Artículo
Alberto Fuentes · 27 oct, 2022 Lectura de 8 min

Open Exchange

Interoperabilidad de InterSystems IRIS con Python Embebido

1. Interoperability-embedded-python

Esta prueba de concepto pretende mostrar cómo el ***framework* de interoperabilidad de IRIS** puede utilizarse con **Python Embebido**.

1.1. Índice

1.2. Ejemplo

```
import grongier.pex
import iris
import MyResponse

class MyBusinessOperation(grongier.pex.BusinessOperation):

    def OnInit(self):
        print("[Python] ...MyBusinessOperation:OnInit() is called")
        self.LOGINFO("Operation OnInit")
        return
```

```
    def OnTeardown(self):
        print("[Python] ...MyBusinessOperation:OnTeardown() is called")
        return

    def OnMessage(self, messageInput):
        if hasattr(messageInput,"_IsA"):
            if messageInput._IsA("Ens.StringRequest"):
                self.LOGINFO(f"[Python] ...This iris class is a Ens.StringRequest wit
h this message {messageInput.StringValue}")
        self.LOGINFO("Operation OnMessage")
        response = MyResponse.MyResponse("...MyBusinessOperation:OnMessage() echos")
        return response
```

1.3. Registrar un componente

No es necesario código en ObjectScript.
Gracias al método Grongier.PEX.Utils.RegisterComponent():

Inicia un Embedded Python Shell:

```
/usr/irissys/bin/irispython
```

A continuación, utiliza este método de clase para añadir un nuevo archivo py a la lista de componentes de interoperabilidad.

```
iris.cls("Grongier.PEX.Utils").RegisterComponent(<ModuleName>,<ClassName>,<PathToPyFi
le>,<OverWrite>,<NameOfTheComponent>)
```

por ejemplo:

```
iris.cls("Grongier.PEX.Utils").RegisterComponent("MyCombinedBusinessOperation","MyCom
binedBusinessOperation","/irisdev/app/src/python/demo/",1,"PEX.MyCombinedBusinessOper
ation")
```

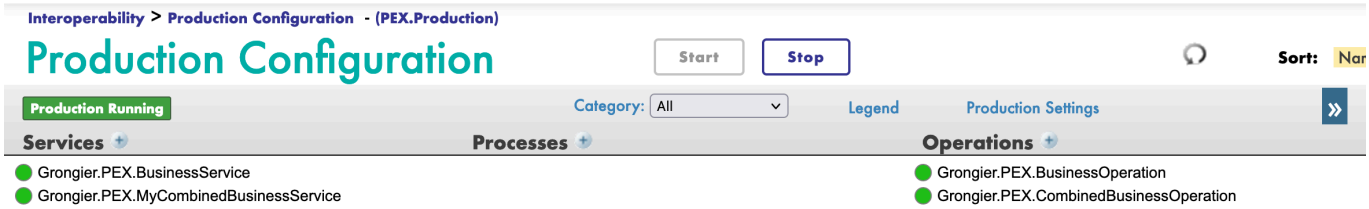
Esto es un truco, no es para producción.

2. Demo

La producción tiene cuatro componentes en Pure Python:

- Dos Business Services:
  - Grongier.PEX.MyCombinedBusinessService, que envía continuamente mensajes de sincronización a una Business Operation
    - Esos mensajes son objetos de Python en formato JSON y almacenados en Grongier.PEX.Message.
    - Código de Python: src/python/demo/MyCombinedBusinessService.py
  - Grongier.PEX.MyBusinessService, que básicamente no hace nada, es un Business Service en bruto que escribe registros de mensajes

        - Código de Python: src/python/demo/MyBusinessService.py
- Dos Business Operations:
    - Grongier.PEX.BusinessOperation, que recibe el mensaje de Grongier.PEX.MyCombinedBusinessService
        - Código de Python: src/python/demo/MyBusinessOperation.py
    - Grongier.PEX.CombinedBusinessOperation, puede recibir el mensaje Ens.StringRequest y la respuesta con Ens.StringResponse
        - Código de Python: src/python/demo/MyCombinedBusinessOperation.py



Nueva traza json para los mensajes nativos de Python:

3. Requisitos previos

Asegúrate de tener instalado git y el escritorio Docker.

4. Instalación con Docker

Clona el repositorio a cualquier directorio local

```
git clone https://github.com/grongierisc/interpeorability-embedded-python
```

Abre el terminal en este directorio y ejecuta:

```
docker-compose build
```

Ejecuta el contenedor IRIS con tu proyecto:

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

5. Instalación sin Docker

Instala el *grongierpex-1.0.0-py3-none-any.whl* en tu instancia local de iris:

```
/usr/irissys/bin/irispython -m pip install grongier_pex-1.0.0-py3-none-any.whl
```

A continuación, carga las clases de ObjectScript:

```
do $System.OBJ.LoadDir("/opt/irisapp/src","cubk","*.cls",1)
```

6. Cómo ejecutar el ejemplo

Abre la producción y comienza a utilizarla. Se comenzará a ejecutar el código del ejemplo.

7. Qué hay dentro del repositorio

7.1. Dockerfile

Un dockerfile que instala algunas dependencias de Python (pip, venv) y sudo en el contenedor por conveniencia. Después, crea el directorio dev y copia el repositorio git.

Inicia IRIS e importa los archivos csv de Titanic, después activa **%ServiceCallIn** para **Python Shell**. Utiliza el docker-compose.yml relacionado para configurar fácilmente parámetros adicionales como el número de puerto y dónde mapear contraseñas y carpetas del host.

Este dockerfile termina con la instalación de los requisitos para los módulos de python.

