Procédure pas à pas : écriture de requêtes en C# (LINQ)

Visual Studio 2012

Cette procédure pas à pas présente les fonctionnalités du langage C# utilisées pour écrire des expressions de requête LINQ. Une fois cette procédure pas à pas terminée, vous serez prêt à passer aux exemples et à la documentation du fournisseur LINQ spécifique qui vous intéresse, tel que LINQ to SQL, LINQ to DataSets ou LINQ to XML.

Composants requis

Cette procédure pas - à - pas requiert des fonctionnalités qui sont présentées dans Visual Studio 2008.

 Pour obtenir une version vidéo de cette rubrique, consultez Vidéo : comment écrire des requêtes LINQ en C# (page éventuellement en anglais).

Création d'un projet C#.

Pour créer un projet

1. Démarrez Visual Studio.
2. Dans la barre de menus, sélectionnez Fichier, Nouveau, Project.

   La boîte de dialogue Nouveau projet s'affiche.

3. Développez Installé, développez Modèles, développez Visual C#, puis choisissez Application console.
4. Dans la zone de texte Nom, entrez un nom différent ou acceptez le nom par défaut, puis choisissez le bouton OK .

   Le nouveau projet s'affiche dans l'Explorateur de solutions.

5. Notez que votre projet a une référence à System.Core.dll et une directive using pour l'espace de noms System.Linq.

Création d'une source de données en mémoire

La source de données pour les requêtes est une simple liste d'objets Student. Chaque enregistrement Student comporte un prénom, un nom et un tableau d'entiers représentant les notes d'examens en classe. Copiez ce code dans votre projet. Notez les caractéristiques suivantes :

- La classe Student se compose de propriétés implémentées automatiquement.
- Chaque étudiant de la liste est initialisé avec un initialiseur d'objet.
- La liste elle-même est initialisée avec un initialiseur de collection.

La totalité de cette structure de données sera initialisée et instanciée sans appels explicites à un constructeur quelconque ou accès au membre explicite. Pour plus d'informations sur ces nouvelles fonctionnalités, consultez Propriétés implémentées automatiquement (Guide de programmation C#) et Initialiseurs d'objets et de collection (Guide de programmation C#).

Pour ajouter la source de données

- Ajoutez la classe Student et la liste initialisée d'étudiants à la classe Program dans votre projet.

  C#

```
public class Student
{
    public string First { get; set; }
    public string Last { get; set; }
    public int ID { get; set; }
    public List<int> Scores;
}

// Create a data source by using a collection initializer.
static List<Student> students = new List<Student>
{
   new Student {First="Svetlana", Last="Omelchenko", ID=111, Scores=
new List<int> {97, 92, 81, 60}},
   new Student {First="Claire", Last="O'Donnell", ID=112, Scores= new
List<int> {75, 84, 91, 39}},
   new Student {First="Sven", Last="Mortensen", ID=113, Scores= new
List<int> {88, 94, 65, 91}},
   new Student {First="Cesar", Last="Garcia", ID=114, Scores= new
List<int> {97, 89, 85, 82}},
   new Student {First="Debra", Last="Garcia", ID=115, Scores= new
List<int> {35, 72, 91, 70}},
   new Student {First="Fadi", Last="Fakhouri", ID=116, Scores= new
List<int> {99, 86, 90, 94}},
   new Student {First="Hanying", Last="Feng", ID=117, Scores= new
List<int> {93, 92, 80, 87}},
   new Student {First="Hugo", Last="Garcia", ID=118, Scores= new
List<int> {92, 90, 83, 78}},
   new Student {First="Lance", Last="Tucker", ID=119, Scores= new
List<int> {68, 79, 88, 92}},
   new Student {First="Terry", Last="Adams", ID=120, Scores= new
List<int> {99, 82, 81, 79}},
   new Student {First="Eugene", Last="Zabokritski", ID=121, Scores=
new List<int> {96, 85, 91, 60}},
   new Student {First="Michael", Last="Tucker", ID=122, Scores= new
List<int> {94, 92, 91, 91} }
};
```

Pour ajouter un nouvel étudiant à la liste des étudiants

- Ajoutez un nouveau Student à la liste Students et utilisez le nom et les notes d'examens de votre choix. Essayez d'entrer toutes les informations relatives au nouvel étudiant pour mieux apprendre la syntaxe de l'initialiseur d'objet.

Création de la requête

Pour créer une requête simple

- Dans la méthode Main de l'application, créez une requête simple qui, lorsqu'elle est exécutée, produit une liste de tous les étudiants dont la note pour le premier test est supérieure à 90. Notez que, puisque l'objet Student entier est sélectionné, le type de la requête est IEnumerable<Student>. Bien que le code puisse également utiliser un type implicite à l'aide du mot clé var, les types explicites sont utilisés pour illustrer clairement les résultats. Pour plus d'informations sur var, consultez Variables locales implicitement typées (Guide de programmation C#).

