Artículo
Kurro Lopez · 17 jul, 2019 Lectura de 12 min

Clases de consulta en InterSystems Caché

Las clases de consulta en InterSystems Caché son una herramienta muy útil que separa las consultas SQL del código Object Script de Caché. Básicamente funciona de la siguiente manera: supongamos que quiere utilizar la misma consulta SQL con distintos argumentos en varios lugares diferentes. En este caso, puede evitar la duplicación del código si declara el contenido de la consulta como una clase de consulta y después llama a esta consulta por su nombre. Este método también es conveniente para las consultas personalizadas, donde el desarrollador define con cuál de las tareas obtendrá la siguiente fila. ¿Esto le parece interesante? Entones, ¡siga leyendo!

Clase básica de consulta

En resumen, las consultas de clases básicas le permiten representar consultas SQL SELECT. El optimizador y el compilador SQL las administran de la misma forma que las consultas SQL estándar, pero son más convenientes en el momento de ejecutarlas desde el contexto de Caché Object Script. Se declaran como elementos de consulta en las definiciones de clase (al igual que los métodos o propiedades), de la siguiente manera:

- Tipo: %SQLQuery
- Todos los argumentos de su consulta SQL deben estar en la lista de argumentos
- Tipo de consulta: SELECT
- Utilice los dos puntos para acceder a cada argumento (de forma similar al SQL estático)
- Defina el parámetro ROWSPEC, el cual contiene información sobre los nombres y el tipo de datos en los resultados de salida, al igual que el orden de los campos
- (Opcional) Defina el parámetro CONTAINID, el cual corresponde al orden numérico si el campo contiene un ID. Si no necesita la devolución del ID, no asigne ninguno de los valores a CONTAINID
- (Opcional) Defina el parámetro COMPILEMODE, el cual corresponde a un parámetro similar que se encuentra en el SQL estático y especifica cuando debe compilarse la expresión SQL. Cuando este parámetro se configura para IMMEDIATE (de forma predeterminada), la consulta se compilará simultáneamente con la clase. Cuando este parámetro se configura para DYNAMIC, la consulta se compilará antes de que se ejecute por primera vez (de forma similar al SQL dinámico)
- (Opcional) Defina el parámetro SELECTMODE, el cual especifica el formato que tendrán los resultados de las consultas
- Si desea llamar esta consulta como un procedimiento SQL, agregue la propiedad SqlProc.
- Si desea renombrar la consulta, configure la propiedad SqlName. El nombre predeterminado de una consulta en un contexto SQL es el siguiente: PackageName.ClassNameQueryName
- Caché Studio proporciona un asistente incorporado para crear consultas de clases

Defina la muestra para la clase Sample.Person mediante la consulta ByName, la cual devolverá los nombres de todos los usuarios que comiencen con una letra específica

```
Class Sample.Person Extends %Persistent
{
Property Name As %String;
Property DOB As %Date;
Property SSN As %String;
```

```
Query ByName(name As %String = "") As %SQLQuery
    (ROWSPEC="ID:%Integer,Name:%String,DOB:%Date,SSN:%String",
     CONTAINID = 1, SELECTMODE = "RUNTIME",
     COMPILEMODE = "IMMEDIATE") [ SqlName = SP_Sample_By_Name, SqlProc ]
{
SELECT ID, Name, DOB, SSN
FROM Sample.Person
WHERE (Name %STARTSWITH :name)
ORDER BY Name
}
}
```

Puede llamar esta consulta desde Caché Object Script de la siguiente manera:

```
Set statement=##class(%SQL.Statement).%New()
Set status=statement.%PrepareClassQuery("Sample.Person","ByName")
If $$$ISERR(status) {
    Do $system.OBJ.DisplayError(status)
}
Set resultset=statement.%Execute("A")
While resultset.%Next() {
    Write !, resultset.%Get("Name")
}
```

O bien, puede obtener un resultado utilizando el método queryNameFunc que se generó automáticamente:

```
Set resultset = ##class(Sample.Person).ByNameFunc("A")
While resultset.%Next() {
    Write !, resultset.%Get("Name")
}
```

Esta consulta también puede llamarse desde SQLcontext de las siguientes dos formas:

```
Call Sample.SP_Sample_By_Name('A')
Select * from Sample.SP_Sample_By_Name('A')
```

Puede encontrar esta clase en el namespace predeterminado de Caché para SAMPLES. Eso es todo lo que necesita saber sobre las consultas simples. Ahora, prosigamos con las personalizadas.

