

Conceptos básicos de las expresiones de consultas

Artículo • 18/12/2023

En este artículo se presentan los conceptos básicos relacionados con las expresiones de consulta en C#.

¿Qué es una consulta y qué hace?

Una *consulta* es un conjunto de instrucciones que describen qué datos se recuperan de uno o varios orígenes de datos determinados y qué forma y qué organización deben tener los datos devueltos. Una consulta es distinta de los resultados que genera.

Por lo general, los datos de origen se organizan lógicamente como una secuencia de elementos del mismo tipo. Por ejemplo, una tabla de base de datos SQL contiene una secuencia de filas. En un archivo XML, hay una "secuencia" de elementos XML (aunque los elementos XML se organizan jerárquicamente en una estructura de árbol). Una colección en memoria contiene una secuencia de objetos.

Desde el punto de vista de la aplicación, el tipo y la estructura específicos de los datos de origen originales no es importante. La aplicación siempre ve los datos de origen como una colección `IEnumerable<T>` o `IQueryable<T>`. Por ejemplo, en LINQ to XML, los datos de origen se hacen visibles como `IEnumerable<XElement>`.

Dada esta secuencia de origen, una consulta puede hacer una de estas tres cosas:

- Recuperar un subconjunto de los elementos para generar una nueva secuencia sin modificar los elementos individuales. Después, la consulta puede ordenar o agrupar la secuencia devuelta de varias maneras, como se muestra en el ejemplo siguiente (supongamos que `scores` es `int[]`):

C#

```
IEnumerable<int> highScoresQuery =  
    from score in scores  
    where score > 80  
    orderby score descending  
    select score;
```

- Recuperar una secuencia de elementos como en el ejemplo anterior, pero transformándolos en un nuevo tipo de objeto. Por ejemplo, una consulta puede recuperar solo los apellidos de ciertos registros de clientes de un origen de datos. También puede recuperar el registro completo y, luego, usarlo para construir otro tipo de objeto en memoria, o incluso datos XML, antes de generar la secuencia de resultado final. En el ejemplo siguiente muestra una proyección de `int` a `string`. Observe el nuevo tipo de `highScoresQuery`.

C#

```
IEnumerable<string> highScoresQuery2 =  
    from score in scores  
    where score > 80  
    orderby score descending  
    select $"The score is {score}";
```

- Recuperar un valor singleton sobre los datos de origen, por ejemplo:
 - El número de elementos que coinciden con una condición determinada.
 - El elemento que tiene el mayor o el menor valor.
 - El primer elemento que coincide con una condición, o bien la suma de determinados valores de un conjunto de elementos especificado. Por ejemplo, la consulta siguiente devuelve el número de resultados mayor que 80 de la matriz de enteros `scores`:

C#

```
var highScoreCount = (  
    from score in scores  
    where score > 80  
    select score  
) .Count();
```

En el ejemplo anterior, observe el uso de los paréntesis alrededor de la expresión de consulta antes de llamar al método [Enumerable.Count](#). También puede usar una nueva variable para almacenar el resultado concreto.

C#

```
IEnumerable<int> highScoresQuery3 =  
    from score in scores
```

```
where score > 80
select score;

var scoreCount = highScoresQuery3.Count();
```

En el ejemplo anterior, la consulta se ejecuta en la llamada a `Count`, ya que `Count` debe iterar los resultados para determinar el número de elementos devueltos por `highScoresQuery`.

¿Qué es una expresión de consulta?

Una *expresión de consulta* es una consulta que se expresa en sintaxis de consulta. Una expresión de consulta es una construcción de lenguaje de primera clase. Es igual que cualquier otra expresión y puede usarse en cualquier contexto en el que una expresión de C# sea válida. Una expresión de consulta consta de un conjunto de cláusulas escritas en una sintaxis declarativa similar a SQL o XQuery. Cada cláusula contiene una o más expresiones de C#, y estas expresiones pueden ser una expresión de consulta en sí mismas o bien contener una expresión de consulta.

Una expresión de consulta debe comenzar con una cláusula `from` y debe terminar con una cláusula `select` o `group`. Entre la primera cláusula `from` y la última cláusula `select` o `group`, puede contener una o varias de estas cláusulas opcionales: `where`, `orderby`, `join`, `let` e incluso otras cláusulas `from`. También puede usar la palabra clave `into` para que el resultado de una cláusula `join` o `group` actúe como el origen de más cláusulas de consulta en la misma expresión de consulta.

