

Introducción a las consultas LINQ (C#)

Artículo • 21/12/2023

Una *consulta* es una expresión que recupera datos de un origen de datos. Los distintos orígenes de datos tienen diferentes lenguajes de consulta nativos, por ejemplo SQL para bases de datos relacionales y XQuery para XML. Los programadores deben aprender un lenguaje de consultas nuevo para cada tipo de origen de datos o formato de datos que deben admitir. LINQ simplifica esta situación al ofrecer un modelo de lenguaje C# coherente para tipos de orígenes de datos y formatos. En una consulta LINQ, siempre se trabaja con objetos de C#. Use los mismos patrones de codificación básicos para consultar y transformar datos en documentos XML, bases de datos SQL, colecciones de .NET y cualquier otro formato cuando un proveedor LINQ esté disponible.

Las tres partes de una operación de consulta

Todas las operaciones de consulta LINQ constan de tres acciones distintas:

1. Obtener el origen de datos.
2. Crear la consulta.
3. Ejecutar la consulta.

En el siguiente ejemplo se muestra cómo se expresan las tres partes de una operación de consulta en código fuente. En el ejemplo se usa una matriz de enteros como origen de datos para su comodidad, aunque se aplican los mismos conceptos a otros orígenes de datos. En el resto de este tema se hará referencia a este artículo.

C#

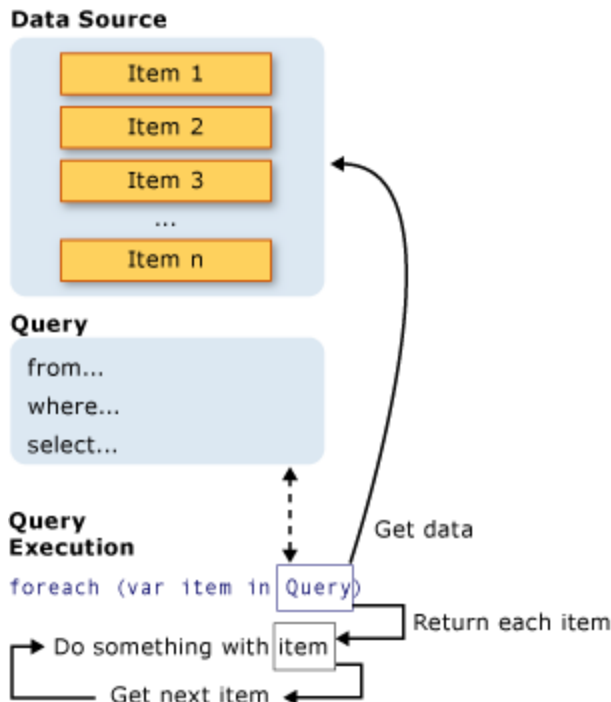
```
// The Three Parts of a LINQ Query:
// 1. Data source.
int[] numbers = [ 0, 1, 2, 3, 4, 5, 6 ];

// 2. Query creation.
// numQuery is an IEnumerable<int>
var numQuery =
    from num in numbers
    where (num % 2) == 0
    select num;

// 3. Query execution.
foreach (int num in numQuery)
```

```
{
    Console.WriteLine("{0,1} ", num);
}
```

En la siguiente ilustración se muestra toda la operación de consulta. En LINQ, la ejecución de la consulta es distinta de la propia consulta. En otras palabras, no se recupera ningún dato mediante la creación de una variable de consulta.



El origen de datos

El origen de datos del ejemplo anterior es una matriz, que admite la interfaz genérica `IEnumerable<T>`. Este hecho implica que se puede consultar con LINQ. Se ejecuta una consulta en una instrucción `foreach`, y `foreach` requiere `IEnumerable` o bien `IEnumerable<T>`. Los tipos compatibles con `IEnumerable<T>` o una interfaz derivada, como la interfaz genérica `IQueryable<T>`, se denominan *tipos consultables*.

Un tipo consultable no requiere ninguna modificación ni ningún tratamiento especial para actuar como origen de datos de LINQ. Si el origen de datos no está en la memoria como tipo consultable, el proveedor de LINQ debe representarlo como tal. Por ejemplo, LINQ to XML carga un documento XML en un tipo consultable `XElement`:

C#

```
// Create a data source from an XML document.
// using System.Xml.Linq;
```

```
XElement contacts = XElement.Load(@"c:\myContactList.xml");
```

Con [EntityFramework](#), se crea una asignación relacional de objetos entre las clases de C# y el esquema de la base de datos. Después, se escriben las consultas en los objetos y, en tiempo de ejecución, EntityFramework controla la comunicación con la base de datos. En el ejemplo siguiente, Customers representa una tabla específica en una base de datos, y el tipo del resultado de la consulta, [IQueryable<T>](#), se deriva de [IEnumerable<T>](#).

