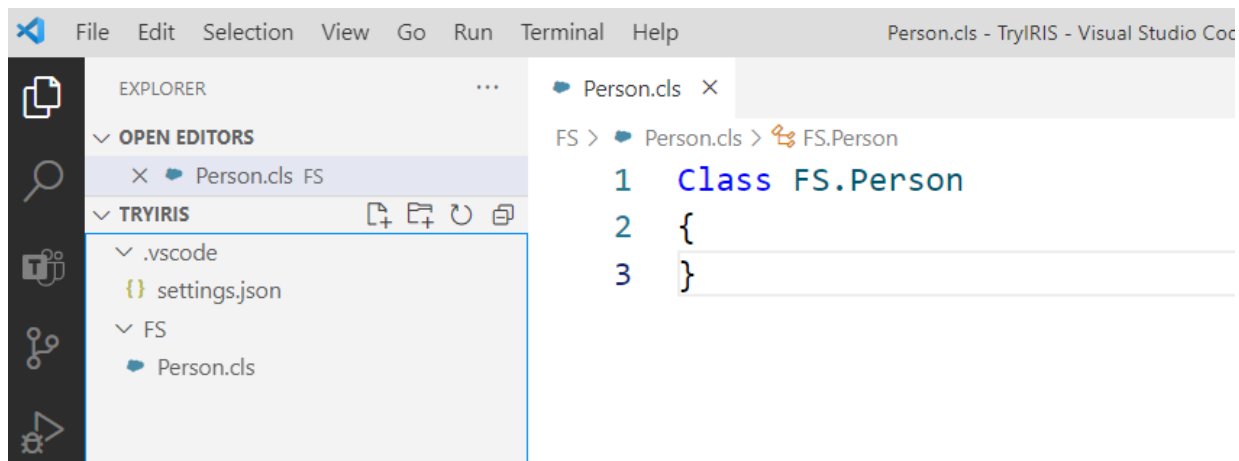


図 7 FS.Person クラスの新規作成 (Person.cls の作成)

Person.cls を作成すると上図のようなクラス定義が準備されます (この時点では何も継承していないクラスが作成されます)。

今回は、クラス定義に沿って作成したインスタンスを永続化したいので、

システムクラス %Library.Persistent クラス²を継承します。

継承する場合は、**Extends** の右側にスーパークラス名を記述します (**Extends** の後ろに 1 つスペースを入れると候補が出るので、候補から %Persistent を選択することもできます)。

クラス名、プロパティ名、メソッド名などは大小文字を区別しますので気を付けて記述してください。

例) **Class FS.Person Extends %Persistent**
{
}

² %Library パッケージはデータタイプやクラスの構築基盤になるスーパークラスが含まれているパッケージで非常によく利用するため、パッケージ名が省略できます。永続クラスにする場合の継承する %Library.Persistent は、%Persistent に省略できます。

続いて、プロパティを追加します。人の名前を設定する Name プロパティを%String³で、誕生日を設定する DOB プロパティを%Date、年齢を設定する Age プロパティを%Integer で設定します。プロパティ定義文は以下の通りです。

Property プロパティ名 **As** データタイプまたはクラス名;

プロパティ以外にもメソッドの引数や戻り値のデータタイプの指定に **As** が登場します。

As の後ろはデータタイプやクラス名を指定する、で覚えてください。

定義例は以下の通りです。

```
Class FS.Person Extends %Persistent
{
  ///人の名前
  Property Name As %String;
  Property DOB As %Date;
  Property Age As %Integer;
}
```

定義文の直前に置いた /// の行は定義の説明文で、クラスリファレンスで参照できます。

³ %String など、よく利用するデータタイプは%Library パッケージ以下のクラスで、%String、%Integer、%Date と省略できます。

(2) 計算プロパティ：Age の設定

Age プロパティは、DOB プロパティから算出できるため、計算プロパティとして設定を追加します。プロパティに以下の属性を追加します。

Calculated	常時計算タイプ。値を保持しません。
SqlComputed	プロパティ名 Computation() メソッドを実行させる場合指定します。

追加後の定義は以下の通りです。

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

次に、Age プロパティにアクセスした時に動作する AgeComputation() メソッドをクラスメソッドとして作成し、使用言語に Python を指定します。