La última parte es sobre la instalación de Jupyter Notebook y sus kernels.

Utiliza el archivo .env/ para ajustar el dockerfile que se utiliza en docker-compose.

7.2. .vscode/settings.json

Archivo de configuración para poder codificar en VSCode con el plugin VSCode ObjectScript

7.3. .vscode/launch.json

Archivo de configuración si quieres depurar con VSCode ObjectScript

Lee sobre todos los archivos en este artículo

7.4. .vscode/extensions.json

Archivo de recomendación para añadir extensiones si quieres ejecutar VSCode en el contenedor.

Más información aquí

Esto es muy útil para trabajar con Python Embebido.

7.5. src folder

```
src
??? Grongier
?   ??? PEX // ObjectScript classes that wrap python code
?       ??? BusinessOperation.cls
?       ??? BusinessProcess.cls
?       ??? BusinessService.cls
?       ??? Common.cls
?       ??? Director.cls
?       ??? InboundAdapter.cls
?       ??? Message.cls
?       ??? OutboundAdapter.cls
```

```
?       ??? Python.cls
?       ??? Test.cls
?       ??? Utils.cls
??? PEX // Some example of wrapped classes
?   ??? MyBusinessOperationWithAdapter.cls
?   ??? MyBusinessOperationWithIrisAdapter.cls
?   ??? MyBusinessOperationWithPythonAdapter.cls
?   ??? MyBusinessService.cls
?   ??? MyOutboundAdapter.cls
?   ??? Production.cls
??? python
    ??? demo // Actual python code to run this demo
    ?   ??? MyBusinessOperation.py
    ?   ??? MyBusinessOperationWithAdapter.py
    ?   ??? MyBusinessOperationWithIrisAdapter.py
    ?   ??? MyBusinessProcess.py
    ?   ??? MyBusinessService.py
    ?   ??? MyCombinedBusinessOperation.py
    ?   ??? MyCombinedBusinessProcess.py
    ?   ??? MyCombinedBusinessService.py
    ?   ??? MyInboundAdapter.py
    ?   ??? MyLoggingOperation.py
    ?   ??? MyNonPollingStarter.py
    ?   ??? MyOutboundAdapter.py
    ?   ??? MyRequest.py
    ?   ??? MyResponse.py
    ?   ??? MySyncBusinessProcess.py
    ?   ??? SimpleObject.py
    ??? dist // Wheel used to implement python interoperability components
    ?   ??? grongier_pex-1.0.0-py3-none-any.whl
    ??? grongier
    ?   ??? pex // Helper classes to implement interoperability components
    ?       ??? _BusinessHost.py
    ?       ??? _BusinessOperation.py
    ?       ??? _BusinessProcess.py
    ?       ??? _BusinessService.py
    ?       ??? _Common.py
    ?       ??? _Director.py
    ?       ??? _InboundAdapter.py
    ?       ??? _Message.py
    ?       ??? _OutboundAdapter.py
    ?       ??? __init__.py
    ??? setup.py // setup to build the wheel
```

8. Cómo añadir un nuevo componente

8.1. InboundAdapter

Para implementar InboundAdapter en Python:

Subclase de grongier.pex.InboundAdapter en Python. Sobreescribe el método OnTask().

8.2. OutboundAdapter

Para implementar OutboundAdapter en Python:

Subclase de grongier.pex.OutboundAdapter en Python. Implementar los métodos de acción requeridos.

8.3. BusinessService

Para implementar BusinessService en Python:

Subclase de grongier.pex.BusinessService en Python. Sobreescribe el método OnProcessInput().

8.4. BusinessProcess

Para implementar BusinessProcess en Python:

Subclase de grongier.pex.BusinessProcess en Python. Sobreescribe los métodos OnRequest(), OnResponse() y OnComplete().

8.5. BusinessOperation

Para implementar BusinessOperation en Python:

Subclase de grongier.pex.BusinessOperation en Python. Sobreescribe el método OnMessage().

8.6. Registrar un componente

Iniciar un Embedded Python Shell:

```
/usr/irissys/bin/irispython
```

A continuación, utiliza este método de clase para añadir un nuevo archivo py a la lista de componentes de interoperabilidad.

```
iris.cls("Grongier.PEX.Utils").RegisterComponent(<ModuleName>,<ClassName>,<PathToPyFile>,<OverWrite>,<NameOfTheComponent>)
```

por ejemplo:

```
iris.cls("Grongier.PEX.Utils").RegisterComponent("MyCombinedBusinessOperation","MyCombinedBusinessOperation","/irisdev/app/src/python/demo/",1,"PEX.MyCombinedBusinessOperation")
```

8.7. Uso directo de Grongier.PEX

Si no quieres utilizar la utilidad RegisterComponent. Puedes añadir un componente Grongier.PEX.Business* y configurar las propiedades:

- %module :
    - Nombre del módulo correspondiente a tu código de python
- %classname :

    - Nombre de la clase de tu componente
- %classpaths
    - Ruta donde se encuentra tu componente.
    - Puede ser una o más Rutas de clase (separadas por el carácter '¦') necesarias además de PYTHONPATH

Por ejemplo:

9. Trabajo futuro

- Solo se ha probado Service y Operation.
- Trabajo en curso sobre el adaptador

10. Créditos

La mayor parte del código procede de PEX para Python, de Mo Cheng y Summer Gerry.

La parte del registro procede del formulario de características no liberado de IRIS 2021.3.

#Embedded Python #Mejores prácticas #Python #InterSystems IRIS
Ir a la aplicación en InterSystems Open Exchange

URL de fuente:https://es.community.intersystems.com/post/interoperabilidad-de-intersystems-iris-con-python-embebido