Article
Guillaume Rongier · Juin 3, 2022 13m de lecture

Class Query dans InterSystems IRIS

Class Query dans InterSystems IRIS (et Cache, Ensemble, HealthShare) est un outil utile qui sépare les requêtes SQL du code Object Script. En principe, cela fonctionne comme suit : supposons que vous souhaitiez utiliser la même requête SQL avec différents arguments à plusieurs endroits différents. Dans ce cas, vous pouvez éviter la duplication du code en déclarant le corps de la requête comme une Class Query, puis en appelant cette requête par son nom. Cette approche est également pratique pour les requêtes personnalisées, dans lesquelles la tâche consistant à obtenir la ligne suivante est définie par un développeur. Cela vous intéresse ? Alors lisez la suite !

Class queries de base

Plus simplement, les Class Queries de base vous permettent de représenter des requêtes SQL SELECT. L'optimiseur et le compilateur SQL les traitent comme des requêtes SQL standards, mais elles sont plus pratiques lorsqu'il s'agit de les exécuter à partir du contexte Caché Object Script. Ils sont déclarés en tant qu'éléments de requête Query dans les définitions de classe (similaires aux méthodes ou aux propriétés) de la manière suivante :

- Type: %SQLQuery
- Tous les arguments de votre requête SQL doivent être énumérés dans la liste des arguments
- Type de requête: SELECT
- Utiliser les deux-points pour accéder à chaque argument (similaire au SQL statique)
- Définissez le paramètre ROWSPEC qui contient des informations sur les noms et les types de données des résultats de sortie ainsi que l'ordre des champs
- (Facultatif) Définissez le paramètre CONTAINID qui correspond à l'ordre numérique si le champ contient l'ID. Si vous n'avez pas besoin de renvoyer l'ID, n'attribuez pas de valeur à CONTAINID
- (Facultatif) Définissez le paramètre COMPILEMODE qui correspond au paramètre similaire en SQL statique et spécifie quand l'expression SQL doit être compilée. Lorsque ce paramètre est défini sur IMMEDIATE (par défaut), la requête sera compilée en même temps que la classe. Lorsque ce paramètre a la valeur DYNAMIC, la requête sera compilée avant sa première exécution (similaire au SQL dynamique)
- (Facultatif) Définissez le paramètre SELECTMODE qui spécifie le format des résultats de la requête
- Ajoutez la propriété SqlProc, si vous voulez appeler cette requête comme une procédure SQL.
- Définissez la propriété SqlName, si vous souhaitez renommer la requête. Le nom par défaut d'une requête dans le contexte SQL est le suivant : PackageName.ClassNameQueryName
- Caché Studio fournit l'assistant intégré pour la création de Class Query

Exemple de définition de la classe Sample.Person avec la requête ByName qui renvoie tous les noms d'utilisateur qui commencent par une lettre spécifiée

```
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
}
}
```

Vous pouvez appeler cette requête depuis Caché Object Script de la manière suivante :

```
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")
}
```

Vous pouvez également obtenir un ensemble de résultats en utilisant la méthode générée automatiquement queryNameFunc :

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

Cette requête peut également être appelée à partir du SQLcontext de ces deux manières :

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

Cette classe peut être trouvée dans l'espace de nom par défaut SAMPLES Caché. Et c'est tout pour les requêtes simples. Passons maintenant aux requêtes personnalisées

Class queries personnalisées

Bien que les Class Queries de base fonctionnent parfaitement dans la plupart des cas, il est parfois nécessaire d'exécuter un contrôle total sur le comportement des requêtes dans les applications, par exemple :

- Des critères de sélection sophistiqués. Puisque dans les requêtes personnalisées vous implémentez une méthode Caché Object Script qui renvoie la ligne suivante de façon autonome, ces critères peuvent être aussi sophistiqués que vous le souhaitez.
- Si les données sont accessibles uniquement via l'API dans un format que vous ne souhaitez pas utiliser
- Si les données sont stockées dans des globales (sans classes)
- Si vous avez besoin d'élever les droits afin d'accéder aux données
- Si vous devez appeler une API externe afin d'accéder à des données
- Si vous devez accéder au système de fichiers afin d'accéder aux données
- Vous devez effectuer des opérations supplémentaires avant d'exécuter la requête (par exemple, établir une connexion, vérifier les autorisations, etc.)

Alors, comment créer des requêtes de classes personnalisées ? Tout d'abord, vous devez définir 4 méthodes qui mettent en œuvre l'ensemble du flux de travail de votre requête, de l'initialisation à la destruction :

- queryName — fournit des informations sur une requête (similaire aux requêtes de classe de base)
- queryName**Execute** — construit une requête
- queryName**Fetch** — obtient le résultat de la ligne suivante d'une requête
- queryName**Close** — détruit une requête

Analysons maintenant ces méthodes plus en détail.

