Article
Guillaume Rongier · Juil 4, 2022 11m de lecture

Open Exchange

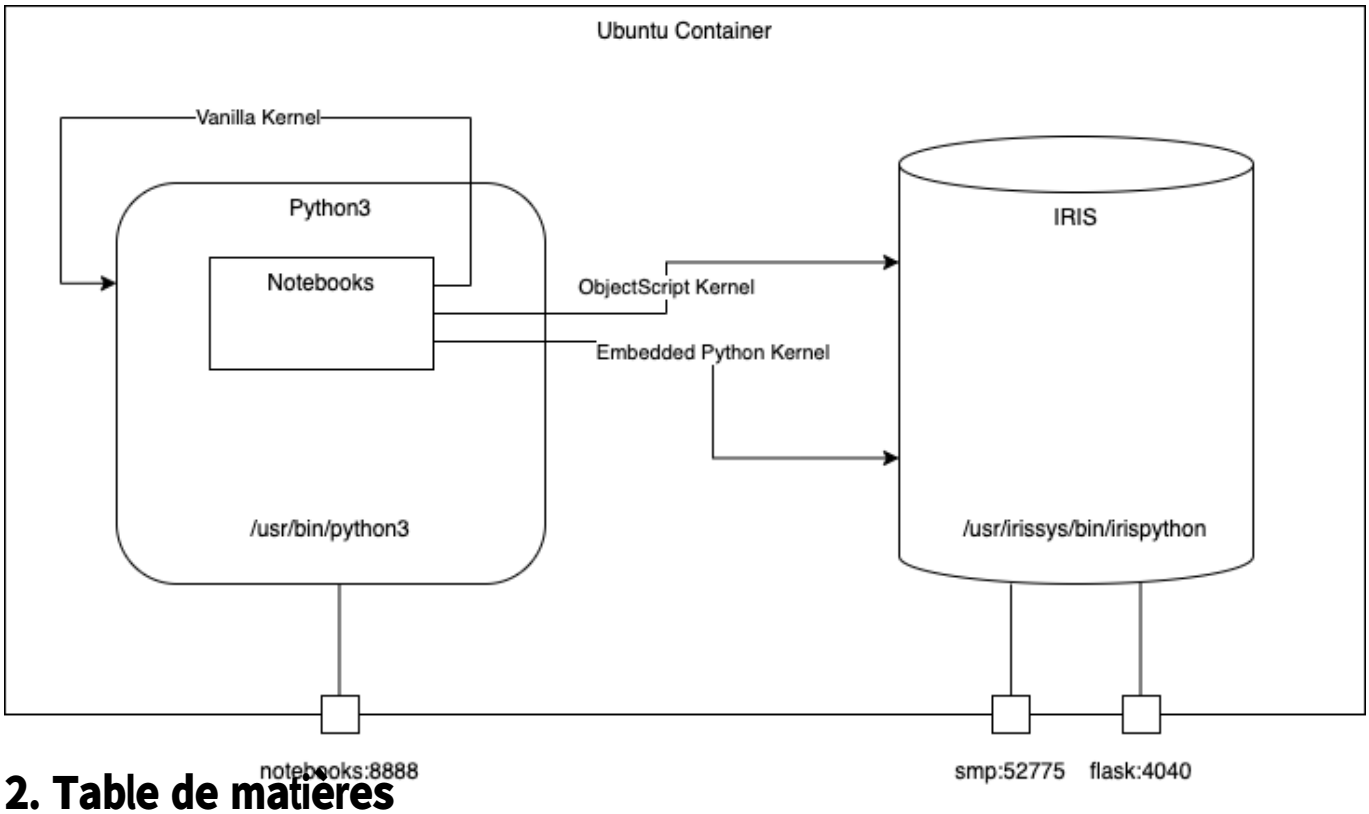
InterSystems IRIS 2021.2+ Exemples Python (Embedded, Native API et Notebooks)

Iris-python-template

Projet modèle avec divers codes Python à utiliser pour InterSystems IRIS Community Edition avec conteneur.