定義は、以下の通りです。

メソッド名	プロパティ名 Computation() の命名規則があるので、AgeComputation() を定義します。
引数	実行時、DOB などインスタンスのプロパティ値を取得するときに使用します。データタイプに %PropertyHelper を指定します。
戻り値	Age プロパティのデータタイプを指定します。(%Integer)

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

DOB 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。

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

コードは以下の通りです。記述ができたなら Ctrl+s で保存 + コンパイルを行います。

```
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)
}
```

作成が完了したら、Ctrl+S でファイルを保存します。

```

FS > Person.cls > FS.Person > AgeComputation
1  Class FS.Person Extends %Persistent
2  {
3
4  /// 名前
5  Property Name As %String;
6
7  Property DOB As %Date;
8
9  Property Age As %Integer [ Calculated, SqlComputed ];
10
11  Debug this method
12  ClassMethod AgeComputation(cols As %PropertyHelper) As %Integer [ Language = python ]
13  {
14      if cols.getfield("DOB")=="":
15          return ""
16      import datetime
17      today=datetime.date.today().toordinal()
18      iris0=datetime.date(1840,12,31).toordinal()
19      return ((today-iris0-cols.getfield("DOB"))//365)
20  }

```

ObjectScriptを選択します

ObjectScript

06/14/2022 12:31:53 に修飾子 'cuk' でコンパイルを開始しました。
 クラスのコンパイル中 FS.Person
 テーブルのコンパイル中 FS.Person
 ルーチンのコンパイル中 FS.Person.1
 コンパイルが正常に終了しました (所要時間: 0.132秒)。

出力欄で ObjectScript を選択すると 保存時にコンパイルが開始されていることを確認できます。

(3) データの登録

クラス定義が作成できたので、データを作成し保存します。

Windows を利用されている場合は、ターミナルを起動します。Windows 以外の場合⁴は、IRIS にログインします。

ログイン後、プロンプトに USER と表示されていれば、ログイン成功です⁵。

Python のシェルに切り替えるため、%SYS.Python クラスの Shell() メソッド（クラスメソッド）を実行します。

ObjectScript でクラスメソッドを実行する場合、##class() 構文を使用し、括弧内にパッケージ名.クラス名を指定します。また、戻りがないプロシージャやメソッドを実行する場合、DO コマンドを利用します。

例) Do ##class(%SYS.Python).Shell()

実行例は、次ページをご参照ください。

⁴ iris session インスタンス名（または構成名）を実行します。

iris session の引数はインストール時指定のインスタンス名（構成名）です。インスタンス名が不明な場合は iris list を打つと確認できます。以下の例の場合は IRIS がインスタンス名です。

```
irisowner@dc47786c4ca9:~$ iris list
```

```
Configuration 'IRIS' (default)
```

```
directory: /usr/irissys/
```

```
versionid: 2021.2.0.649.0
```

```
datadir: /data/config/
```

```
conf file: iris.cpf (SuperServer port = 1972, WebServer = 52773)
```

```
status: running, since Fri Feb 4 10:32:13 2022
```

```
state: warn
```

```
product: InterSystems IRISHealth
```

```
irisowner@dc47786c4ca9:~$
```

iris session 実行時、直接ネームスペースを指定してログインする方法もあります。

iris session IRIS **-U TRAINING**

（-U の引数にネームスペース名を指定します。USER ネームスペースを使用する場合は不要です。）

⁵ USER ネームスペース以外を利用する場合は、ネームスペースを移動します。ネームスペースの移動には、**特殊変数 \$namespace** を使用し、移動したいネームスペース名を二重引用符で括り、設定します。変数に値を割り当てるには、ObjectScript の SET コマンドを使用します。なお、**ネームスペース名、コマンドや特殊変数、関数は大小文字の区別がありません。**

