Artículo
Alberto Fuentes · 27 ene, 2022 Lectura de 6 min

Cómo listar todas las propiedades en una clase (y por qué me encanta ObjectScript)

@Ming.Zhou hizo una buena pregunta: ¿Cómo conseguir todas las propiedades definidas en una clase desde %Dictionary.ClassDefinition? y la respuesta resume exactamente por qué ObjectScript es mi lenguaje favorito.

Cuando describo por primera vez ObjectScript o IRIS a alguien, siempre explico que puedes escribir una clase, compilarla, obtener una tabla y trabajar con tus datos desde una perspectiva orientada a objetos o relacional - la que resulte más natural. En cualquier caso, es sólo un fino envoltorio alrededor de los *Globals*, las estructuras de datos super rápidas existentes por debajo, y que también puedes usar cuando realmente necesitas ese acelerón extra.

Cuando hablo con alguien muy friki, entonces destaco que ObjectScript permite todos los tipos de metaprogramación sofisticada, porque puedes interactuar con *la clase que acabas de escribir* exactamente de la misma forma - desde una perspectiva de objeto o relacional, o usando las estructuras de datos super rápidas subyacentes, si necesitas ese acelerón extra.

Puedes verlo en la respuesta a la pregunta: "¿Cómo consigo todas las propiedades en una clase, incluyendo las propiedades heredadas?"

Aquí hay tres formas diferentes de obtener la misma respuesta:

```
Class DC.Demo.PropertyQuery Extends %Persistent
{

Property Foo As %String;

Property Bar As %Boolean;

/// Demonstrates all the ways to skin this particular cat
ClassMethod Run()
{
    for method = "FromRelationship","WithQuery","AsQuicklyAsPossible" {
        write !,method,":"
        do $classmethod($classname(),"GetProperties"_method)
        write !
    }
}

/// Get properties using the properties relationship in %Dictionary.CompiledClass
ClassMethod GetPropertiesFromRelationship()
{
    // Minor problem: %OpenId and Properties.GetNext() are slow because they load mor
e data than you strictly need.
    // More global references = it takes longer.
    set class = ##class(%Dictionary.CompiledClass).IDKEYOpen($classname(),,.sc)
    $$$ThrowOnError(sc)
    set key = ""
    for {
```

```
        set property = class.Properties.GetNext(.key)
        quit:key=""
        set properties(property.Name) = $listbuild(property.Type,property.Origin)
    }

    do ..Print(.properties)
}

/// Get properties using a query against %Dictionary.CompiledProperty
ClassMethod GetPropertiesWithQuery()
{
    // Getting properties with SQL avoids the overhead of unnecessary references
    set result = ##class(%SQL.Statement).%ExecDirect(,
        "select Name,Type,Origin from %Dictionary.CompiledProperty where parent = ?",
        $classname())
    if result.%SQLCODE < 0 {
        throw ##class(%Exception.SQL).CreateFromSQLCODE(result.%SQLCODE,result.%Messa
ge)
    }
    while result.%Next(.sc) {
        $$$ThrowOnError(sc)
        set properties(result.Name) = $listbuild(result.Type,result.Origin)
    }
    $$$ThrowOnError(sc)

    do ..Print(.properties)
}

/// Get properties using macros wrapping direct global references
ClassMethod GetPropertiesAsQuicklyAsPossible()
{
    // Getting properties via macro-
wrapped direct global references is harder to read,
    // but is the fastest way to do it.
    set key = ""
    set class = $classname()
    for {
        set key = $$$comMemberNext(class,$$$cCLASSproperty,key)
        quit:key=""
        set type = $$$comMemberKeyGet(class,$$$cCLASSproperty,key,$$$cPROPtype)
        set origin = $$$comMemberKeyGet(class,$$$cCLASSproperty,key,$$$cPROPorigin)
        set properties(key) = $listbuild(type,origin)
    }

    do ..Print(.properties)
}

ClassMethod Print(ByRef properties)
{
    set key = ""
    for {
        set key = $order(properties(key),1,data)
        quit:key=""
        set $listbuild(type,origin) = data
        write !,"property: ",key,"; type: ",type,"; origin: ",origin
    }
}

Storage Default
```

```
{
<Data name="PropertyQueryDefaultData">
<Value name="1">
<Value>%%CLASSNAME</Value>
</Value>
<Value name="2">
<Value>Foo</Value>
</Value>
<Value name="3">
<Value>Bar</Value>
</Value>
</Data>
<DataLocation>^DC.Demo.PropertyQueryD</DataLocation>
<DefaultData>PropertyQueryDefaultData</DefaultData>
<IdLocation>^DC.Demo.PropertyQueryD</IdLocation>
<IndexLocation>^DC.Demo.PropertyQueryI</IndexLocation>
<StreamLocation>^DC.Demo.PropertyQueryS</StreamLocation>
<Type>%Storage.Persistent</Type>
}

}
```