Caractéristiques :

- Notebooks
- Noyau Python intégré
  - Noyau ObjectScript
  - Noyau Vanilla Python
- Python intégré
- Code exemplaire
  - Démonstration de Flask
- API natives Python d'IRIS
- Code exemplaire



2. Table de matières

3. Installation

3.1. Docker

Le dépôt est dockerisé, vous pouvez donc cloner/git puller le dépôt dans n'importe quel local

```
git clone https://github.com/grongierisc/iris-python-template.git
```

Ouvrez le terminal dans ce dossier et exécutez :

```
docker-compose up -d
```

et ouvrez ensuite http://localhost:8888/tree pour Notebooks

Ou bien, ouvrez le dossier cloné dans VSCode, démarrez docker-compose et ouvrez l'URL via le menu VSCode :

4. Comment commencer le codage

4.1. Conditions préalables

Vérifiez que vous avez git and Docker desktop installé.

TCe dépôt est prêt à être codé dans VSCode avec le plugin ObjectScript.
Installer le plugin VSCode, Docker et ObjectScript et ouvrir le dossier dans VSCode.

4.1.1. Commencer le codage en ObjectScript

Ouvrez la classe /src/ObjectScript/Embedded/Python.cls et essayez d'y apporter des modifications - elles seront compilées dans le conteneur docker IRIS en cours d'exécution.

4.1.2. Commencer le codage avec Python intégré

Le moyen le plus simple est d'exécuter VsCode dans le conteneur.

Pour vous attacher à un conteneur Docker, sélectionnez **Remote-Containers : Attach to Running Container...** dans la palette de commande (kbstyle(F1)) ou utiliser le **Remote Explorer** dans la barre d'activité et dans la vue **Containers**, sélectionner l'action en ligne **Attach to Container** sur le conteneur auquel vous voulez vous connecter.

Ensuite, configurez votre interpréteur python pour /usr/irissys/bin/irispython

4.1.3. Commencez le codage avec les Notebooks

Ouvrez cette url : http://localhost:8888/tree

Vous avez alors accès à trois notebooks différents avec trois noyaux différents.

- Noyau Python E;Embedded
- Noyau ObjectScript
- Noyau Vanilla python3

5. Le contenu du dépôt

5.1. Dockerfile

Un dockerfile qui installe quelques dépendances de python (pip, venv) et sudo dans le conteneur pour faciliter le travail.
Puis il crée le dossiers dev et y copie ce dépôt git.

Il lance IRIS et importe les fichiers csv de Titanics, puis il active **%ServiceCallIn** pour **Python Shell**.
Utilisez le fichier docker-compose.yml correspondant pour configurer facilement des paramètres supplémentaires tels que le numéro de port et l'emplacement des clés et des dossiers d'hôte.

Ce dockerfile se termine par l'installation des exigences pour les modules python.

La dernière partie concerne l'installation de Jupyter Notebook et de ses noyaux.

Utilisez le fichier .env/ pour ajuster le dockerfile utilisé dans docker-compose.

5.2. .vscode/settings.json

Fichier de configuration pour vous permettre de coder immédiatement en VSCode avec le plugin VSCode ObjectScript

5.3. .vscode/launch.json

Fichier de configuration si vous voulez déboguer avec VSCode ObjectScript.

Découvrez tous les fichiers dans cet article

5.4. .vscode/extensions.json

Fichier de recommandation pour ajouter des extensions si vous voulez fonctionner avec VSCode dans le conteneur.

Plus d'informations ici

C'est très utile pour travailler avec du python intégré.

5.5. src folder

Ce dossier est divisé en deux parties, une pour les exemples ObjectScript et une pour le code Python..

5.5.1. src/ObjectScript

Un fragment de code différent qui montre comment utiliser python en IRIS.