例) set \$namespace = "TRAINING"

ターミナルを開き Python シェルに切替た例は以下の通りです。

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

Python 3.9.10 (tags/v3.9.10:f2f3f53, Jan 17 2022, 15:14:21) [MSC v.1929 64 bit
(AMD64)] on win32
Type quit() or Ctrl-D to exit this shell.
>>>
```

図 8 Python シェルへの切り替え

次に、作成した永続クラスのインスタンスを生成し、プロパティに値を割り当てます。

iris モジュールの **cls()** クラスに パッケージ名.クラス名を二重引用符で括り 指定し、**__New()** メソッドを使用してインスタンスを生成します。

DOB プロパティは、IRIS の内部日付の値を設定する必要があるため、日付操作ライブラリメソッド (iris.system.SQL.TODATE("1999-12-31","YYYY-MM-DD")) を使用します。

```
>>> p=iris.cls("FS.Person").__New()
>>> p.Name="山田太郎"
>>> p.DOB=iris.system.SQL.TODATE("1999-12-31","YYYY-MM-DD")
>>> p.Age
22
```

DOB プロパティに値を設定した後で、Age プロパティにアクセスしてみると、年齢を算出していることを確認できます。

最後に、**__Save()** メソッドを使用してインスタンスを永続化します。成功すると 1 が返ります。

```
>>> st=p.__Save()
>>> print(st)
1
```

<メモ>

IRIS に定義したクラス名、メソッド名、プロパティ名、パッケージ名などは、大小文字を区別します。Python からインスタンス生成時にクラス名やメソッド名の大小文字を間違えると、以下のエラーメッセージを出力します。

```
>>> p=iris.cls("SAMPLE.PERSON")._New()
```

Traceback (most recent call last):

File "<input>", line 1, in <module>

RuntimeError: iris.class: error finding class

```
>>> p=iris.cls("Sample.Person")._new()
```

Traceback (most recent call last):

File "<input>", line 1, in <module>

AttributeError: Property _new not found in object of type iris.FS.Person