  Notez également que la variable de portée de la requête, student, sert de référence à chaque Student dans la source, en fournissant l'accès au membre pour chaque objet.

C#

```
// Create the query.
// The first line could also be written as "var studentQuery ="
IEnumerable<Student> studentQuery =
    from student in students
    where student.Scores[0] > 90
    select student;
```

Exécution de la requête

Pour exécuter la requête

1. Écrivez maintenant la boucle foreach qui entraînera l'exécution de la requête. Notez les points suivants concernant le code :
   - Chaque élément de la séquence retournée est accessible via la variable d'itération dans la boucle foreach.
   - Le type de cette variable est Student et le type de la variable de requête est compatible (IEnumerable<Student>).
2. Après avoir ajouté ce code, générez et exécutez l'application en appuyant sur Ctrl + F5 pour visualiser les résultats dans la fenêtre de console.

C#

```
// Execute the query.
// var could be used here also.
foreach (Student student in studentQuery)
{
    Console.WriteLine("{0}, {1}", student.Last, student.First);
}
```

Pour ajouter une autre condition de filtre

- Vous pouvez associer plusieurs conditions booléennes dans la clause where pour affiner davantage une requête. Le code suivant ajoute une condition afin que la requête retourne les étudiants dont la première note est supérieure à 90 et dont la dernière est inférieure à 80. La clause where doit être semblable au code suivant.
- `where student.Scores[0] > 90 && student.Scores[3] < 80`

  Pour plus d'informations, consultez where, clause (Référence C#).

Modifier la requête

Pour classer les résultats

1. Il sera plus facile d'analyser les résultats s'ils sont classés. Vous pouvez classer la séquence retournée selon tous les champs accessibles dans les éléments sources. Par exemple, la clause orderby suivante classe les résultats dans l'ordre alphabétique de A à Z d'après le nom de chaque étudiant. Ajoutez la clause orderby suivante à votre requête, juste après l'instruction where et avant l'instruction select :
2. `orderby student.Last ascending`
3. Modifiez à présent la clause orderby pour qu'elle classe les résultats dans l'ordre inverse d'après la note au premier examen, de la plus élevée à la plus faible.
4. `orderby student.Scores[0] descending`
5. Modifiez la chaîne de format WriteLine pour pouvoir consulter les notes :
6. `Console.WriteLine("{0}, {1} {2}", student.Last, student.First, student.Scores[0]);`

   Pour plus d'informations, consultez orderby, clause (Référence C#).

Pour regrouper les résultats

1. Le regroupement est une fonction puissante des expressions de requête. Une requête avec une clause group produit une séquence de groupes dont chaque groupe contient lui-même un Key et une séquence composée de tous les membres de ce groupe. La nouvelle requête suivante regroupe les étudiants en utilisant la première lettre de leur nom comme clé.

   C#

   ```
   // studentQuery2 is an IEnumerable<IGrouping<char, Student>>
   var studentQuery2 =
       from student in students
       group student by student.Last[0];
   ```

2. Notez que le type de la requête a maintenant changé. Il produit désormais une séquence de groupes qui ont un type char comme clé et une séquence d'objets Student. Étant donné que le type de la requête a changé, le code suivant modifie également la boucle d'exécution foreach :

   C#

```
// studentGroup is a IGrouping<char, Student>
foreach (var studentGroup in studentQuery2)
{
    Console.WriteLine(studentGroup.Key);
    foreach (Student student in studentGroup)
    {
        Console.WriteLine("   {0}, {1}",
                  student.Last, student.First);
    }
}
```

3. Appuyez sur Ctrl + F5 pour exécuter l'application et visualiser les résultats dans la fenêtre de console.

   Pour plus d'informations, consultez group, clause (Référence C#).

Pour rendre les variables implicitement typées

- Le codage explicite de **IEnumerables** de **IGroupings** peut rapidement s'avérer laborieux. Vous pouvez écrire la même requête et la même boucle foreach beaucoup plus facilement à l'aide de var. Le mot clé var ne modifie pas les types de vos objets ; il indique simplement au compilateur de déduire les types. Modifiez le type d' studentQuery et l'itération groupvariable par var et réexécuter la requête. Notez que dans la boucle foreach interne, la variable d'itération est encore typée comme Student et que la requête fonctionne exactement comme précédemment. Modifiez la variable d'itération s en var et exécutez à nouveau la requête. Vous pouvez constater que les résultats obtenus sont exactement les mêmes.

