Tutorial: Escribir consultas en C# (LINQ)

.NET Framework 3.5

Actualización: noviembre 2007

Este tutorial le guiará a través de las nuevas características del lenguaje de C# 3.0 y le mostrará cómo se utilizan para escribir expresiones de consulta LINQ. Después de completar este tutorial, estará preparado para ver los ejemplos y la documentación del proveedor LINQ específico que le interese, como LINQ to SQL, LINQ to DataSets o LINQ to XML.

Requisitos previos

Para este tutorial se requiere Visual Studio 2008.

Para ver una demostración en vídeo, visite Video How to: Writing Queries in C# (LINQ).

Crear un proyecto de C#

Para crear un proyecto de C# orientado a la versión 3.5 de .NET Framework

1. Inicie Visual Studio.
2. En el menú Archivo, elija Nuevo y, a continuación, haga clic en Proyecto.
3. En la esquina superior derecha del cuadro de diálogo Nuevo proyecto hay tres iconos. Haga clic en el icono izquierdo y asegúrese de que esté activada la opción .NET Framework versión 3.5.
4. Haga clic en el icono Aplicación de consola en Plantillas instaladas de Visual Studio.
5. Asigne un nuevo nombre a su aplicación o acepte el nombre predeterminado y haga clic en Aceptar.
6. Observe que su proyecto tiene una referencia a System.Core.dll y una directiva **using** para el espacio de nombres System.Linq.

Crear un origen de datos en memoria

El origen de datos de las consultas es una simple lista de objetos `Student`. Cada registro `Student` tiene un nombre, un apellido y una matriz de enteros que representa sus puntuaciones de exámenes en la clase. Copie este código en el proyecto. Observe las siguientes características:

- La clase `Student` está formada por propiedades autoimplementadas.
- Cada estudiante de la lista se inicializa con un inicializador de objeto.
- La propia lista se inicializa con un inicializador de colección.

Esta estructura de datos completa se inicializará, y se crearán instancias de ella, sin llamadas explícitas a un constructor o sin acceso a miembros explícito. Para obtener más información acerca de estas nuevas características, vea Propiedades

autoimplementadas (Guía de programación de C#) y Inicializadores de objeto y de colección (Guía de programación de C#).

Para agregar el origen de datos

- Agregue la clase `Student` y la lista inicializada de estudiantes a la clase `Program` de su proyecto.

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} }
};
```

Para agregar un nuevo estudiante a la lista de estudiantes

- Agregue un nuevo `Student` a la lista `Students` y utilice el nombre y las puntuaciones de examen que desee. Intente escribir toda la información del nuevo estudiante para aprender mejor la sintaxis del inicializador de objeto.

Crear la consulta

Para crear una consulta sencilla

- En el método `Main` de la aplicación, cree una consulta simple que, cuando se ejecute, genere una lista de todos los estudiantes cuya puntuación fue mayor que 90 en el primer examen. Tenga en cuenta que, como se selecciona el objeto `Student` completo, el tipo de la consulta es `IEnumerable<Student>`. Aunque en el código también se podría utilizar un tipo implícito mediante la palabra clave var, se usa un tipo explícito para ilustrar los resultados de forma más clara. (Para obtener más información acerca de **var**, vea Variables locales con asignación implícita de tipos (Guía de programación de C#).)

  Observe también que la variable de rango de la consulta, `student`, sirve de referencia para cada `Student` del origen, proporcionando acceso a los miembros de cada objeto.

C#

```
// Create the query.
// studentQuery is an IEnumerable<Student>
var studentQuery =
    from student in students
    where student.Scores[0] > 90
    select student;
```

Ejecutar la consulta

Para ejecutar la consulta

1. Ahora escriba el bucle **foreach** que hará que se ejecute la consulta. Tenga en cuenta lo siguiente acerca del código:
   - Se tiene acceso a cada elemento de la secuencia devuelta a través de la variable de iteración del bucle **foreach**.
   - El tipo de esta variable es `Student`, y el tipo de la variable de consulta es compatible, `IEnumerable<Student>`.
2. Después de haber agregado este código, genere y ejecute la aplicación; para ello, presione Ctrl + F5. Los resultados aparecerán en la ventana Consola.

C#

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

Para agregar otra condición de filtro

- Puede combinar varias condiciones booleanas en la cláusula **where** para delimitar más la consulta. El código siguiente agrega una condición para que la consulta devuelva los estudiantes cuya primera puntuación fue superior a 90 y cuya última puntuación fue inferior a 80. La cláusula **where** se debería parecer a la del código siguiente.

```
where student.Scores[0] > 90 && student.Scores[3] < 80
```

Para obtener más información, consulte where (Cláusula, Referencia de C#).

Modificar la consulta

Para ordenar los resultados

1. Será más fácil examinar los resultados si están ordenados de alguna manera. Puede ordenar la secuencia devuelta según cualquier campo accesible de los elementos de origen. Por ejemplo, la cláusula **orderby** siguiente ordena los resultados alfabéticamente, de la A a la Z, por apellido de estudiante. Agregue la siguiente cláusula **orderby** a la consulta, justo detrás de la instrucción **where** y delante de la instrucción **select**:

```
orderby student.Last ascending
```

2. Ahora cambie la cláusula **orderby** de forma que ordene los resultados en orden inverso, según la puntuación del primer examen, de mayor a menor.

