Article
Timothy Leavitt · Jan 21, 2022 7m read

Listing all of the properties in a class (and why I love ObjectScript)

@Ming Zhou asked a great question in https://community.intersystems.com/post/how-get-all-properties-defined-c... and the answer sums up exactly why ObjectScript is my favorite.

When I'm first describing ObjectScript or IRIS to someone I always explain that you can write a class, compile it, get a table, and work with your data from an object or relational perspective - whichever is most natural. Either way, it's just a thin(ish) wrapper around super fast under-the-hood data structures called Globals, and you can use those too when you really need that extra burst of speed.

When I'm talking to someone really nerdy, I then point out that ObjectScript allows all sorts of fancy metaprogramming, because you can interact with *the class you just wrote* in exactly the same way - from an object or relational perspective, or using the super fast under-the-hood data structures if you need that extra burst of speed.

You can see that in the answer to the question: "How do I get all the properties in a class, including inherited properties?"

Here are three different ways to get the same answer:

```
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,":"
        kill properties
        do $classmethod($classname(),"GetProperties"_method,.properties)
        do ..Print(.properties)
        write !
    }
}

ClassMethod Benchmark()
{
    for method = "FromRelationship","WithQuery","AsQuicklyAsPossible" {
        write !,method,":",!
        set start = $zhorolog
        set startGlobalRefs = $system.Process.GlobalReferences($job)
        set startLines = $system.Process.LinesExecuted($job)
        for i=1:1:1000 {
            kill properties
            do $classmethod($classname(),"GetProperties"_method,.properties)
        }
```

```
        set endLines = $system.Process.LinesExecuted($job)
        set endGlobalRefs = $system.Process.GlobalReferences($job)
        write "Elapsed time (1000x): ",($zhorolog-start)," seconds; ",(endGlobalRefs-
startGlobalRefs)," global references; ",(endLines-startLines)," routine lines",!
    }
}

/// Get properties using the properties relationship in %Dictionary.CompiledClass
ClassMethod GetPropertiesFromRelationship(Output properties)
{
    // 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)
        // Avoids consuming excess memory
        do class.Properties.%UnSwizzleAt(key)
    }
}

/// Get properties using a query against %Dictionary.CompiledProperty
ClassMethod GetPropertiesWithQuery(Output properties)
{
    // 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)
}

/// Get properties using macros wrapping direct global references
ClassMethod GetPropertiesAsQuicklyAsPossible(Output properties)
{
    // 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)
    }
}
```

```
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>
}

}
```

And of course, through any of these approaches, the answer is the same:

```
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
```

Comparing the performance:

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

FromRelationship:
Elapsed time (1000x):
.78834 seconds; 1056000 global references; 2472003 routine lines

WithQuery:
Elapsed time (1000x): .095235 seconds; 28001 global references; 537007 routine lines

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

Analyzing this a bit, what we see is entirely expected. Object access for the class and properties is way more expensive because class definitions and compiled class metadata are stored across a bunch of globals - not with all the data in a $listbuild list (for fewer global references), but in a tree with a single value in each global node. Opening an object means reading all of these, so of course our "FromRelationship" method is the slowest by far. Of course, this isn't representative of performance of object access in IRIS in general - this just happens to be a particularly bad case for using objects.

Our query and raw global-based approaches are similar in terms of global references, but not routine lines. The simple approach above with Dynamic SQL has the overhead of preparing the query, which we incur on each iteration. To avoid some of this overhead, we could reuse a prepared %SQL.Statement, use embedded SQL with a cursor (which I dislike for a few reasons), or do something tricky like:

```
/// 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)
    }
}
```

Adding this in to the benchmark, I see:

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

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

Which is pretty close!

#Object Data Model #ObjectScript #InterSystems IRIS