Variable de consulta

En LINQ, una variable de consulta es cualquier variable que almacene una *consulta* en lugar de los *resultados* de una consulta. Más concretamente, una variable de consulta es siempre un tipo enumerable que genera una secuencia de elementos cuando se itere en una instrucción `foreach` o en una llamada directa a su método `IEnumerator.MoveNext()`.

En el ejemplo de código siguiente se muestra una expresión de consulta simple con un origen de datos, una cláusula de filtrado, una cláusula de clasificación y ninguna transformación en los elementos de origen. La cláusula `select` finaliza la consulta.

C#

```
// Data source.
int[] scores = [90, 71, 82, 93, 75, 82];

// Query Expression.
IEnumerable<int> scoreQuery = //query variable
    from score in scores //required
    where score > 80 // optional
    orderby score descending // optional
    select score; //must end with select or group

// Execute the query to produce the results
foreach (var testScore in scoreQuery)
{
    Console.WriteLine(testScore);
}

// Output: 93 90 82 82
```

En el ejemplo anterior, `scoreQuery` es una *variable de consulta*, que a veces se conoce simplemente como una *consulta*. La variable de consulta no almacena datos de resultado reales, que se producen en el bucle `foreach`. Cuando se ejecuta la instrucción `foreach`, los resultados de la consulta no se devuelven a través de la variable de consulta `scoreQuery`, sino a través de la variable de iteración `testScore`. La variable `scoreQuery` se puede iterar en un segundo bucle `foreach`. Siempre y cuando ni esta ni el origen de datos se hayan modificado, producirá los mismos resultados.

Una variable de consulta puede almacenar una consulta expresada en sintaxis de consulta, en sintaxis de método o en una combinación de ambas. En los ejemplos siguientes, `queryMajorCities` y `queryMajorCities2` son variables de consulta:

C#

```
//Query syntax
IEnumerable<City> queryMajorCities =
    from city in cities
    where city.Population > 100000
    select city;

// Method-based syntax
IEnumerable<City> queryMajorCities2 = cities.Where(c => c.Population > 100000);
```

Por otro lado, en los dos ejemplos siguientes se muestran variables que no son de consulta, a pesar de que se inicialicen con una consulta. No son variables de consulta porque

almacenan resultados:

C#

```
var highestScore = (  
    from score in scores  
    select score  
).Max();  
  
// or split the expression  
IEnumerable<int> scoreQuery =  
    from score in scores  
    select score;  
  
var highScore = scoreQuery.Max();  
// the following returns the same result  
highScore = scores.Max();
```

C#

```
var largeCitiesList = (  
    from country in countries  
    from city in country.Cities  
    where city.Population > 10000  
    select city  
).ToList();  
  
// or split the expression  
IEnumerable<City> largeCitiesQuery =  
    from country in countries  
    from city in country.Cities  
    where city.Population > 10000  
    select city;  
var largeCitiesList2 = largeCitiesQuery.ToList();
```

Asignación implícita y explícita de tipos de variables de consulta

En esta documentación se suele proporcionar el tipo explícito de la variable de consulta para mostrar las relaciones de tipo entre la variable de consulta y la [cláusula select](#). Pero también se puede usar la palabra clave `var` para indicarle al compilador que infiera el tipo de una variable de consulta (u otra variable local) en tiempo de compilación. Por ejemplo, la consulta de ejemplo que se mostró anteriormente en este artículo también se puede expresar mediante la asignación implícita de tipos:

C#

```
var queryCities =  
    from city in cities  
    where city.Population > 100000  
    select city;
```

En el ejemplo anterior, el uso de `var` es opcional. `queryCities` es un `IEnumerable<City>` que indica si se escribe implícita o explícitamente.

Iniciar una expresión de consulta

Una expresión de consulta debe comenzar con una cláusula `from`, que especifica un origen de datos junto con una variable de rango. La variable de rango representa cada elemento sucesivo de la secuencia de origen a medida que esta se recorre. La variable de rango está fuertemente tipada en función del tipo de elementos del origen de datos. En el ejemplo siguiente, como `countries` es una matriz de objetos `Country`, la variable de rango también está tipada como `country`. Dado que la variable de rango está fuertemente tipada, se puede usar el operador punto para tener acceso a cualquier miembro disponible del tipo.

