Tema 7. Desarrollo de aplicaciones Web MVC (II)

ANEXO

Programación asincrónica con async y await

La utilización de la programación asincrónica **puede mejorar el rendimiento y la capacidad de respuesta de una aplicación Web**. La asincronía es esencial para actividades en las que se pueden producir bloqueos, como es el caso del acceso a Internet. Tener acceso a un recurso Web a veces ser lento y puede retrasar la realización de otras tareas. Si el acceso a un recurso Web queda bloqueado en un proceso sincrónico, todo el procesamiento de la aplicación Web que esté pendiente de procesamiento deberá esperar. Mientras que, **en un procesamiento asincrónico, la ejecución de la aplicación Web puede continuar con otra tarea que no dependa lógicamente del recurso Web bloqueado,** que puede ejecutarse hasta que finalice la ejecución del recurso Web que ha provocado el bloqueo. Normalmente, la tarea que puede ejecutarse, porque no depende desde un punto de vista lógico con la tarea bloqueada, suele estar relacionada con la construcción de la interfaz de usuario.

La programación asíncrona también tiene inconvenientes. La aplicación de las técnicas para escribir, depurar y mantener el código de las aplicaciones asincrónicas puede resultar complicado a nivel de programación. Sin embargo, actualmente el lenguaje C#, a partir de la versión 5, ha introducido un enfoque simplificado de la programación asincrónica, que aprovecha la compatibilidad asincrónica de .NET Framework, NET Core y Windows Runtime. La clave de esta simplificación consiste en que el código de la aplicación conserva una estructura lógica similar al código sincrónico y es el compilador quien realiza el trabajo más complejo. En efecto, el código se escribe como una secuencia de instrucciones, tal como se realiza habitualmente. De este modo, el código puede leerse como si cada instrucción se completase totalmente antes de comenzar la ejecución de la siguiente. El compilador es quién realiza una serie de transformaciones complejas que permiten gestionar la devolución de las llamadas y las continuaciones de ejecución que implica, normalmente, el uso de la programación asincrónica. De este modo, se pueden las características async y await del lenguaje C# para escribir programas asincrónicos. Como resultado, se obtienen las ventajas de la programación asincrónica y, al mismo tiempo, se usa un mecanismo de programación asincrónica fácil y ágil para especificar el código.

El mecanismo de programación asíncrona, que se describe en este anexo, se basa en el uso del modificador **async** y del operador **await** del lenguaje C#. Además, se utiliza la clase **Task** que representa el trabajo en curso, cuando se desea devolver un resultado desde un método asíncrono (async). Las aplicaciones Web basadas en ASP.NET Core utilizan este tipo de programación asíncrona para facilitar la realización de llamadas a métodos asíncronos como si fuesen síncronos y poder mejorar la respuesta en la ejecución de las aplicaciones Web interactivas.

El modificador **async** posibilita que un método o controlador de eventos pueda realizar llamadas asíncronas. Y el operador **await** permite realizar la llamada a un método de forma asíncrona. Solo se puede usar el operador await dentro de un método declarado como async. Hay que tener en cuenta que async y await no están relacionados con la creación de métodos asíncronos, sino con la llamada a esos métodos. Los métodos llamados con el operador async deben devolver su resultado como un objeto **Task<tipo_de_datos>**, o bien Task si no devuelven ningún resultado. Así, por ejemplo, si el método asíncrono debe devolver un entero, deberá estar declarado con el tipo de retorno Task<int>.

Por ejemplo, la forma de declarar un método como async, se realiza poniendo el modificador delante del nombre del método:

```
// Se declara el método como async, por lo que puede usarse await dentro
async Task<int> GetTaskOfTResultAsync()
{
    int hours = 0;

    . . .

    await GetTaskAsync();

    . . .

    // Se devuelve un entero
    return hours;
}
```

Y para realizar la llamada a un método declarado como async, se realiza poniendo el operador await delante del nombre del método, de la siguiente forma:

```
int i = await GetTaskOfTResultAsync();
```

O bien, de la siguiente forma:

```
Task<int> returnedTaskTResult = GetTaskOfTResultAsync();
int intResult = await returnedTaskTResult;
```

Por convención, habitualmente, los métodos que se llaman de forma asíncrona incluyen el sufijo *Async* en el nombre del método, aunque esto no es obligatorio.

Para declarar un método como async cuando no se devuelve ningún resultado, se haría:

```
async Task GetTaskAsync()
{
    . . .

    await DelayAsync(0);

    . . .
    // Este método no incluye instrucción return
}
```

Y la llamada al método se realizaría de la siguiente forma:

```
await GetTaskAsync();
```

Lo que ocurre cuando se llama a un método con el modificador async es que el control de ejecución pasa automáticamente a otra tarea, normalmente la tarea que procesa la interfaz de usuario, hasta que el método finaliza y el control continúa a partir la siguiente instrucción. De esta manera, se consigue que la aplicación continúe respondiendo mientras se realizan operaciones que conllevan tiempos de espera largos, como puede ser: la selección o modificación de información compleja, la descarga de un fichero, la realización de cálculos matemáticos complejos, etc. permitiendo realizar al usuario otras tareas al mismo tiempo.

El resultado es que el código asíncrono funciona mejor, porque Iniciará todas las tareas asincrónicas a la vez y esperará a una tarea solo cuando se necesiten los resultados. **Este comportamiento se parece más al funcionamiento que debería tener el código en las aplicaciones Web interactivas**, que puede realizar solicitudes a diferentes procesamientos (métodos, servicios Web, microservicios, etc.) y, después, combinar los resultados obtenidos en una única página Web resultante. Mediante la programación asíncrona se pueden realizar todas las solicitudes, empleando para ello la instrucción await y, después, esperar el resultado de todas esas tareas para componer, finalmente, la página Web de respuesta resultante del procesamiento.

Es necesario tener siempre presente que async y await no se refieren a la creación de métodos asíncronos, sino que se refieren a la llamada a esos métodos. El modificador async no significa “Este método es asíncrono”, sino que significa que el método quiere realizar llamadas a métodos asíncronos y sincronizarse con ellos. El operador await no significa “Espérate (bloquéate) hasta que termine la llamada al método asíncrono”, sino que significa que, si la tarea asíncrona no ha terminado, entonces vuelve al módulo llamador y marca el código posterior al await como código a ejecutarse una vez que se termine la tarea.

Puede obtenerse más información sobre programación asincrónica con async y await en las siguientes direcciones Web:

- https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/async/task-asynchronous-programming-model
- https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/async/