Clases de consulta personalizadas

Aunque las consultas de clases básicas funcionan bien en la mayoría de los casos, algunas veces es necesario ejecutar un control absoluto sobre el comportamiento que tendrán las consultas en las aplicaciones, por ejemplo:

- Un sofisticado criterio de selección. Dado que en las consultas personalizadas se implementa un método Object Script de Caché, el cual devuelve la siguiente fila por su cuenta, estos criterios pueden ser tan sofisticados como sea necesario.
- Si los datos únicamente son accesibles mediante una API, en un formato que no desea utilizar
- Si los datos se almacenaron en globales (sin clases)
- Si necesita aumentar los permisos, con el fin de acceder a los datos
- Si necesita llamar a una API externa, con el fin de acceder a los datos

- Si necesita obtener acceso al sistema de archivos, con el fin de acceder a los datos
- Si necesita realizar operaciones adicionales antes de ejecutar la consulta (por ejemplo, establecer una conexión, comprobar las autorizaciones, etc.)

Por lo tanto, ¿cómo se crean las consultas de clases personalizadas? En primer lugar, necesita definir los 4 métodos que implementará durante todo el flujo de trabajo para su consulta, desde la inicialización hasta la eliminación:

- queryName — proporciona información sobre una consulta (de forma similar a las consultas básicas de clases)
- queryName**Execute** — ejecuta una consulta
- queryName**Fetch** — obtiene la siguiente fila como resultado de una consulta
- queryName**Close** — elimina una consulta

Ahora, analizaremos estos métodos con mayor detalle.

El método queryName

El método queryName representa la información sobre una consulta.

- Tipo: %Query
- Deje el contenido en blanco
- Defina el parámetro ROWSPEC, el cual contiene la información sobre los nombres y el tipo de datos en los resultados de salida, al igual que el orden de los campos
- (Opcional) Defina el parámetro CONTAINID, el cual corresponde al orden numérico si el campo contiene un ID. Si no necesita la devolución del ID, no asigne ningún valor a CONTAINID

Por ejemplo, crearemos la consulta AllRecords (queryName = AllRecords, y el método se llama simplemente AllRecords) de la cual saldrán todas las instancias, una por una, de la nueva clase persistente Utils.CustomQuery. Primero, crearemos una nueva clase persistente Utils.CustomQuery:

```
Class Utils.CustomQuery Extends (%Persistent, %Populate){
Property Prop1 As %String;
Property Prop2 As %Integer;
}
```

Ahora, escribiremos la consulta AllRecords:

```
Query AllRecords() As %Query(CONTAINID = 1, ROWSPEC = "Id:%String,Prop1:%String,Prop2
:%Integer") [ SqlName = AllRecords, SqlProc ]
{
}
```

El método queryNameExecute

Con el método queryNameExecute se inicializa por completo una consulta. La estructura característica de este método es la siguiente:

```
ClassMethod queryNameExecute(ByRef qHandle As %Binary, args) As %Status
```

Donde:

- qHandle se utiliza para las comunicaciones con otros métodos durante la implementación de una consulta
- Este método debe configurar a qHandle dentro del estado, el cual después aprobará al método queryNameFetch
- qHandle puede configurarse para OREF, una variante o una variable multidimensional
- Los args son parámetros adicionales aprobados para la consulta. Puede agregar tantos args como necesite (o no utilizarlos en absoluto)
- El método debe regresar la consulta al estado de inicialización

Pero continuemos con nuestro ejemplo. Es libre de realizar iteraciones mediante varias formas de extensión (a continuación, describiré los métodos de trabajo básicos para las consultas personalizadas), pero en este ejemplo realizaré iteraciones mediante el global utilizando la función $Order. En este caso, qHandle almacenará el ID actual y, dado que no necesita ningún argumento adicional, no se necesita el argumento arg. El resultado se verá de la siguiente manera:

```
ClassMethod AllRecordsExecute(ByRef qHandle As %Binary) As %Status {
    Set qHandle = ""    Quit $$$OK
}
```

El método queryNameFetch

El método queryNameFetch devuelve un solo resultado en el formulario $List The signature of this method is as follows:

```
ClassMethod queryNameFetch(ByRef qHandle As %Binary, ByRef Row As %List, ByRef AtEnd
As %Integer = 0) As %Status [ PlaceAfter = queryNameExecute ]
```

where:

- qHandle se utiliza para las comunicaciones con otros métodos durante la implementación de una consulta
- Cuando se ejecuta la consulta, qHandle comienza asignando los valores especificados por queryNameExecute o por llamadas previas de queryNameFetch.
- Cada fila debe configurarse con un valor de %List o hacia una cadena vacía, si se procesaron todos los datos
- AtEnd debe configurarse en 1, una vez que llegó al final de los datos.
- La función de la palabra clave PlaceAfter es identificar la posición del método en el código int. El método "Fetch" debe colocarse después del método "Execute", pero esto solamente es importante para el SQL estático, por ejemplo cursors dentro de las consultas