C#

```
IEnumerable<Country> countryAreaQuery =  
    from country in countries  
    where country.Area > 500000 //sq km  
    select country;
```

La variable de rango está en el ámbito hasta que se cierra la consulta con un punto y coma o con una cláusula de [continuación](#).

Una expresión de consulta puede contener varias cláusulas `from`. Use más cláusulas `from` cuando cada elemento de la secuencia de origen sea una colección en sí mismo o contenga una colección. Por ejemplo, supongamos que tiene una colección de objetos `Country`, cada uno de los cuales contiene una colección de objetos `city` denominados `cities`. Para consultar los objetos `city` de cada `country`, use dos cláusulas `from`, como se muestra aquí:

C#

```
IEnumerable<City> cityQuery =  
    from country in countries  
    from city in country.Cities  
    where city.Population > 10000  
    select city;
```

Para obtener más información, vea [from clause](#) (Cláusula from).

Finalizar una expresión de consulta

Una expresión de consulta debe finalizar con una cláusula `group` o `select`.

group (cláusula)

Use la cláusula `group` para generar una secuencia de grupos organizados por la clave que especifique. La clave puede ser cualquier tipo de datos. Por ejemplo, la siguiente consulta crea una secuencia de grupos que contienen uno o más objetos `Country` y cuya clave es un tipo `char` con un valor que es la primera letra del nombre de un país.

C#

```
var queryCountryGroups =  
    from country in countries  
    group country by country.Name[0];
```

Para obtener más información sobre la agrupación, vea [group clause](#) (Cláusula group).

select (cláusula)

Use la cláusula `select` para generar todos los demás tipos de secuencias. Una cláusula `select` simple solo genera una secuencia del mismo tipo de objetos que los objetos contenidos en el origen de datos. En este ejemplo, el origen de datos contiene objetos `Country`. La cláusula `orderby` simplemente ordena los elementos con un orden nuevo y la cláusula `select` genera una secuencia con los objetos `Country` reordenados.

C#

```
IEnumerable<Country> sortedQuery =  
    from country in countries
```

```
orderby country.Area  
select country;
```

La cláusula `select` puede usarse para transformar los datos de origen en secuencias de nuevos tipos. Esta transformación también se denomina *proyección*. En el ejemplo siguiente, la cláusula `select` *proyecta* una secuencia de tipos anónimos que solo contiene un subconjunto de los campos del elemento original. Los nuevos objetos se inicializan mediante un inicializador de objeto.

C#

```
var queryNameAndPop =  
    from country in countries  
    select new  
    {  
        Name = country.Name,  
        Pop = country.Population  
    };
```

Por lo tanto, en este ejemplo, se requiere el `var` porque la consulta genera un tipo anónimo.

Para obtener más información sobre todas las formas en que se puede usar una cláusula `select` para transformar datos de origen, vea [select clause](#) (Cláusula select).

Continuaciones con *into*

Puede usar la palabra clave `into` en una cláusula `select` o `group` para crear un identificador temporal que almacene una consulta. Use la cláusula `into` cuando deba realizar operaciones de consulta adicionales en una consulta después de una operación de agrupación o selección. En el siguiente ejemplo se agrupan los objetos `countries` según su población en intervalos de 10 millones. Una vez que se han creado estos grupos, más cláusulas filtran algunos grupos y, después, ordenan los grupos en orden ascendente. Para realizar esas operaciones adicionales, es necesaria la continuación representada por `countryGroup`.

C#

```
// percentileQuery is an IEnumerable<IGrouping<int, Country>>  
var percentileQuery =  
    from country in countries
```



```
let percentile = (int)country.Population / 10_000_000
group country by percentile into countryGroup
where countryGroup.Key >= 20
orderby countryGroup.Key
select countryGroup;

// grouping is an IGrouping<int, Country>
foreach (var grouping in percentileQuery)
{
    Console.WriteLine(grouping.Key);
    foreach (var country in grouping)
    {
        Console.WriteLine(country.Name + ":" + country.Population);
    }
}
```

Para obtener más información, vea [into](#).