5.5.1.1. src/ObjectScript/Embedded/Python.cls

Tous les commentaires sont en français pour vous permettre d'améliorer vos compétences en français également.

```
/// Exemple de python intégré
Class ObjectScript.Embbeded.Python Extends %SwizzleObject
{

/// HelloWorld avec un paramètre
ClassMethod HelloWorld(name As %String = "toto") As %Boolean [ Language = python ]
{
    print("Hello",name)
    return True
}

/// Description
Method compare(modèle, chaine) As %Status [ Language = python ]
{
    import re

    # compare la chaîne [chaîne] au modèle [modèle]
    # affichage résultats
    print(f"\nRésultats({chaine},{modèle})")
    match = re.match(modèle, chaine)
    if match:
        print(match.groups())
    else:
        print(f"La chaîne [{chaine}] ne correspond pas au modèle [{modèle}]")
}

/// Description
Method compareObjectScript(modèle, chaine) As %Status
{
    w !,"Résultats("_chaine_","_modèle_")",!
    set matcher=##class(%Regex.Matcher).%New(modèle)
    set matcher.Text=chaine
    if matcher.Locate() {
```

```
        write matcher.GroupGet(1)
    }
    else {
        w "La chaîne ["_chaine_"] ne correspond pas au modèle ["_modèle_"]"
    }
}

/// Description
Method DemoPyhtonToPython() As %Status [ Language = python ]
{
    # expression régulières en python
    # récupérer les différents champs d'une chaîne
    # le modèle : une suite de chiffres entourée de caractères quelconques
    # on ne veut récupérer que la suite de chiffres
    modèle = r"^.*?(\d+).*?$"

    # on confronte la chaîne au modèle
    self.compare(modèle, "xyz1234abcd")
    self.compare(modèle, "12 34")
    self.compare(modèle, "abcd")
}

Method DemoPyhtonToObjectScript() As %Status [ Language = python ]
{
    # expression régulières en python
    # récupérer les différents champs d'une chaîne
    # le modèle : une suite de chiffres entourée de caractères quelconques
    # on ne veut récupérer que la suite de chiffres
    modèle = r"^.*?(\d+).*?$"

    # on confronte la chaîne au modèle
    self.compareObjectScript(modèle, "xyz1234abcd")
    self.compareObjectScript(modèle, "12 34")
    self.compareObjectScript(modèle, "abcd")
}

/// Description
Method DemoObjectScriptToPython() As %Status
{
    // le modèle - une date au format jj/mm/aa
    set modèle = "^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$"
    do ..compare(modèle, "10/05/97")
    do ..compare(modèle, " 04/04/01 ")
    do ..compare(modèle, "5/1/01")
}

}
```

- HelloWorld
  - Une fonction simple pour dire "Bonjour!" en python
  - Il utilise l'enveloppe OjectScript avec la balise [Language = python]
- comparer
  - Une fonction python qui compare une chaîne de caractères avec un regx, s'il y a une correspondance, elle l'affiche, sinon elle affiche qu'aucune correspondance n'a été trouvée
- compareObjectScript
  - Même fonction que celle en python mais en ObjectScript
- DemoPyhtonToPython
  - Montrer comment utiliser une fonction python avec du code python enveloppé dans de

l'ObjectScript.

```
set demo = ##class(ObjectScript.Embbeded.Python).%New()

zw demo.DemoPyhtonToPython()
```

- DemoPyhtonToObjectScript
  - Une fonction python qui montre comment lancer une fonction ObjecScript
- DemoObjectScriptToPython
  - Une fonction ObjectScript qui montre comment appeler une fonction python

5.5.1.2. src/ObjectScript/Gateway/Python.cls

Une classe ObjectScript qui montre comment appeler un code phyton externe avec la fonctionnalité de la passerelle.

Dans cet exemple, le code python **n'est pas exécuté** dans le même processus d'IRIS.

```
/// Description
Classe Gateway.Python
{

/// Démonstration d'une passerelle python pour exécuter du code python en dehors d'un
 processus iris.
ClassMethod Demo() As %Status
{
    Set sc = $$$OK

    set pyGate = $system.external.getPythonGateway()

    d pyGate.addToPath("/irisdev/app/src/Python/gateway/Address.py")

    set objectBase = ##class(%Net.Remote.Object).%New(pyGate,"Address")

    set street = objectBase.street

    zw street

    Return sc
}

}
```

5.5.2. src/Python

Un autre fragment de code python qui montre comment utiliser le python intégré dans IRIS.