```
>>>
```


(4) データを確認する（管理ポータル）

データを 1 件登録できたかどうか、まずは管理ポータルを利用して確認します。

管理ポータル > システムエクスプローラ > **SQL** を開きネームスペース **USER** に接続します。

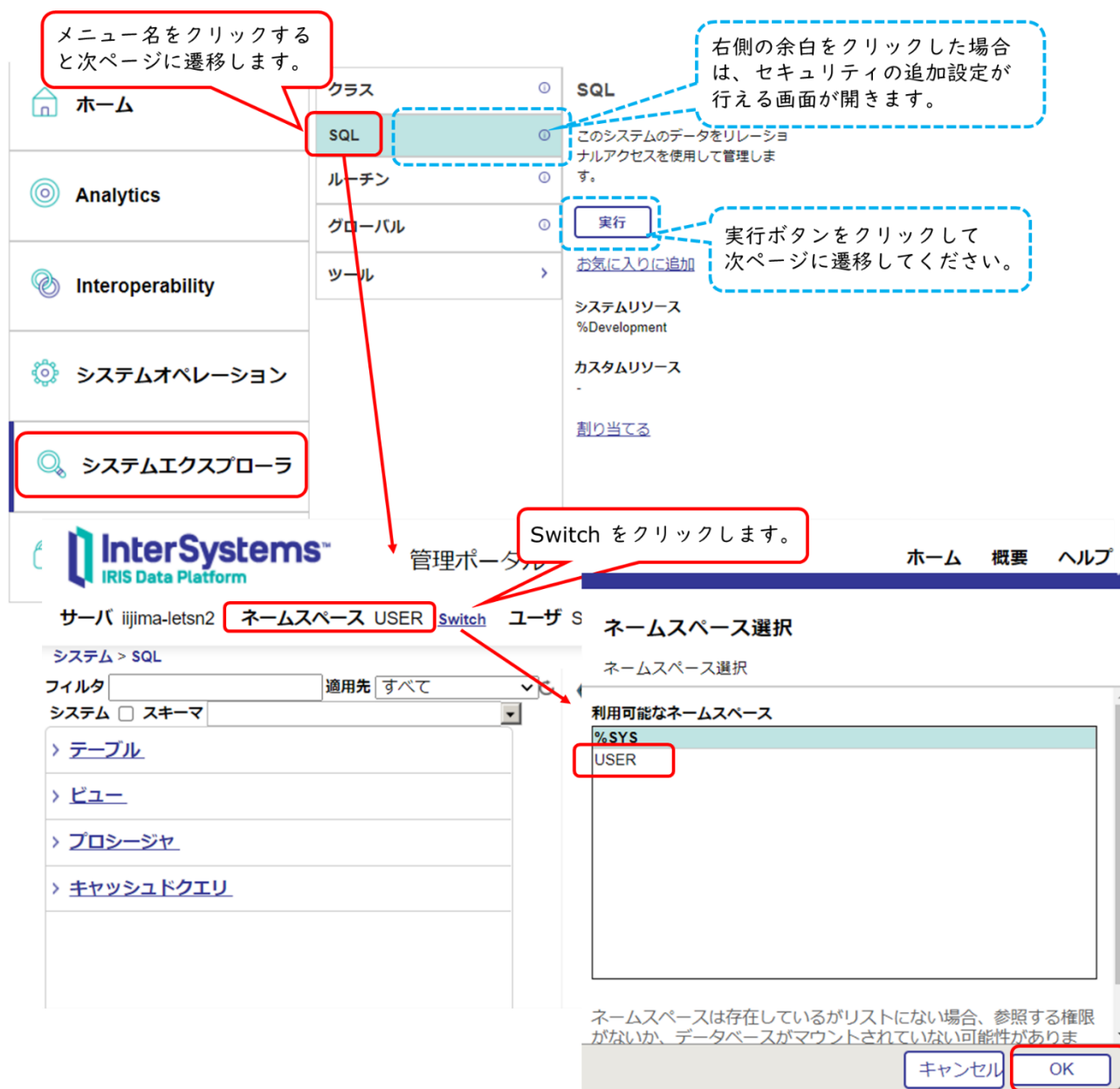


図 9 管理ポータル：SQL メニューを開く／ネームスペースの切り替え方

スキーマで FS を選択し、**テーブル** 赤丸の部分をクリックすると、Person テーブルができて
いることを確認できます。



図 10 管理ポータル SQL メニュー：FS.Person テーブルの確認

画面左で、FS.Person を選択し、画面右の **フィールド** をクリックすると、クラスとして定義した FS.Person の詳細情報が確認できます（データタイプ：%String の設定は、VARCHAR(50)の定義として登録されていることを確認できます）。

では、さっそくデータを確認してみましょう。

一番簡単な方法は、FS.Person テーブル名を選択した状態で画面右側の「テーブルを開く」をクリックする方法です。その他、「クエリの実行」タブで確認することもできます。

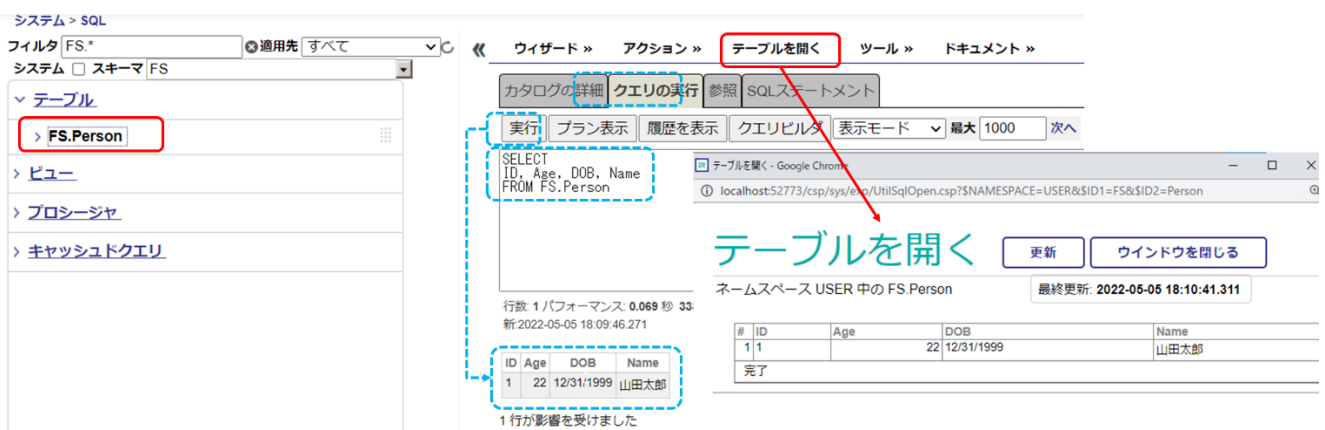


図 11 管理ポータル：SQL メニュー データの確認

この後の演習で検索を行うため、INSERT 文を利用して、1 件データを追加ください。

