Escritura de consultas LINQ de C# para consultar datos

Artículo • 19/12/2023

La mayoría de las consultas de la documentación introductoria de Language Integrated Query (LINK) se escribe con la sintaxis de consulta declarativa de LINQ. Pero la sintaxis de consulta debe traducirse en llamadas de método para .NET Common Language Runtime (CLR) al compilar el código. Estas llamadas de método invocan los operadores de consulta estándar, que tienen nombres tales como `Where`, `Select`, `GroupBy`, `Join`, `Max` y `Average`. Puede llamarlas directamente con la sintaxis de método en lugar de la sintaxis de consulta.

La sintaxis de consulta y la sintaxis de método son idénticas desde el punto de vista semántico, pero la sintaxis de consulta suele ser mucho más sencilla y fácil de leer. Algunos métodos deben expresarse como llamadas de método. Por ejemplo, debe usar una llamada de método para expresar una consulta que recupera el número de elementos que cumplen una condición especificada. También debe usar una llamada de método para una consulta que recupera el elemento que tiene el valor máximo de una secuencia de origen. La documentación de referencia de los operadores de consulta estándar del espacio de nombres System.Linq generalmente usa la sintaxis de método. Debería familiarizarse con cómo usar la sintaxis del método en consultas y en las propias expresiones de consulta.

Métodos de extensión de operador de consulta estándar

En el ejemplo siguiente se muestra una *expresión de consulta* sencilla y la consulta equivalente desde el punto de vista semántico que se escribe como *consulta basada en métodos*.

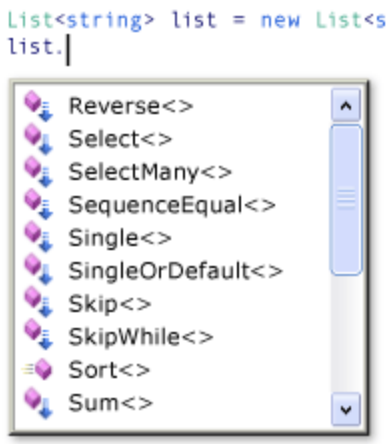
C#

```
int[] numbers = [ 5, 10, 8, 3, 6, 12 ];

//Query syntax:
IEnumerable<int> numQuery1 =
    from num in numbers
    where num % 2 == 0
    orderby num
    select num;
```

```
//Method syntax:
IEnumerable<int> numQuery2 = numbers.Where(num => num % 2 == 0).OrderBy(n =>
n);

foreach (int i in numQuery1)
{
    Console.Write(i + " ");
}
Console.WriteLine(System.Environment.NewLine);
foreach (int i in numQuery2)
{
    Console.Write(i + " ");
}
```

El resultado de los dos ejemplos es idéntico. Como puede ver, el tipo de variable de consulta es el mismo en ambos formularios: IEnumerable<T>.

Para entender la consulta basada en métodos, vamos a examinarla más detenidamente. En el lado derecho de la expresión, observe que la cláusula `where` ahora se expresa como un método de instancia en el objeto `numbers`, que tiene un tipo de `IEnumerable<int>`. Si está familiarizado con la interfaz genérica IEnumerable<T>, sabrá que no tiene un método `Where`. Pero si se invoca la lista de finalización de IntelliSense en el IDE de Visual Studio, verá no solo un método `Where`, sino muchos otros métodos tales como `Select`, `SelectMany`, `Join` y `Orderby`. Estos métodos implementan los operadores de consulta estándar.



Aunque parece que IEnumerable<T> incluye métodos adicionales, no lo hace. Los operadores de consulta estándar se implementan como *métodos de extensión*. Los métodos de extensión "extienden" un tipo existente; se pueden llamar como si fueran métodos de instancia en el tipo. Los operadores de consulta estándar extienden IEnumerable<T> y esta es la razón por la que la puede escribir `numbers.Where(...)`.

