Article
Guillaume Rongier · Juil 8, 2022 3m de lecture

Appel de classmethods avec Native API pour Python

InterSystems Native SDK pour Python est une interface légère pour les API d'InterSystems IRIS qui n'étaient auparavant disponibles que via ObjectScript.

Je suis particulièrement intéressé par la possibilité d'appeler des méthodes ObjectScript, plus précisément des méthodes de classe. Cela fonctionne, et cela fonctionne très bien, mais par défaut, les appels ne supportent que les arguments scalaires : chaînes de caractères, booléens, nombres intégraux et flottants.

Mais si vous voulez :
- Transmettre ou retourner des structures, telles que des dicts ou des listes
- Transmettre ou retourner des flux

Vous aurez besoin d'écrire un code glue ou de prendre ce projet (s'installe avec pip install edpy). Le paquet edpy vous donne une signature simple :

```
call(iris, class_name, method_name, args)
```

qui vous permet d'appeler n'importe quelle méthode ObjectScript et d'obtenir des résultats en retour.

Importez-le comme ceci :

```
from edpy import iris
```

call accepte 4 arguments requis :
- iris - la référence à un objet IRIS
- classname - Classe IRIS à appeler
- methodname - Méthode IRIS à appeler
- args - liste de 0 ou plusieurs arguments

Arguments

Chaque argument peut être l'un des suivants :

- une chaîne de caractères (n'importe quelle longueur, si elle est supérieure à $$$MaxStringLength ou à 3641144 symboles, elle sera automatiquement convertie en un flux de données
- booléen
- nombre entier
- flottant
- dict (se transforme en un objet dynamique)
- liste ou tuplet (converti en tableau dynamique)

Les arguments de type dict, liste et tuplet peuvent contenir récursivement d'autres dicts, listes et tuplets (tant que la mémoire le permet).

Valeur de retour

En retour, nous attendons soit une table/ un objet dynamique, soit une chaîne/un flux JSON. Dans ce cas, edpy le convertit d'abord en chaîne Python et, si possible, l'interprète comme un dict ou une liste Python. Sinon, le résultat sera retourné à l'appelant tel quel.

C'est à peu près tout, mais laissez-moi vous donner quelques exemples de méthodes ObjectScript et comment les appeler en utilisant cette fonction Python.

Exemple 1: Pong

```
ClassMethod Test(arg1, arg2, arg3) As %DynamicArray
{
    renvoie [(arg1), (arg2), (arg3)]
}
```

Appelez avec:

```
>>> iris.call(iris_native, "User.Py", "Test", [1, 1.2, "ABC"])
[1, 1.2, 'ABC']
```

Pas de surprise ici. Les arguments sont regroupés dans une liste et renvoyés à l'appelant.

Exemple 2: Propriétés

```
ClassMethod Test2(arg As %DynamicObject) As %String
{
    renvoie arg.Prop
}
```

Appelez comme ça :

```
>>> iris.call(iris_native, "User.Py", "Test2", [{"Prop":123}])
123
```

Maintenant, pour une invocation plus intégrée :

```
>>> iris.call(iris_native, "User.Py", "Test2", [{"Prop":{"Prop2":123}}])
{'Prop2': 123}
```

Si une propriété est trop longue, ce n'est pas grave non plus - des flux seront utilisés pour l'envoyer à IRIS et/ou la renvoyer :

```
ret = iris.call(iris_native, "User.Py", "Test2", [{"Prop":"A" * 10000000}])
>>> len(ret)
10000000
```

Si vous avez besoin de flux garantis du côté d'InterSystems IRIS, vous pouvez utiliser %Get:

```
set stream = arg.%Get("Prop",,"stream")
```

Si le flux est codé en base64, vous pouvez le décoder automatiquement avec :

```
set stream = arg.%Get("Prop",,"stream<base64")
```