C#

```
Northwnd db = new Northwnd(@"c:\northwnd.mdf");

// Query for customers in London.
IQueryable<Customer> custQuery =
    from cust in db.Customers
    where cust.City == "London"
    select cust;
```

Para obtener más información sobre cómo crear tipos específicos de orígenes de datos, consulte la documentación de los distintos proveedores de LINQ. Aun así, la regla básica es sencilla: un origen de datos de LINQ es cualquier objeto que admita la interfaz genérica [IEnumerable<T>](#) o una interfaz que la haya heredado, normalmente [IQueryable<T>](#).

ⓘ Nota

Los tipos como **ArrayList**, que admiten la interfaz no genérica **IEnumerable**, también se pueden usar como origen de datos de LINQ. Para más información, consulte el [procedimiento para consultar un objeto ArrayList con LINQ \(C#\)](#).

Consulta

La consulta especifica la información que se debe recuperar de los orígenes de datos. Opcionalmente, una consulta también especifica cómo se debe ordenar, agrupar y dar forma a esa información antes de devolverse. Las consultas se almacenan en una variable de consulta y se inicializan con una expresión de consulta. Use [sintaxis de consulta de C#](#) para escribir consultas.

La consulta del ejemplo anterior devuelve todos los números pares de la matriz de enteros. La expresión de consulta contiene tres cláusulas: `from`, `where` y `select` (Si está

familiarizado con SQL, ha observado que la ordenación de las cláusulas se invierte del orden en SQL). La cláusula `from` especifica el origen de datos, la cláusula `where` aplica el filtro y la cláusula `select` especifica el tipo de los elementos devueltos. Todas las cláusulas de consulta se describen en detalle en esta sección. Por ahora, lo importante es que en LINQ la variable de consulta no efectúa ninguna acción y no devuelve ningún dato. Lo único que hace es almacenar la información necesaria para generar los resultados cuando se ejecuta la consulta en algún momento posterior. Para obtener más información sobre cómo se construyen las consultas, consulte [Información general sobre operadores de consulta estándar \(C#\)](#).

❗ Nota

Las consultas también se pueden expresar empleando una sintaxis de método. Para obtener más información, vea [Query Syntax and Method Syntax in LINQ](#) (Sintaxis de consulta y sintaxis de método en LINQ).

Ejecución de la consulta

Ejecución aplazada

La variable de consulta solo almacena los comandos de consulta. La ejecución real de la consulta se aplaza hasta que se procese una iteración en la variable de consulta en una instrucción `foreach`. Este concepto se conoce como *ejecución aplazada* y se muestra en el ejemplo siguiente:

C#

```
foreach (int num in numQuery)
{
    Console.WriteLine("{0,1} ", num);
}
```

La instrucción `foreach` es también donde se recuperan los resultados de la consulta. Por ejemplo, en la consulta anterior, la variable de iteración `num` contiene cada valor (de uno en uno) en la secuencia devuelta.

Dado que la propia variable de consulta nunca contiene los resultados de la consulta, puede ejecutarla repetidamente para recuperar los datos actualizados. Por ejemplo, se

puede tener una base de datos que esté siendo actualizada de forma continua por una aplicación independiente. En la aplicación, podría crear una consulta que recupere los datos más recientes y podría ejecutarla a intervalos para recuperar los resultados actualizados.

Forzar la ejecución inmediata

Las consultas que llevan a cabo funciones de agregación en un intervalo de elementos de origen primero deben recorrer en iteración dichos elementos. Ejemplos de estas consultas son `Count`, `Max`, `Average` y `First`. Estos métodos se ejecutan sin una instrucción `foreach` explícita porque la propia consulta debe usar `foreach` para devolver un resultado. Estas consultas devuelven un único valor, y no una colección `IEnumerable`. La consulta siguiente devuelve un recuento de los números pares de la matriz de origen:

C#

```
var evenNumQuery =  
    from num in numbers  
    where (num % 2) == 0  
    select num;  
  
int evenNumCount = evenNumQuery.Count();
```

Para forzar la ejecución inmediata de cualquier consulta y almacenar en caché los resultados correspondientes, puede llamar a los métodos [ToList](#) o [ToArray](#).

C#

```
List<int> numQuery2 =  
    (from num in numbers  
     where (num % 2) == 0  
     select num).ToList();  
  
// or like this:  
// numQuery3 is still an int[]  
  
var numQuery3 =  
    (from num in numbers  
     where (num % 2) == 0  
     select num).ToArray();
```

También puede forzar la ejecución colocando el bucle `foreach` justo después de la expresión de consulta, aunque, si se llama a `ToList` o a `ToArray`, también se almacenan en caché todos los datos de un objeto de colección.

Consulte también

- [Tutorial: Escribir consultas en C#](#)
- [foreach, in](#)
- [Palabras clave para consultas \(LINQ\)](#)


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](#)