La méthode queryName

La méthode queryName représente des informations sur une requête

- Type: %Query
- Laissez le corps vide
- Définissez le paramètre ROWSPEC qui contient les informations sur les noms et les types de données des résultats de sortie ainsi que l'ordre des champs
- (Facultatif) Définissez le paramètre CONTAINID qui correspond à l'ordre numérique si le champ contient l'ID. Si vous ne renvoyez pas d'ID, n'attribuez pas de valeur à CONTAINID

Par exemple, créons la requête AllRecords (queryName = AllRecords, et la méthode est simplement appelée AllRecords) qui produira toutes les instances de la nouvelle classe persistante Utils.CustomQuery, une par une. Tout d'abord, créons une nouvelle classe persistante Utils.CustomQuery :

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

Maintenant, écrivons la requête AllRecords :

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

La méthode queryNameExecute

La méthode queryNameExecute initialise complètement une requête. La signature de cette méthode est la suivante :

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

où:

- qHandle est utilisé pour la communication avec les autres méthodes de l'implémentation de la requête
- Cette méthode doit mettre qHandle dans l'état qui sera ensuite transmis à la méthode queryNameFetch
- qHandle peut être défini comme OREF, une variable ou une variable multidimensionnelle
- Les args sont des paramètres supplémentaires transmis à la requête. Vous pouvez ajouter autant d'args que vous le souhaitez (ou ne pas les utiliser du tout)
- La méthode doit retourner le statut d'initialisation de la requête

Revenons à notre exemple. Vous pouvez itérer dans l'étendue de plusieurs façons (je décrirai plus loin les approches de travail de base pour les requêtes personnalisées), mais pour cet exemple, itérons dans la globale en utilisant la fonction $Order. Dans ce cas, qHandle stockera l'ID actuel, et puisque nous n'avons pas besoin d'arguments supplémentaires, l'argument arg n'est pas nécessaire. Le résultat est le suivant :

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

La méthode queryNameFetch

La méthode queryNameFetch renvoie un seul résultat sous la forme $List. La signature de cette méthode est la suivante :

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

where:

- qHandle est utilisé pour la communication avec les autres méthodes de l'implémentation de la requête
- Lorsque la requête est exécutée, les valeurs spécifiées par queryNameExecute ou par un appel précédent de queryNameFetch sont attribuées à qHandle.
- Le rang sera défini soit par une valeur de %List, soit par une chaîne vide, si toutes les données ont été traitées
- AtEnd doit être mis à 1, une fois que la fin des données est atteinte.
- La méthode "Fetch" doit être positionnée après la méthode "Execute", mais cela n'est important que pour SQL statique, c'est-à-dire les curseurs à l'intérieur des requêtes.

En général, les opérations suivantes sont effectuées dans le cadre de cette méthode :

1. Vérifier si nous avons atteint la fin des données
2. S'il reste encore des données : Créez une nouvelle %List et attribuez une valeur à la variable Row
3. Sinon, mettez AtEnd à 1
4. Préparer qHandle pour la prochaine récupération de résultat
5. Retourner l'état

Voici comment cela se présente dans notre exemple :

```
ClassMethod AllRecordsFetch(ByRef qHandle As %Binary, ByRef Row As %List, ByRef AtEnd
 As %Integer = 0) As %Status {
    #; itérer dans ^Utils.CustomQueryD
    #; ecrire le prochain id dans qHandle et écriture de la valeur de la globale avec
 le nouvel id dans val
    Set qHandle = $Order(^Utils.CustomQueryD(qHandle),1,val)
    #; Vérifier s'il reste des données
       If qHandle = "" {
        Set AtEnd = 1
        Set Row = ""
        Quit $$$OK
    }
    #; Si ce n'est pas le cas, créer %List
    #; val = $Lb("", Prop1, Prop2) voir définition de Storage
    #; Row =$lb(Id,Prop1, Prop2)  voir ROWSPEC pour la demande AllRecords
    Set Row = $Lb(qHandle, $Lg(val,2), $Lg(val,3))
    Quit $$$OK
}
```

La méthode queryNameClose

La méthode queryNameClose met fin à la requête, une fois toutes les données obtenues. La signature de cette méthode est la suivante :

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

où :

- Caché exécute cette méthode après le dernier appel à la méthode queryNameFetch
- En d'autres termes, il s'agit d'un destructeur de requête
- Par conséquent, vous devez disposer de tous les curseurs SQL, des requêtes et des variables locales dans sa mise en œuvre
- Les méthodes renvoient l'état actuel

Dans notre exemple, nous devons supprimer la variable locale qHandle :

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

Et voilà ! Une fois que vous aurez compilé la classe, vous serez en mesure d'utiliser la requête AllRecords à partir de %SQL.Statement - tout comme les requêtes de la classe de base.