Exemple 3: Chaîne ou flux

```
ClassMethod Test3(arg As %Stream.Object) As %String
{
    set file = ##class(%Stream.FileCharacter).%New()
    set file.TranslateTable = "UTF8"
    set filename = ##class(%File).ManagerDirectory() _ "test.txt"
    do file.LinkToFile(filename)
    if $isObject(arg) {
        set sc = file.CopyFromAndSave(arg)
    } else {
        do file.Write(arg)
        set sc = file.%Save()
    }
    if $$$ISERR(sc) {
        set jsonret = {"status":0, "payload":($system.Status.GetErrorText(sc))}
    } else {
        set jsonret = {"status":1}
    }
    quit jsonret.%ToJSON()
}
```

Ici, nous écrivons soit une chaîne de caractères, soit un flux à <mgr>test.txt.

```
>>> iris.call(iris_native, "User.Py", "Test3", ["&#x1f60a;"])
{'status': 1}
```

Remarque: dans tous les échantillons de code, "&# x1f642 ;" est saisi comme .

Et si j'ouvre le fichier, je verrai un &#x1f60a ; et non deux ?? - donc nous préservons l'encodage.

```
>>> iris.call(iris_native, "User.Py", "Test3", ["&#x1f642;" * 10000000])
{'status': 1}
```

Je vais omettre la sortie du fichier pour des raisons de brièveté, mais c'est le cas.

Enfin, en passant un objet ou un tableau dynamique à l'intérieur, vous pouvez éviter complètement la dichotomie chaîne/flux, même si vous ne savez pas si la propriété sera plus courte ou plus longue que la limite de la chaîne. Dans ce cas, vous pouvez toujours obtenir votre propriété suspecte comme un flux.

Exemple 4: Les flux de retour

```
ClassMethod Test4(arg As %DynamicArray) As %String
{
    return arg.%Get(0)
}
```

Voilà à quoi ça ressemble :

```
>>> ret = iris.call(iris_native, "User.Py", "Test4", [["&#x1f60a;" * 10000000]])
>>> len(ret)
10000000
>>> ret[:5]
'&#x1f60a;&#x1f60a;&#x1f60a;&#x1f60a;&#x1f60a;'
```

Une dernière chose

Il existe également une fonction getiris(ip="localhost", port=1972, espace de nom="USER", nom d'utilisateur="SYSTEM", mot de passe="SYS") qui vous permet d'obtenir un objet IRIS fonctionnel..
Voici donc un exemple complet, si vous voulez l'essayer vous-même :

Téléchargez d'abord la classe User.Py et installez la bibliothèque python edpy :

```
pip install edpy
```

Et ensuite dans l'appel python :

```
from edpy import iris
iris_native = iris.get_iris()
iris.call(iris_native, "User.Py", "Test",  [1, 1.2, "ABC"])
iris.call(iris_native, "User.Py", "Test2", [{"Prop":123}])
iris.call(iris_native, "User.Py", "Test2", [{"Prop":{"Prop2":123}}])
ret2 = iris.call(iris_native, "User.Py", "Test2", [{"Prop":"A" * 10000000}])
iris.call(iris_native, "User.Py", "Test3", ["&#x1f60a;"])
iris.call(iris_native, "User.Py", "Test3", ["&#x1f60a;" * 10000000])
ret4 = iris.call(iris_native, "User.Py", "Test4", [["&#x1f60a;" * 10000000]])
```

Conclusion

Native SDK for Python est un outil puissant, vous fournissant un accès complet et sans restriction à InterSystems IRIS. J'espère que ce projet vous fera gagner du temps dans le traitement des appels InterSystems IRIS. Y a-t-il une combinaison d'arguments de méthode qu'il ne supporte pas ? Si oui, partagez dans les commentaires comment vous les appelez.

Liens

- Native SDK Docs
- User.Py class
- Repo

#Python #InterSystems IRIS

URL de la source:https://fr.community.intersystems.com/post/appel-de-classmethods-avec-native-api-pour-python