Filtrar, ordenar y combinar

Entre la cláusula de inicio `from` y la cláusula de finalización `select` o `group`, todas las demás cláusulas (`where`, `join`, `orderby`, `from`, `let`) son opcionales. Cualquiera de las cláusulas opcionales puede usarse cero o varias veces en el cuerpo de una consulta.

where (cláusula)

Use la cláusula `where` para filtrar los elementos de los datos de origen en función de una o varias expresiones de predicado. La cláusula `where` del ejemplo siguiente tiene un predicado con dos condiciones.

C#

```
IEnumerable<City> queryCityPop =
    from city in cities
    where city.Population is < 200000 and > 100000
    select city;
```

Para obtener más información, vea [where \(Cláusula\)](#).

orderby (cláusula)

Use la cláusula `orderby` para ordenar los resultados en orden ascendente o descendente. También puede especificar criterios de ordenación secundaria. En el ejemplo siguiente se realiza una ordenación primaria de los objetos `country` mediante la propiedad `Area`. Después, se realiza una ordenación secundaria mediante la propiedad `Population`.

C#

```
IEnumerable<Country> querySortedCountries =  
    from country in countries  
    orderby country.Area, country.Population descending  
    select country;
```

La palabra clave `ascending` es opcional; es el criterio de ordenación predeterminado si no se especifica ningún orden. Para obtener más información, vea [orderby \(Cláusula\)](#).

join (cláusula)

Use la cláusula `join` para asociar o combinar elementos de un origen de datos con elementos de otro origen de datos en función de una comparación de igualdad entre las claves especificadas en cada elemento. En LINQ, las operaciones de combinación se realizan en secuencias de objetos cuyos elementos son de tipos diferentes. Después de combinar dos secuencias, debe usar una instrucción `select` o `group` para especificar qué elemento se va a almacenar en la secuencia de salida. También puede usar un tipo anónimo para combinar propiedades de cada conjunto de elementos asociados en un nuevo tipo para la secuencia de salida. En el ejemplo siguiente se asocian objetos `prod` cuya propiedad `category` coincide con una de las categorías de la matriz de cadenas `categories`. Los productos cuya propiedad `Category` no coincide con ninguna cadena de `categories` se filtran. La instrucción `select` proyecta un nuevo tipo cuyas propiedades se toman de `cat` y `prod`.

C#

```
var categoryQuery =  
    from cat in categories  
    join prod in products on cat equals prod.Category  
    select new  
    {  
        Category = cat,  
        Name = prod.Name  
    };
```

También puede realizar una combinación agrupada. Para ello, almacene los resultados de la operación `join` en una variable temporal mediante el uso de la palabra clave [into](#). Para obtener más información, vea [join \(Cláusula, Referencia de C#\)](#).

let (cláusula)

Use la cláusula `let` para almacenar el resultado de una expresión, como una llamada de método, en una nueva variable de rango. En el ejemplo siguiente, la variable de rango `firstName` almacena el primer elemento de la matriz de cadenas devuelta por `Split`.

C#

```
string[] names = ["Svetlana Omelchenko", "Claire O'Donnell", "Sven Mortensen",  
"Cesar Garcia"];  
IEnumerable<string> queryFirstNames =  
    from name in names  
    let firstName = name.Split(' ')[0]  
    select firstName;  
  
foreach (var s in queryFirstNames)  
{  
    Console.Write(s + " ");  
}  
  
//Output: Svetlana Claire Sven Cesar
```

Para obtener más información, vea [let \(Cláusula\)](#).

Subconsultas en una expresión de consulta

Una cláusula de consulta puede contener una expresión de consulta, en ocasiones denominada *subconsulta*. Cada subconsulta comienza con su propia cláusula `from` que no necesariamente hace referencia al mismo origen de datos de la primera cláusula `from`. Por ejemplo, la consulta siguiente muestra una expresión de consulta que se usa en la instrucción `select` para recuperar los resultados de una operación de agrupación.

C#

```
var queryGroupMax =  
    from student in students  
    group student by student.Year into studentGroup  
    select new  
    {
```

```
Level = studentGroup.Key,  
HighestScore = (  
    from student2 in studentGroup  
    select student2.ExamScores.Average()  
).Max()  
};
```

Para más información, consulte [Realizar una subconsulta en una operación de agrupación](#).

Consulte también

- [Palabras clave de consultas \(LINQ\)](#)
- [Información general sobre operadores de consulta estándar](#)


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](#)