```
INSERT INTO FS.Person (Name,DOB) VALUES('鈴木花子',TODATE('2003-02-03','YYYY-MM-DD'))
```

(5) SQL でアクセス

管理ポータルを利用して SQL でデータを確認することができました。次は、Python から SQL を実行してみます。

データ登録時に使用したシェルに戻り、Python 上で SQL を実行する方法を確認します。

SQL を直接実行する場合は、**iris** モジュールの **sql** クラスの **exec()** メソッドの引数に実行したいクエリを指定します。

```
rs=iris.sql.exec("SELECT * FROM FS.Person")
```

実行結果は IRIS の [%SYS.Python.ResultSet](#) のオブジェクトとして返されます。

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

イテレータを利用する場合は、**next()** メソッドを利用して行移動しながら次の要素（リスト）を取得できます⁶。

```
>>> print(next(rs))
['1', 22, 58073, '山田太郎']
>>> print(next(rs))
['2', 19, 59203, '鈴木花子']
>>>
```

メモ：DOB は %Date のデータタイプを使用しているため、内部数値がデータベースに登録されています。

next() を StopIteration 例外が出るまで実行して全件取得する例は以下の通りです。

```
>>> rs=iris.sql.exec("SELECT * FROM FS.Person")
>>> while True:
...     try:
...         print(next(rs))
...     except:
...         break
...
['1', 22, 58073, '山田太郎']
['2', 19, 59203, '鈴木花子']
```

⁶ リストの順序はカラム順です。確認方法は P17[図 10 管理ポータル SQL メニュー：FS.Person テーブルの確認]をご覧ください。

次に、**enumerate()**関数を利用して検索結果を取得する例を確認します。

今回は、DOB の値を内部数値から表示形式に変換するため、`iris.system.SQL.TOCHAR()`を利用して
います（第 1 引数に DOB の値、第 2 引数に表示形式のフォーマットを指定します）。

```
>>> 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
```

iris モジュールの **sql クラス** には、**exec()**メソッドの他に実行準備を行う **prepare()**メソッドもあります。**exec()**メソッドと同様に引数にクエリを指定します。

クエリ指定時、入力引数に置き換え文字として **?** を指定できます。**prepare()**メソッドの実行でエラーがない場合は、**execute()**メソッドを使用して SQL を実行します。引数の値は **prepare()**メソッドで指定した引数を順番通りに **execute()**メソッドに指定します。

以下の例は、レコードの内部番号である ID（＝永続オブジェクトの ID）を利用し、引数で指定した値と完全一致のレコードを取得しています。

