Artículo
Ricardo Paiva · 11 ago, 2021 Lectura de 3 min

Cómo aprovechar al máximo $Query

Me encontré con un interesante caso de uso de ObjectScript con una solución general que quería compartir.

Caso de uso:

Tengo una matriz JSON (específicamente, en mi caso, una matriz de problemas de *Jira*) que quiero agregar en algunos campos, por ejemplo: categoría, prioridad y tipo de problema. Después quiero combinar los agregados en una lista simple con el total de cada uno de los grupos. Por supuesto, para la agregación, tiene sentido utilizar una matriz local en el formulario:

```
agg(category, priority, type) = total
```

De tal manera que para cada registro en la matriz de entrada simplemente puedo hacer lo siguiente:

```
Do $increment(agg(category, priority, type))
```

Pero, cuando haya hecho la agregación, quiero conseguir un formulario más fácil sobre el que iterar, como una matriz con subíndices enteros:

```
summary = n
summary(1) = $listbuild(total1, category1, priority1, type1)
...
summary(n) = $listbuild(totalN, categoryN, priorityN, typeN)
```

Solución básica:

El enfoque sencillo es simplemente tener tres bucles "For" anidados con $Order, por ejemplo:

```
Set category = ""
For {
    Set category = $Order(agg(category))
    Quit:category=""

    Set priority = ""
    For {
        Set priority = $Order(agg(category,priority))
        Quit:priority=""

        Set type = ""
        For {
            Set type = $Order(agg(category,priority,type),1,total)
            Quit:type=""

            Set summary($i(summary)) = $listbuild(total,category,priority,type)
```

```
        }
    }


}
```

Esto es lo que empecé a hacer, pero es mucho código, y si tuviera más dimensiones que agregar se volvería difícil de manejar rápidamente. Esto me hizo preguntarme: ¿hay una solución general para conseguir lo mismo? ¡Resulta que sí la hay!

Mejor solución con $Query:

Decidí que usar $query podría ayudar. Ten en cuenta que esta solución asume una capacidad uniforme de los subíndices/valores en toda la matriz local, sucederían cosas extrañas si se vulnerara esta suposición.

```
ClassMethod Flatten(ByRef deep, Output flat) [ PublicList = deep ]
{
    Set reference = "deep"
    For {
        Set reference = $query(@reference)
        Quit:reference=""
        Set value = $listbuild(@reference)
        For i=1:1:$qlength(reference) {
            Set value = value_$listbuild($qsubscript(reference,i))
        }
        Set flat($i(flat)) = value
    }
}
```

Así que el fragmento de código anterior se sustituye por:

```
Do ..Flatten(.agg,.summary)
```

Hay que tener en cuenta algunas cosas sobre esta solución:

- *deep* necesita estar en la PublicList de $query para ser capaz de operar en ella
- en cada iteración, *reference* se cambia para hacer referencia al siguiente conjunto de subíndices que tenga un valor en *deep*, por ejemplo el valor podría ser: *deep("foo","bar")*
- $qlength devuelve el número de subíndices en *reference*
- $qsubscript devuelve el valor del enésimo subíndice de *reference*
- Cuando las listas $listbuild están concatenadas, el resultado es una lista $listbuild válida con las listas combinadas (¡esto es mucho mejor que usar cualquier otro separador!)

Resumen

$query, $qlength y $qsubscript son útiles para lidiar con matrices globales/locales de capacidad arbitraria.

Lecturas adicionales

$Query: https://docs.intersystems.com/irisforhealthlatest/csp/docbook/DocBook.UI...
$QSubscript: https://docs.intersystems.com/irisforhealthlatest/csp/docbook/Doc.View.c...
$QLength: https://docs.intersystems.com/irisforhealthlatest/csp/docbook/Doc.View.c...

#Code Snippet #Mejores prácticas #ObjectScript #Caché #InterSystems IRIS