```
orderby student.Scores[0] descending
```

3. Cambie la cadena de formato `WriteLine` para poder ver las puntuaciones:

```
Console.WriteLine("{0}, {1} {2}", s.Last, s.First, s.Scores[0]);
```

Para obtener más información, consulte orderby (Cláusula, Referencia de C#).

Para agrupar los resultados

1. La agrupación es una funcionalidad eficaz para las expresiones de consulta. Una consulta con una cláusula group genera una secuencia de grupos donde cada grupo contiene una `Key` y una secuencia compuesta por todos los miembros de ese grupo. La siguiente consulta nueva agrupa los estudiantes utilizando como clave la inicial de su apellido.

C#

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

2. Observe que el tipo de la consulta ha cambiado. Ahora genera una secuencia de grupos que tienen un tipo **char** como clave y una secuencia de objetos `Student`. Dado que el tipo de la consulta ha cambiado, el código siguiente también cambia el bucle de ejecución **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. Presione Ctrl + F5 para ejecutar la aplicación y ver los resultados en la ventana Consola.

Para obtener más información, consulte group (Cláusula, Referencia de C#).

Para hacer que las variables sean de tipo implícito

- Codificar explícitamente **IEnumerables** de **IGroupings** se puede convertir pronto en una tarea complicada. Puede escribir la misma consulta y el mismo bucle **foreach** de una forma mucho más sencilla utilizando **var**. La palabra clave **var** no cambia los tipos de los objetos; simplemente indica al compilador que los deduzca. Cambie el tipo de `studentQuery` y la variable de iteración `group` a **var** y vuelva a ejecutar la consulta. Observe que en el bucle **foreach** interno, la variable de iteración todavía es de tipo `Student` y la consulta funciona igual que antes. Cambie la variable de iteración `s` a **var** y ejecute de nuevo la consulta. Verá que obtiene exactamente los mismos resultados.

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);
    }
}
```

Para obtener más información acerca de var, vea Variables locales con asignación implícita de tipos (Guía de programación de C#).

Para ordenar los grupos por su valor de clave

- Al ejecutar la consulta anterior, verá que los grupos no están ordenados alfabéticamente. Para cambiar esto, debe proporcionar una cláusula **orderby** después de la cláusula **group**. Sin embargo, para usar una cláusula **orderby**, primero necesita un identificador que sirva de referencia para los grupos creados por la cláusula **group**. Proporcione el identificador utilizando la palabra clave **into**, como se indica a continuación:

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);
    }
}
```

Al ejecutar esta consulta, verá que ahora los grupos están ordenados alfabéticamente.

Incluir un identificador mediante let

- Puede utilizar la palabra clave **let** para incluir un identificador para cualquier resultado de expresión de la expresión de consulta. Este identificador puede usarse por comodidad, como en el ejemplo siguiente, o para mejorar el rendimiento, ya que almacena los resultados de una expresión para que no tenga que calcularse varias veces.

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);
}
```

Para obtener más información, consulte let (Cláusula, Referencia de C#).

Para utilizar la sintaxis de método en una expresión de consulta

- Como se describe en Sintaxis de consultas y Sintaxis de métodos (LINQ), algunas operaciones de consulta sólo se pueden expresar utilizando sintaxis de método. El código siguiente calcula la puntuación total de cada `Student` de la secuencia de origen y, después, llama al método **Average()** en los resultados de esa consulta para calcular la puntuación promedio de la clase. Observe la posición de los paréntesis alrededor de la expresión de consulta.

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);
```

Para transformar o proyectar en la cláusula select

1. Es muy común que una consulta genere una secuencia cuyos elementos difieren de los elementos de las secuencias de origen. Elimine o marque como comentario la consulta y el bucle de ejecución anteriores y reemplácelos con el código siguiente. Observe que la consulta devuelve una secuencia de cadenas (no `Students`) y este hecho se refleja en el bucle **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. El código anterior de este tutorial indicaba que la puntuación promedio de la clase es de 334, aproximadamente. Para generar una secuencia de `Students` cuya puntuación total sea mayor que la media de la clase, junto con su `Student ID`, puede utilizar un tipo anónimo en la instrucción **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);
    }
```

Pasos siguientes

Una vez que esté familiarizado con los aspectos básicos del uso de consultas in C#, estará preparado para leer la documentación y ver los ejemplos del tipo de proveedor LINQ específico que le interese:

LINQ to SQL

LINQ to DataSet

LINQ to XML

LINQ to Objects

Ejemplos de LINQ en C#

Vea también

Conceptos

Expresiones de consultas con LINQ (Guía de programación de C#)

Recursos suplementarios de LINQ

Otros recursos

Language-Integrated Query (LINQ)

Introducción a LINQ en C#

Adiciones de comunidad