5.5.2.1. src/Python/embedded/demo.cls

Tous les commentaires sont en français pour vous permettre d'améliorer vos compétences en français également.

```
import iris

person = iris.cls('Titanic.Table.Passenger')._OpenId(1)
```

```
print(person.__dict__)
```

Importez d'abord le module iris qui permet d'activer les capacités de python intégré.
Ouvrez une classe persistante avec la fonction cls du module iris.
Notez que toutes les fonctions % sont remplacées par _

Pour exécuter cet exemple, vous devez utiliser le shell iris python :

```
/usr/irissys/bin/irispython /opt/irisapp/src/Python/embedded/demo.py
```

5.5.2.2. src/Python/native/demo.cls

Montrer comment utiliser l'api native dans le code python.

```
import irisnative

# créer une connexion à la base de données et une instance IRIS
connection = irisnative.createConnection("localhost", 1972, "USER", "superuser", "SYS
", sharedmemory = False)
myIris = irisnative.createIris(connection)

# classMethod
passenger = myIris.classMethodObject("Titanic.Table.Passenger","%OpenId",1)
print(passenger.get("name"))

# globale
myIris.set("hello","myGlobal")
print(myIris.get("myGlobal"))
```

Pour importer iris native, vous devez installer les Api Wheels native dans votre environnement Python.

```
pip3 install /usr/irissys/dev/python/intersystems_irispython-3.2.0-py3-none-any.whl
```

Ensuite, vous pouvez exécuter ce code python

```
/usr/bin/python3 /opt/irisapp/src/Python/native/demo.py
```

Notez que dans ce cas, une connexion est établie avec la base de données IRIS, ce qui signifie que **ce code est exécuté dans un processus différent de celui d'IRIS**.

5.5.2.3. src/Python/flask

Une démo complète de la combinaison entre python intégré et le micro framework flask.
Vous pouvez tester ce point final :

```
GET http://localhost:4040/api/passengers?currPage=1&pageSize=1
```

5.5.2.3.1. Comment cela fonctionne

Afin d'utiliser Python embarqué, nous utilisons irispython comme interpréteur python, et faisons :

```
import iris
```

Juste au début du fichier.

Nous serons alors en mesure d'appliquer des méthodes telles que :

Comme vous pouvez le voir, pour obtenir un passager avec un ID, il suffit d'exécuter une requête et d'utiliser son ensemble de résultats.

Nous pouvons également utiliser directement les objets IRIS :

Ici, nous utilisons une requête SQL pour obtenir tous les IDs dans le tableau, et nous récupérons ensuite chaque passager du tableau avec la méthode %OpenId() de la classe Titanic.Table.Passenger (notez que puisque % est un caractère illégal en Python, nous utilisons _au lieu de cela).

Grâce à Flask, nous implémentons toutes nos itinéraires et méthodes de cette façon.

5.5.2.3.2. Lancement du serveur flask

Pour lancer le serveur, nous utilisons gunicorn avec irispython.

Dans le fichier docker-compose, nous ajoutons la ligne suivante :

```
iris:
  command: -a "sh /opt/irisapp/server_start.sh"
```

Cela lancera, après le démarrage du conteneur (grâce au flag -a), le script suivant :

```
#!/bin/bash

cd ${SRC_PATH}/src/Python/flask

${PYTHON_PATH} -m gunicorn --bind "0.0.0.0:8080" wsgi:app &

exit 1
```

Avec les variables d'environnement définies dans le Dockerfile comme suit :

```
ENV PYTHON_PATH=/usr/irissys/bin/irispython
ENV SRC_PATH=/opt/irisapp/
```

5.5.3. src/Notebooks

Trois notebooks avec trois noyaux différents :

- Un noyau Python3 pour exécuter les API natives.
- Un noyau Python intégré
- Un noyau ObjectScript

Les notebooks sont accessibles ici http://localhost:8888/tree

5.5.3.1. src/Notebooks/HelloWorldEmbedded.ipynb

Ce notebook utilise le noyau python intégré d'IRIS.

Il montre des exemples pour ouvrir et sauvegarder des classes persistantes et comment exécuter des requêtes SQL.

5.5.3.2. src/Notebooks/IrisNative.ipynb

Ce notebook utilise le noyau python vanilla.

Il montre un exemple d'exécution d'iris native apis.

5.5.3.3. src/Notebooks/ObjectScript.ipynb

Ce notebook utilise le noyau ObjectScript.

Il montre un exemple d'exécution de code ObjectSCript et comment utiliser python intégré dans ObjectScript.

#Python #InterSystems IRIS
Voir l'application sur InterSystems Open Exchange

URL de la source:https://fr.community.intersystems.com/post/intersystems-iris-20212-exemples-python-embedded-native-api-et-notebooks