En general, dentro de este método se realizan las siguientes operaciones:

1. Compruebe que haya llegado hasta el final de los datos
2. Si aún quedan algunos datos: Cree una nueva %List y asigne un valor a la variable Row
3. De lo contrario, configure AtEnd en 1
4. Prepare qHandle para el siguiente resultado de fetch
5. Devuelva el estado

Así es como se verá en nuestro ejemplo:

```
ClassMethod AllRecordsFetch(ByRef qHandle As %Binary, ByRef Row As %List, ByRef AtEnd
```

```
 As %Integer = 0) As %Status {
    #; Iterating through ^Utils.CustomQueryD
    #; Writing the next id to qHandle and writing the global's value with the new id
into val
    Set qHandle = $Order(^Utils.CustomQueryD(qHandle),1,val)
    #; Checking whether there is any more data left
       If qHandle = "" {
        Set AtEnd = 1
        Set Row = ""
        Quit $$$OK
    }
    #; If not, create %List
    #; val = $Lb("", Prop1, Prop2) see Storage definition
    #; Row =$lb(Id,Prop1, Prop2)  see ROWSPEC for the AllRecords request
    Set Row = $Lb(qHandle, $Lg(val,2), $Lg(val,3))
    Quit $$$OK
}
```

El método queryNameClose

The queryNameClose method terminates the query, once all the data is obtained. The signature of this method is as follows:

```
ClassMethod queryNameClose(ByRef qHandle As %Binary) As %Status [ PlaceAfter = queryN
ameFetch ]
```

Donde:

- Caché ejecuta este método después de la última llamada del método queryNameFetch
- En otras palabras, es un eliminador de consultas
- Por lo tanto, debe disponer de todos los cursores SQL, consultas y variables locales en su implementación
- Los métodos devuelven los estados actuales

En nuestro ejemplo, debemos eliminar la variable local qHandle:

```
ClassMethod AllRecordsClose(ByRef qHandle As %Binary) As %Status {
    Kill qHandle
    Quit $$$OK
  }
```

¡Y aquí vamos! Una vez que compile la clase, podrá utilizar la consulta AllRecords desde %SQL.Statement, de manera similar al procedimiento de las consultas de clases básicas.

Métodos lógicos de iteración para consultas personalizadas

¿Cuáles métodos pueden utilizarse para realizar consultas personalizadas? En general, existen 3 métodos básicos:

- Iteración mediante un global
- SQL estático

- SQL dinámico

Iteración mediante un global
Este método se basa en utilizar $Order y funciones similares para realizar iteraciones mediante un global. Este método puede utilizarse en los siguientes casos:

- Cuando los datos se almacenaron en globales (sin clases)
- Cuando desea reducir el número de glorefs en el código
- Los resultados deben/pueden ordenarse por el subíndice del global

SQL estático
Este método se basa en cursores y en el SQL estático. Y se utiliza para:

- Hacer que el código int sea más comprensible
- Hacer que el trabajo con cursores sea más sencillo
- Acelerar el proceso de compilación (el SQL estático incluye las consultas de clases y, por lo tanto, se compila solo una vez).

Nota:

- Los cursores generados por consultas del tipo %SQLQuery se nombran automáticamente, por ejemplo, Q14.
- Todos los cursores utilizados dentro de una clase deben tener nombres diferentes.
- Los mensajes de error se relacionan con los nombres internos de los cursores, los cuales tienen caracteres adicionales que se encuentran al final de sus nombres. Por ejemplo, un error en el cursor Q140 en realidad lo causa el cursor Q14.
- Utilice PlaceAfter y asegúrese de que los cursores se utilizaron en la misma rutina donde se declararon.
- INTO debe utilizarse junto con FETCH, pero no con DECLARE.

Ejemplo de un SQL estático para Utils.CustomQuery:

```
Query AllStatic() As %Query(CONTAINID = 1, ROWSPEC = "Id:%String,Prop1:%String,Prop2:
%Integer") [ SqlName = AllStatic, SqlProc ]
{
}

ClassMethod AllStaticExecute(ByRef qHandle As %Binary) As %Status
{
    &sql(DECLARE C CURSOR FOR
        SELECT Id, Prop1, Prop2
        FROM Utils.CustomQuery
     )
     &sql(OPEN C)
    Quit $$$OK
}

ClassMethod AllStaticFetch(ByRef qHandle As %Binary, ByRef Row As %List, ByRef AtEnd
As %Integer = 0) As %Status [ PlaceAfter = AllStaticExecute ]
{
    #; INTO must be with FETCH
    &sql(FETCH C INTO :Id, :Prop1, :Prop2)
    #; Check if we reached end of data
    If (SQLCODE'=0) {
```

```
        Set AtEnd = 1
        Set Row = ""
        Quit $$$OK
    }
    Set Row = $Lb(Id, Prop1, Prop2)
    Quit $$$OK
}

ClassMethod AllStaticClose(ByRef qHandle As %Binary) As %Status [ PlaceAfter = AllSta
ticFetch ]
{
    &sql(CLOSE C)
    Quit $$$OK
}
```

SQL dinámico

Este método se basa en otras consultas de clases y en el SQL dinámico. Esto es razonable cuando, además de una consulta SQL por sí misma, también necesita realizar algunas operaciones adicionales, por ejemplo, ejecutar una consulta SQL en varios namespaces, o aumentar las autorizaciones antes de ejecutar la consulta.