Y por supuesto, utilizando cualquiera de las aproximaciones, la respuesta es la misma:

```
d ##class(DC.Demo.PropertyQuery).Run()

FromRelationship:
property: %%OID; type: %Library.RawString; origin: %Library.RegisteredObject
property: %Concurrency; type: %Library.RawString; origin: %Library.Persistent
property: Bar; type: %Library.Boolean; origin: DC.Demo.PropertyQuery
property: Foo; type: %Library.String; origin: DC.Demo.PropertyQuery

WithQuery:
property: %%OID; type: %Library.RawString; origin: %Library.RegisteredObject
property: %Concurrency; type: %Library.RawString; origin: %Library.Persistent
property: Bar; type: %Library.Boolean; origin: DC.Demo.PropertyQuery
property: Foo; type: %Library.String; origin: DC.Demo.PropertyQuery

AsQuicklyAsPossible:
property: %%OID; type: %Library.RawString; origin: %Library.RegisteredObject
property: %Concurrency; type: %Library.RawString; origin: %Library.Persistent
property: Bar; type: %Library.Boolean; origin: DC.Demo.PropertyQuery
property: Foo; type: %Library.String; origin: DC.Demo.PropertyQuery
```

Analizándolo mínimamente, lo que vemos es completamente lo que esperábamos. El acceso a la clase y las propiedades a través de objetos es mucho más costoso porque las definiciones de clases y los metadatos de clases compiladas se almacenan en una serie de *globals* - y no todas cuentan con los datos almacenados en una lista $listbuild, sino en un árbol con un valor por cada nodo del *global*. Abrir un objeto implica cargar toda esa información, así que por supuesto "FromRelationship" es el método más lento. Por supuesto, esto no es representativo del rendimiento del acceso a objetos en IRIS en general - esto sucede aquí en particular porque es un mal escenario para la utilización de objetos.

Las aproximaciones de consulta y acceso directo a *globals* son similares en términos de acceso a *globals*, pero no en líneas de rutina. La aproximación simple con SQL dinámico tiene el sobrecoste de preparar la consulta, cosa que debe hacer en cada iteración. Para evitar algunos de estos sobrecostes, podríamos reutilizar un %SQL.Statement previamente preparado, o emplear SQL embebido con un cursor, o incluso intentar algún truco tipo:

```
/// Get properties using a query against %Dictionary.CompiledProperty
ClassMethod GetPropertiesWithEmbeddedQuery(Output properties)
{
    set classname = $classname()

    // Quick/easy, skip writing a cursor and just extract the data after running a qu
ery that returns one row.
    // The following approach outperforms cursors (left as an exercise), and I hate w
orking with cursors anyway.
    &SQL(SELECT %DLIST($ListBuild(Name,Type,Origin)) INTO :allProperties FROM %Dictio
nary.CompiledProperty WHERE parent = :classname)
    if (SQLCODE < 0) {
        throw ##class(%Exception.SQL).CreateFromSQLCODE(SQLCODE,%msg)
    }
    if (SQLCODE = 100) {
        quit
    }
    set pointer = 0
    while $listnext(allProperties,pointer,propertyInfo) {
        set properties($list(propertyInfo)) = $list(propertyInfo,2,3)
    }
}
```

Añadiendo esto, el resultado quedaría así:

```
WithEmbeddedQuery:
Elapsed time (1000x): .024862 seconds; 25000 global references; 95003 routine lines

AsQuicklyAsPossible:
Elapsed time (1000x): .016422 seconds; 25000 global references; 33003 routine lines
```

#Modelo de datos de objetos #ObjectScript #InterSystems IRIS

URL de fuente:https://es.community.intersystems.com/post/c%C3%B3mo-listar-todas-las-propiedades-en-una-clase-y-por-qu%C3%A9-me-encanta-objectscript