```
>>> statement=iris.sql.prepare("SELECT * FROM FS.Person WHERE ID=?")
>>> rs=statement.execute(1)
>>> 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
>>>
>>> rs=statement.execute(2)
>>> 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:鈴木花子、誕生日：2003-02-03
>>>
```

iris モジュールの **sql クラス**には、SELECT の実行結果を dataframe に変換する **dataframe()** メソッドもあります。

Windows 環境では、dataframe を使用するために、pandas モジュールのインストールを事前に行う必要があります。詳細は、脚注⁷をご参照ください。

Python シェルでの実行は以下の通りです。

```
>>> import pandas
>>> df=iris.sql.exec("SELECT * FROM FS.Person").dataframe()
>>> df
   id  age      dob  name
0   1   22 1999-12-31  山田太郎
1   2   19 2003-02-03  鈴木花子
>>>
>>> statement=iris.sql.prepare("SELECT * FROM FS.Person WHERE ID<?")
>>> df=statement.execute(3).dataframe()
>>> df
   id  age      dob  name
0   1   22 1999-12-31  山田太郎
1   2   19 2003-02-03  鈴木花子
>>>
```

iris モジュールの **sql クラス**が提供するメソッドについては [ドキュメント](#) もご参照ください。

⁷ IRIS のインストールディレクトリ以下 bin に移動し、irisip を利用して pandas モジュールをインストールします。インストール時、インストールターゲットを --target で指定します。ターゲットは <IRIS インストールディレクトリ>%mgr%python を指定します。

例)

```
c:\InterSystems\IRIS\bin>irisip install --target c:\intersystems\iris\mgr\python pandas
```

(6) インスタンスメソッドの作成と実行

FS.Person のプロパティを表示するインスタンスメソッドを記述します。

インスタンスメソッドの定義は **Method** から始まります。

メソッド内で Python を記述する場合は、メソッドの属性 **[Language=python]** を指定します。

Method を記述すると、デフォルトで ObjectScript のひな形が表示されますが、引数、戻り値がないインスタンスメソッド Print() を作成したいので、戻り値、引数の指定を削除してください。

カレントインスタンスを示すには、self を利用します。

```
Method Print() [ Language = python ]
{
    print(f"名前は {self.Name} - 年齢は{self.Age} です")
}
```

Ctrl+S でファイルを保存し、コンパイルを実施します。

Python のシェルに戻り、FS.Person を 1 件オープンし、Print()メソッドを実行します。

オープンには、**__OpenId()メソッド**を使用します。引数には ID を指定します。

```
>>> p=iris.cls("FS.Person").__OpenId(1)
>>> p.Print()
名前は 山田太郎 - 年齢は 22 です
>>>
```

(7) クラスメソッドの作成と実行

メールアドレスを作成するクラスメソッドを定義します。

引数にアカウント名となる英数字の文字列を指定し、戻り値に適当なドメイン名を結合したメールアドレスを返すように記述します。

クラスメソッドの定義は、**ClassMethod** から始まります。(Class を付けないとインスタンスメソッドとして定義されます。)

インスタンスメソッドの定義を同様に、戻り値の後ろに **[Language=python]** を指定します。

引数、戻り値のデータタイプの指定には、Property 定義と同様に、As を利用します。(As の後ろにスペースを 1 つ入れると候補が表示されます)

引数、戻り値の文字列の指定は **As %String** で定義してください。

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

Python シェルでの実行例は以下の通りです。

```
>>> iris.cls("FS.Person").CreateEmail("abc")
'abc@mail.com'
>>>
```

ここで、作成したクラスメソッドに **SqlProc 属性** を追加します。

```
ClassMethod CreateEmail(uid As %String) As %String [ Language = python ,SqlProc]
```

この属性の追加により、FS.Person クラスの CreateEmail()メソッドがストアードプロシージャ **FS.Person_CreateEmail()** として投影されます。

Python シェルでの実行は以下の通りです。