Para usar métodos de extensión, es necesario incluirlos en el ámbito de las directivas `using`. Desde el punto de vista de la aplicación, un método de extensión y un método de instancia normal son iguales.

Para obtener más información sobre los métodos de extensión, vea Métodos de extensión. Para obtener más información sobre los operadores de consulta estándar, vea Información general sobre operadores de consulta estándar (C#). Algunos proveedores LINQ, como Entity Framework y LINQ to XML, implementan sus propios operadores de consulta estándar y métodos de extensión para otros tipos además de IEnumerable<T>.

Expresiones lambda

En el ejemplo anterior, observe que la expresión condicional (`num % 2 == 0`) se pasa como argumento insertado al método Enumerable.Where: `Where(num => num % 2 == 0)`. Esta expresión insertada se denomina expresión lambda. Es una manera cómoda de escribir código que, de lo contrario, tendría que escribirse de forma más complicada. La `num` situada a la izquierda del operador es la variable de entrada que corresponde a `num` en la expresión de consulta. El compilador puede deducir el tipo de `num` porque sabe que `numbers` es un tipo IEnumerable<T> genérico. El cuerpo de la expresión lambda es exactamente igual que la expresión de la sintaxis de consulta o de cualquier otra expresión o instrucción de C#. Puede incluir llamadas de método y otra lógica compleja. El "valor devuelto" es simplemente el resultado de la expresión. Determinadas consultas solo se pueden expresar en sintaxis de método y algunas requieren expresiones lambda. Las expresiones lambda son una herramienta eficaz y flexible en el cuadro de herramientas de LINQ.

Capacidad de composición de consultas

En el ejemplo de código anterior, el método Enumerable.OrderBy se invoca mediante el operador de punto en la llamada a `Where`. `Where` genera una secuencia filtrada y, a continuación, `Orderby` ordena la secuencia generada por `Where`. Dado que las consultas devuelven un `IEnumerable`, redáctelas con la sintaxis de método encadenando las llamadas de método. El compilador realiza esta composición al escribir consultas mediante la sintaxis de consulta. Dado que una variable de consulta no almacena los resultados de la consulta, es posible modificarla o usarla como base para una nueva consulta en cualquier momento, incluso después de ejecutarla.

Los ejemplos siguientes muestran algunas consultas LINQ sencillas mediante cada enfoque enumerado anteriormente.

> **Nota**
>
> Estas consultas funcionan en colecciones en memoria simples, pero la sintaxis básica es idéntica a la empleada en LINQ to Entities y LINQ to XML.

Ejemplo: Sintaxis de consulta

La mayoría de las consultas se escriben con *sintaxis de consulta* para crear *expresiones de consulta*. En el siguiente ejemplo se muestran tres expresiones de consulta. La primera expresión de consulta muestra cómo filtrar o restringir los resultados mediante la aplicación de condiciones con una cláusula `where`. Devuelve todos los elementos de la secuencia de origen cuyos valores sean mayores que 7 o menores que 3. La segunda expresión muestra cómo ordenar los resultados devueltos. La tercera expresión muestra cómo agrupar los resultados según una clave. Esta consulta devuelve dos grupos en función de la primera letra de la palabra.

C#

```
List<int> numbers = [5, 4, 1, 3, 9, 8, 6, 7, 2, 0];

// The query variables can also be implicitly typed by using var

// Query #1.
IEnumerable<int> filteringQuery =
    from num in numbers
    where num is < 3 or > 7
    select num;

// Query #2.
IEnumerable<int> orderingQuery =
    from num in numbers
    where num is < 3 or > 7
    orderby num ascending
    select num;

// Query #3.
string[] groupingQuery = ["carrots", "cabbage", "broccoli", "beans", "barley"];
IEnumerable<IGrouping<char, string>> queryFoodGroups =
```

```
    from item in groupingQuery
    group item by item[0];
```

El tipo de las consultas es IEnumerable<T>. Todas estas consultas podrían escribirse mediante var como se muestra en el ejemplo siguiente:

```
var query = from num in numbers...
```

En cada ejemplo anterior, las consultas no se ejecutan realmente hasta que se recorre en iteración la variable de consulta en una instrucción foreach o cualquier otra instrucción.