Ejemplo de un SQL dinámico para Utils.CustomQuery:

```
Query AllDynamic() As %Query(CONTAINID = 1, ROWSPEC = "Id:%String,Prop1:%String,Prop2
:%Integer") [ SqlName = AllDynamic, SqlProc ]
{
}

ClassMethod AllDynamicExecute(ByRef qHandle As %Binary) As %Status
{
    Set qHandle = ##class(%SQL.Statement).%ExecDirect(,"SELECT * FROM Utils.CustomQue
ry")
    Quit $$$OK
}

ClassMethod AllDynamicFetch(ByRef qHandle As %Binary, ByRef Row As %List, ByRef AtEnd
 As %Integer = 0) As %Status
{
    If qHandle.%Next()=0 {
        Set AtEnd = 1
        Set Row = ""
        Quit $$$OK
    }
    Set Row = $Lb(qHandle.%Get("Id"), qHandle.%Get("Prop1"), qHandle.%Get("Prop2"))
    Quit $$$OK
}

ClassMethod AllDynamicClose(ByRef qHandle As %Binary) As %Status
{
    Kill qHandle
    Quit $$$OK
}
```

Método alternativo: %SQL.CustomResultSet

O bien, puede crear una consulta mediante una subclasificación desde la clase %SQL.CustomResultSet. Los beneficios de este método son los siguientes:

- Aumentar ligeramente la velocidad
- No es necesario utilizar ROWSPEC, ya que todos los metadatos se obtienen a partir de la definición de clase
- Cumplir con los principios del diseño orientado a objetos

Para crear una consulta desde la subclase que pertenece a la clase %SQL.CustomResultSet, asegúrese de realizar los siguientes pasos:

1. Definir las propiedades correspondientes en los campos resultantes
2. Definir las propiedades privadas donde se almacenará el contexto de la consulta
3. Anular el método %OpenCursor (de forma similar a queryNameExecute) que inicia el contexto. En caso de que se produzca algún error, establezca %SQLCODE y también %Message
4. Anular el método %Next (de forma similar a queryNameFetch), el cual obtiene el siguiente resultado. Complete las propiedades. El método devuelve 0 si se procesaron todos los datos, y 1 si todavía faltan algunos datos
5. Anular el método %CloseCursor (de forma similar a queryNameClose) si es necesario

Ejemplo de %SQL.CustomResultSet para Utils.CustomQuery:

```
Class Utils.CustomQueryRS Extends %SQL.CustomResultSet
{
Property Id As %String;
Property Prop1 As %String;
Property Prop2 As %Integer;
Method %OpenCursor() As %Library.Status
{
    Set ..Id = ""
    Quit $$$OK
}

Method %Next(ByRef sc As %Library.Status) As %Library.Integer [ PlaceAfter = %Execute
 ]
{
    Set sc = $$$OK
    Set ..Id = $Order(^Utils.CustomQueryD(..Id),1,val)
    Quit:..Id="" 0
    Set ..Prop1 = $Lg(val,2)
    Set ..Prop2 = $Lg(val,3)
    Quit $$$OK
}
}
```

Puede llamarlo desde el código Object Script de Caché, de la siguiente manera:

```
 Set resultset= ##class(Utils.CustomQueryRS).%New()
       While resultset.%Next() {
        Write resultset.Id,!
 }
```

Existe otro ejemplo disponible para el namespace SAMPLES, la clase Sample.CustomResultSet, que implementa una consulta para Samples.Person.

Resumen

Las consultas personalizadas le ayudarán a separar las expresiones SQL del código Object Script de Caché, e implementar un comportamiento sofisticado que puede ser muy difícil de manejar con el SQL puro.

Referenciass

Consultas de clases

Iteración mediante un global

SQL estático

SQL dinámico

%SQL.CustomResultSet

La clase Utils.CustomQuery

La clase Utils.CustomQueryRS

El autor quiere expresar su agradecimiento a Alexander Koblov por la ayuda brindada durante la redacción de este artículo.

#Compilador #Lenguajes #Modelo de datos de objetos #ObjectScript #SQL #Caché