Approches de la logique d'itération pour les requêtes personnalisées

Alors, quelles approches peuvent être utilisées pour les requêtes personnalisées ? En général, il existe 3 approches de base :

- Itération à travers une globale
- Static SQL
- Dynamic SQL

Itération à travers une globale
Cette approche est basée sur l'utilisation de $Order et de fonctions similaires pour l'itération à travers une globale. Elle peut être utilisée dans les cas suivants :

- Les données sont stockées dans des globales (sans classes)
- Vous voulez réduire le nombre de glorefs dans le code
- Les résultats doivent/peuvent être triés par l'indice de la globale

SQL statique
L'approche est basée sur les curseurs et le SQL statique. Elle est utilisée pour :

- Rendre le code int plus lisible
- Faciliter le travail avec les curseurs
- Accélération du processus de compilation (le SQL statique est inclus dans la requête de la classe et n'est donc compilé qu'une seule fois).

Remarque:

- Les curseurs générés à partir de requêtes du type %SQLQuery sont nommés automatiquement, par exemple Q14.
- Tous les curseurs utilisés dans une classe doivent avoir des noms différents
- Les messages d'erreur sont liés aux noms internes des curseurs qui comportent des caractères

supplémentaires à la fin de leur nom. Par exemple, une erreur dans le curseur Q140 est en fait causée par le curseur Q14.
- Utilisez PlaceAfter et assurez-vous que les curseurs sont utilisés dans la même routine int où ils ont été déclarés.
- INTO doit être utilisé en conjonction avec FETCH, mais pas DECLARE.

Exemple de SQL statique pour 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 doit être associé à FETCH
    &sql(FETCH C INTO :Id, :Prop1, :Prop2)
    #; Vérifier si la fin des données est atteinte
    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 dynamique

L'approche est basée sur les requêtes d'autres classes et le SQL dynamique. Cette approche est raisonnable lorsqu'en plus d'une requête SQL proprement dite, vous devez également effectuer certaines opérations supplémentaires, par exemple exécuter une requête SQL dans plusieurs espaces de noms ou escalader les permissions avant d'exécuter la requête.

Exemple de SQL dynamique pour 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
}
```

Approche alternative : %SQL.CustomResultSet

Vous pouvez également créer une requête en sous-classant la classe %SQL.CustomResultSet. Les avantages de cette approche sont les suivants :

- Une légère augmentation de la vitesse
- ROWSPEC est inutile, puisque toutes les métadonnées sont obtenues à partir de la définition de la classe
- Respect des principes de conception orientée objet

Pour créer une requête à partir de la sous-classe de la classe %SQL.CustomResultSet, assurez-vous d'effectuer les étapes suivantes :

1. Définir les propriétés correspondant aux champs résultants
2. Définir les propriétés privées où le contexte de la requête sera stocké
3. Remplacer la méthode %OpenCursor (similaire à queryNameExecute) qui initie le contexte. En cas d'erreur, définissez également %SQLCODE et %Message
4. Remplacer la méthode %Next (similaire à queryNameFetch) qui obtient le résultat suivant. Remplacer les propriétés. La méthode renvoie 0 si toutes les données ont été traitées et 1 s'il reste des données
5. Remplacer la méthode %CloseCursor (similaire à queryNameClose) si nécessaire

Exemple de %SQL.CustomResultSet pour 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
}
}
```

Vous pouvez l'appeler à partir de Caché Object Script code de la manière suivante :

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

Un autre exemple est disponible dans l'espace de noms SAMPLES - il s'agit de la classe Sample.CustomResultSet qui implémente une requête pour Samples.Person.

Résumé

Les requêtes personnalisées vous aideront à séparer les expressions SQL du code Caché Object Script et à mettre en œuvre un comportement sophistiqué qui peut être trop difficile pour le SQL pur.

Références

Class Queries

Itération à travers une globale

SQL statique

Dynamic SQL

%SQL.CustomResultSet

Classe Utils.CustomQuery

Classe Utils.CustomQueryRS

L'auteur tient à remercier [Alexander Koblov] (https://community.intersystems.com/user/alexander-koblov) pour son aide à la composition de cet article.

#Bonnes pratiques #Compilateur #Langues #Object Data Model #ObjectScript #SQL #Caché

URL de la source:https://fr.community.intersystems.com/post/class-query-dans-intersystems-iris