Ejemplo: Sintaxis de método

Algunas operaciones de consulta deben expresarse como una llamada a método. Los más comunes de dichos métodos son aquellos que devuelven valores numéricos de singleton, como Sum, Max, Min, Average y así sucesivamente. A estos métodos siempre se los debe llamar en último lugar en cualquier consulta porque devuelven un solo valor y no pueden servir como origen para una operación de consulta adicional. En el ejemplo siguiente se muestra una llamada a método en una expresión de consulta:

C#

```
List<int> numbers1 = [5, 4, 1, 3, 9, 8, 6, 7, 2, 0];
List<int> numbers2 = [15, 14, 11, 13, 19, 18, 16, 17, 12, 10];

// Query #4.
double average = numbers1.Average();

// Query #5.
IEnumerable<int> concatenationQuery = numbers1.Concat(numbers2);
```

Si el método tiene parámetros System.Action o System.Func<TResult>, se proporcionan estos argumentos en forma de expresión lambda, tal y como se muestra en el ejemplo siguiente:

C#

```
// Query #6.
IEnumerable<int> largeNumbersQuery = numbers2.Where(c => c > 15);
```

En las consultas anteriores, solo la consulta n.º 4 se ejecuta inmediatamente, ya que devuelve un solo valor y no una colección IEnumerable<T> genérica. El propio método usa `foreach` o código similar para procesar su valor.

Cada una de las consultas anteriores puede escribirse mediante tipos implícitos con `var`, tal y como se muestra en el ejemplo siguiente:

C#

```
// var is used for convenience in these queries
double average = numbers1.Average();
var concatenationQuery = numbers1.Concat(numbers2);
var largeNumbersQuery = numbers2.Where(c => c > 15);
```

Ejemplo: Sintaxis de consulta y método combinada

En este ejemplo se muestra cómo usar la sintaxis de método en los resultados de una cláusula de consulta. Simplemente escriba entre paréntesis la expresión de consulta y luego aplique el operador punto y llame al método. En el ejemplo siguiente la consulta número 7 devuelve un recuento de los números cuyo valor está comprendido entre 3 y 7. Sin embargo, en general es mejor usar una segunda variable para almacenar el resultado de la llamada de método. De esta manera es menos probable que la consulta se confunda con los resultados de la consulta.

C#

```
// Query #7.

// Using a query expression with method syntax
var numCount1 = (
    from num in numbers1
    where num is > 3 and < 7
    select num
).Count();

// Better: Create a new variable to store
// the method call result
IEnumerable<int> numbersQuery =
    from num in numbers1
    where num is > 3 and < 7
    select num;
```

```
var numCount2 = numbersQuery.Count();
```

Dado que la consulta número 7 devuelve un solo valor y no una colección, se ejecuta inmediatamente.

La consulta anterior puede escribirse mediante tipos implícitos con `var` como sigue:

C#

```
var numCount = (from num in numbers...
```

Puede escribirse en sintaxis de método como sigue:

C#

```
var numCount = numbers.Count(n => n is > 3 and < 7);
```

Puede escribirse mediante tipos explícitos como sigue:

C#

```
int numCount = numbers.Count(n => n is > 3 and < 7);
```

Consulte también

- Tutorial: Escribir consultas en C#
- where (cláusula)

Colaborar con nosotros en GitHub

El origen de este contenido se puede encontrar en GitHub, donde también puede crear y revisar problemas y solicitudes de incorporación de cambios. Para más información, consulte nuestra guía para colaboradores.

Comentarios de .NET

.NET es un proyecto de código abierto. Seleccione un vínculo para proporcionar comentarios:

Abrir incidencia con la documentación

Proporcionar comentarios sobre el producto