Async and Await in C#

Async and await keywords of C# were introduced in C# 5.0. They were designed to make it easier to write asynchronous code, which can run in the background while other code is executing.

The "async" keyword marks a method asynchronous, meaning it can be run in the background while another code executes. When you mark a method as async, you can use the "await" keyword to indicate that the method should wait for the result of an asynchronous operation before continuing.

Use of 'async' and 'await' in C#

Asynchronous programming is a programming technique that allows code to be executed concurrently without blocking the execution of the calling thread. In other words, asynchronous code can run in the background while other code is executing. In synchronous programming, each line of code is executed sequentially, and the program waits for each operation to complete before moving on to the next one. This can lead to performance problems, particularly in programs that need to perform long-running operations like I/O or network requests.

Asynchronous programming allows programs to perform these operations without blocking the calling thread. When an asynchronous operation is started, the program continues to execute other code while it waits for the operation to complete. The program is notified when the operation is complete and can continue with the following line of code.

Asynchronous programming can be implemented using various techniques, such as callbacks, events, and promises. In C#, the "async" and "await" keywords provide a convenient way to write asynchronous code that looks similar to synchronous code, making it easier to read and maintain.

We will get all the benefits of traditional Asynchronous programming with much less effort with the help of async and await keywords.

Suppose we are using two methods as, Method1 and Method2, respectively, and both methods are not dependent on each other, and Method1 takes a long time to complete its task. In Synchronous programming, it will execute the first Method1, wait for the completion of this method, and then execute Method2. Thus, it will be a time-intensive process even though both methods are not depending on each other.

We can run all the methods parallelly using simple thread programming, but it will block UI and wait to complete all the tasks. To come out of this problem, we have to write too many codes in traditional programming, but if we use the async and await keywords, we will get the solutions in much less code.

Also, we will see more examples, and if any third Method, as Method3 has a dependency on method1, it will wait for the completion of Method1 with the help of await keyword.

Async and await in C# are the code markers that mark code positions from where the control should resume after completing a task.

Let's start with practical examples for understanding the programming concept.

Code examples of C# async await

We are going to take a console application for our demonstration.

**Example 1**

In this example, we will take two methods that are not dependent on each other.

**Code sample**

class Program

{

static void Main(string[] args)

{

Method1();

Method2();

Console.ReadKey();

}

public static async Task Method1()

{

await Task.Run(() =>

{

for (int i = 0; i < 100; i++)

{

Console.WriteLine(" Method 1");

// Do something

Task.Delay(100).Wait();

}

});

}

public static void Method2()

{

for (int i = 0; i < 25; i++)

{

Console.WriteLine(" Method 2");

// Do something

Task.Delay(100).Wait();

}

}

}

C#

Copy

In the code above, Method1 and Method2 are not dependent on each other, and we call from the Main method.

Here, we can see Method1 and Method2 are not waiting for each other.

**Output**

**A screenshot of a computer

Description automatically generated**

Regarding the second example, suppose we have Method3, which depends on Method1.

**Example 2**

In this example, Method1 returns the total length as an integer value, and we pass a parameter as a length in Method3, which comes from Method1.

Here, we have to use await keyword before passing a parameter in Method3, and for it, we have to use the async keyword from the calling method.

If we use C# 7 or less, we cannot use the async keyword in the Main method for the console Application because it will give the error below.



We will create a new method called callMethod, and in this method, we will call all our Methods Method1, Method2, and Method3, respectively.

**Code sample** **C# 7**

class Program

{

static void Main(string[] args)

{

callMethod();

Console.ReadKey();

}

public static async void callMethod()

{

Task<int> task = Method1();

Method2();

int count = await task;

Method3(count);

}

public static async Task<int> Method1()

{

int count = 0;

await Task.Run(() =>

{

for (int i = 0; i < 100; i++)

{

Console.WriteLine(" Method 1");

count += 1;

}

});

return count;

}

public static void Method2()

{

for (int i = 0; i < 25; i++)

{

Console.WriteLine(" Method 2");

}

}

public static void Method3(int count)

{

Console.WriteLine("Total count is " + count);

}

}

C#

Copy

**Code sample C# 9**

class Program

{

static async Task Main(string[] args)

{

await callMethod();

Console.ReadKey();

}

public static async Task callMethod()

{

Method2();

var count = await Method1();

Method3(count);

}

public static async Task<int> Method1()

{

int count = 0;

await Task.Run(() =>

{

for (int i = 0; i < 100; i++)

{

Console.WriteLine(" Method 1");

count += 1;

}

});

return count;

}

public static void Method2()

{

for (int i = 0; i < 25; i++)

{

Console.WriteLine(" Method 2");

}

}

public static void Method3(int count)

{

Console.WriteLine("Total count is " + count);

}

}

C#

Copy

In the code above, Method3 requires one parameter, which is the return type of Method1. Here, the await keyword is vital in waiting for Method1 task completion.

**Output**

**A computer screen with white text

Description automatically generated**

**Real-time example**

Some support APIs from the .NET Framework 4.5, and the Windows runtime contains methods that support async programming.

We can use all of these in the real-time project with the help of async and await keywords for the faster execution of the task.

Some APIs that contain async methods are HttpClient, SyndicationClient, StorageFile, StreamWriter, StreamReader, XmlReader, MediaCapture, BitmapEncoder, BitmapDecoder, etc.

In this example, we will read all the characters from a large text file asynchronously and get the total length of all the characters.

**Sample code**

class Program

{ ke

static void Main()

{

Task task = new Task(CallMethod);

task.Start();

task.Wait();

Console.ReadLine();

}

static async void CallMethod()

{

string filePath = "E:\\sampleFile.txt";

Task<int> task = ReadFile(filePath);

Console.WriteLine(" Other Work 1");

Console.WriteLine(" Other Work 2");

Console.WriteLine(" Other Work 3");

int length = await task;

Console.WriteLine(" Total length: " + length);

Console.WriteLine(" After work 1");

Console.WriteLine(" After work 2");

}

static async Task<int> ReadFile(string file)

{

int length = 0;

Console.WriteLine(" File reading is stating");

using (StreamReader reader = new StreamReader(file))

{

// Reads all characters from the current position to the end of the stream asynchronously

// and returns them as one string.

string s = await reader.ReadToEndAsync();

length = s.Length;

}

Console.WriteLine(" File reading is completed");

return length;

}

}

C#

Copy

In the code given above, we are calling a ReadFile method to read the contents of a text file and get the length of the total characters present in the text file.

In our sampleText.txt, the file contains too many characters, so It will take a long time to read all the characters.

Here, we are using async programming to read all the contents from the file, so it will not wait to get a return value from this method and execute the other lines of code. , However, it still has to wait for the line of code given below because we are using await keywords, and we will use the return value for the line of code below.

int length = await task;

Console.WriteLine(" Total length: " + length);

C#

Copy

Subsequently, other lines of code will be executed sequentially.

Console.WriteLine(" After work 1");

Console.WriteLine(" After work 2");

C#

Copy

**Output**

**A black screen with white text

Description automatically generated**

Here, we must understand important points: if we are not using await keyword, then the method works synchronously. So the compiler will show the warning to us, but it will not show any error.

We can use async and await keywords in C# to implement async programming in this easy way,

If you're new to Async programming, here is a detailed tutorial [**on Asynchronous programming in C#**](https://www.c-sharpcorner.com/UploadFile/84c85b/asynchronous-programming-with-C-Sharp-5-0/)**.**