  C#

```
var studentQuery3 =
    from student in students
    group student by student.Last[0];

foreach (var groupOfStudents in studentQuery3)
{
    Console.WriteLine(groupOfStudents.Key);
    foreach (var student in groupOfStudents)
    {
         Console.WriteLine("   {0}, {1}",
             student.Last, student.First);
    }
}
```

  Pour plus d'informations sur var, consultez Variables locales implicitement typées (Guide de programmation C#).

Pour classer les groupes selon leur valeur de clé

- Lorsque vous exécutez la requête précédente, vous remarquez que les groupes ne sont pas dans l'ordre alphabétique. Pour modifier cet ordre, vous devez fournir une clause orderby après la clause group. Toutefois, pour utiliser une clause orderby, vous avez d'abord besoin d'un identificateur servant de référence aux groupes créés par la clause group. Fournissez l'identificateur à l'aide du mot clé into, comme suit :

  C#

```
var studentQuery4 =
    from student in students
    group student by student.Last[0] into studentGroup
    orderby studentGroup.Key
    select studentGroup;

foreach (var groupOfStudents in studentQuery4)
{
    Console.WriteLine(groupOfStudents.Key);
    foreach (var student in groupOfStudents)
    {
        Console.WriteLine("   {0}, {1}",
            student.Last, student.First);
    }
}
```

  Lorsque vous exécuterez cette requête, vous verrez que les groupes seront désormais classés dans l'ordre alphabétique.

Pour introduire un identificateur à l'aide de let

- Vous pouvez utiliser le mot clé let pour introduire un identificateur pour tous les résultats d'expression dans l'expression de requête. Cet identificateur peut être pratique, comme dans l'exemple suivant, ou améliorer les performances en stockant les résultats d'une expression afin d'éviter qu'ils ne soient calculés plusieurs fois.

  C#

```
// studentQuery5 is an IEnumerable<string>
// This query returns those students whose
// first test score was higher than their
// average score.
var studentQuery5 =
    from student in students
    let totalScore = student.Scores[0] + student.Scores[1] +
        student.Scores[2] + student.Scores[3]
    where totalScore / 4 < student.Scores[0]
    select student.Last + " " + student.First;

foreach (string s in studentQuery5)
{
    Console.WriteLine(s);
}
```

Pour plus d'informations, consultez let, clause (Référence C#).

Pour utiliser la syntaxe de méthode dans une expression de requête

- Comme décrit dans Syntaxe de requête et syntaxe de méthode dans LINQ (C#), certaines opérations de requête peuvent être exprimées uniquement à l'aide de la syntaxe de méthode. Le code suivant calcule la note totale de chaque Student dans la séquence source, puis appelle la méthode **Average()** sur les résultats de cette requête pour calculer la note moyenne de la classe. Notez la présence de parenthèses de part et d'autre de l'expression de requête.

  C#

```
    var studentQuery6 =
        from student in students
        let totalScore = student.Scores[0] + student.Scores[1] +
            student.Scores[2] + student.Scores[3]
        select totalScore;

    double averageScore = studentQuery6.Average();
Console.WriteLine("Class average score = {0}", averageScore);
```

Pour transformer ou projeter la clause select

1. Il n'est pas rare qu'une requête produise une séquence dont les éléments diffèrent des éléments des séquences sources. Supprimez ou commentez votre requête et votre boucle d'exécution précédentes et remplacez-les par le code suivant. Notez que la requête retourne une séquence de chaînes (pas de Students) et que ce fait est répercuté dans la boucle foreach.

   C#

```
IEnumerable<string> studentQuery7 =
    from student in students
    where student.Last == "Garcia"
    select student.First;

Console.WriteLine("The Garcias in the class are:");
foreach (string s in studentQuery7)
{
    Console.WriteLine(s);
}
```

2. Le code utilisé précédemment dans cette procédure pas à pas indique que la note moyenne de la classe est d'environ 334. Pour produire une séquence de Students dont

la note totale est supérieure à la moyenne de classe, avec les Student ID correspondants, vous pouvez utiliser un type anonyme dans l'instruction select :

C#

```
            var studentQuery8 =
                from student in students
                let x = student.Scores[0] + student.Scores[1] +
                    student.Scores[2] + student.Scores[3]
                where x > averageScore
                select new { id = student.ID, score = x };

            foreach (var item in studentQuery8)
            {
                Console.WriteLine("Student ID: {0}, Score: {1}",
item.id, item.score);
            }
```

Étapes suivantes

Une fois que vous êtes familiarisé avec les aspects de base de l'utilisation de requêtes en C#, vous êtes prêt à lire la documentation et les exemples pour le type spécifique de fournisseur LINQ qui vous intéresse :

LINQ to SQL

LINQ to DataSet

LINQ to XML

LINQ to Objects