```
>>> rs=iris.sql.exec("SELECT FS.Person_CreateEmail('test')")
>>> next(rs)
['test@mail.com']
>>>
```

ストアドプロシージャや関数の作成については、CREATE PROCEDURE、CREATE FUNCTION でも作成できます。

作成時、引数 **LANGUAGE PYTHON** を指定すると、Python でコードを記述できます。

CreateEmail()メソッドを CREATE FUNCTION で書き換えた例は、以下の通りです。

```
CREATE FUNCTION CreateEmail2(uid VARCHAR(50))
  RETURNS VARCHAR(50)
  FOR FS.ProcUtils
  LANGUAGE PYTHON
{
    if uid==None:
        return ""
    return uid+"@mail.com"
}
```

上記 CREATE 文を実行する、FS. ProcUtils クラスに、クラスメソッド CreateEmail2()を作成します。CREATE FUNCTION/PROCEDURE で定義した関数やプロシージャは、**パッケージ名.関数プロシージャ名()**で実行できるように、**SqlName 属性**も自動的に設定されます。

```
1 Class FS.ProcUtils Extends %Library.RegisteredObject [ DdlAllowed, Owner = {_SYSTEM}, Not ProcedureBlock ]
2 {
3     Debug this method
4     ClassMethod CreateEmail2(uid As %Library.String(MAXLEN=50)) As %Library.String(MAXLEN=50) [ Language = python, SqlName = CreateEmail2, SqlProc ]
5     {
6
7         if uid==None:
8             return ""
9         return uid+"@mail.com"
10    }
11 }
12
```

図 12 ご参考：CREATE FUNCTION で定義した関数

(8) エラー発生時の動作

Language=python のメソッド内で実行する SQL が不正な場合や、指定するクラスが存在していない場合は、Exception クラスを利用して例外を取得できます。

不正な SQL を実行した時の例外

```
>>> try:
...   rs=iris.sql.exec("SELECT * FROM FS.Person22")
... except Exception as e:
...   print(f"エラー : {str(e)}")
...
エラー : テーブル 'FS.PERSON22' が見つかりません
>>>
```

存在しないクラス名を指定した時の例外

```
>>> try:
...   p=iris.cls("FS.Person22")._New()
... except Exception as e:
...   print(f"エラー : {str(e)}")
...
エラー : iris.class: error finding class
>>>
```

インスタンス保存に使用する **_Save()** メソッドは、エラーが発生しても例外を生成しません。

理由は、メソッドの戻り値に **IRIS** のデータタイプ: **%Status** が指定されているため、エラー発生時エラーステータスを戻して正常終了するため、例外が発生しません。

%Status を戻すメソッドを実行する場合、ステータスエラーが戻っていないかどうか、確認する必要があります。

%Status を Python で確認するには、**iris.check_status()** を使用します。

以下の例では、プロパティ **DOB** に文字列を設定し、保存した場合のエラーの例です。

```
>>> p=iris.cls("FS.Person")._New()
>>> p.DOB="これは文字列"
>>> st=p._Save()
>>> iris.check_status(st)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
RuntimeError: エラー #7207: データタイプ値 'これは文字列' は妥当な数値ではありません
    > エラー #5802: プロパティ 'FS.Person:DOB' のデータタイプ妥当性検証が失敗しました。値は "これは文字列" です。
>>>
```

iris.check_status() の結果、**RuntimeError** が発生することを確認できたので、以下のように **try: except:** を利用したエラー処理を記述することもできます。

```
>>> try:
...   p=iris.cls("FS.Person")._New()
...   p.DOB="これは文字列"
...   st=p._Save()
...   iris.check_status(st)
... except RuntimeError as ex:
...   print(str(ex))
...
エラー #7207: データタイプ値 'これは文字列です' は妥当な数値ではありません
    > エラー #5802: プロパティ 'FS.Person:DOB' のデータタイプ妥当性検証が失敗しました。値は "これは文字列です" です。
```