[목 차]

비동기 호출

C# 5.0에는 async와 await 예약어가 새롭게 추가되었다. 이 예약어를 이용하면 비동기 호출을 마치 동기 방식처럼 호출하는 코드를 작성할 수 있다.

1. 동기 방식

```
using System;
using System.IO;
using System.Text;

class Program
{
    static void Main(string[] args)
    {
        using (FileStream fs = new FileStream(@"C:₩windows₩system32₩drivers₩etc₩HOSTS", FileMode.Open, FileAccess.Read, FileShare.ReadWrite))
        {
            byte[] buf = new byte[fs.Length];
            fs.Read(buf, 0, buf.Length);
            // 스레드가 Read 메서드를 완료한 후 파일의 내용을 화면에 출력하는 코드를 순차적으로 실행
            string txt = Encoding.UTF8.GetString(buf);
            Console.WriteLine(txt);
        }
    }
}
```

2. 기존 비동기 방식

BeginRead 메서드를 호출했을 때는 Read 동작 이후의 코드를 별도로 분리해 Completed와 같은 형식의 메서드에 담아 처리해야 하는 불편함이 있다.

```
using System;
using System.IO;
using System.Text;
class FileState{
    public byte[] Buffer;
    public FileStream File;
}
class Program{
    static void Main(string[] args)
    {
        AsyncRead();
    }

    private static void AsyncRead()
    {
        FileStream fs = new FileStream(@"C:₩windows₩system32₩drivers₩etc₩HOSTS", FileMode.Open, FileAccess.Read, FileShare.ReadWrite);

        FileState state = new FileState();
        state.Buffer = new byte[fs.Length];
        state.File = fs;

        fs.BeginRead(state.Buffer, 0, state.Buffer.Length, readCompleted, state);
        // Read가 완료된 후의 코드를 readCompleted로 넘겨서 처리

        Console.ReadLine();
        fs.Close();
    }

    // 읽기 작업이 완료되면 스레드 풀의 자유 스레드가 readCompleted 메서드를 실행
    static void readCompleted(IAsyncResult ar)
    {
        FileState state = ar.AsyncState as FileState;
        state.File.EndRead(ar);
        string txt = Encoding.UTF8.GetString(state.Buffer);
        Console.WriteLine(txt);
    }
}
```

3. C#5.0 에서 지원하는 비동기 방식

동기를 비동기로 바꾸는 것은 Read 호출 이후의 코드를 BeginRead에 전달하는 것으로 해결된다는 사실을 알 수 있다. 이 작업을 컴파일러가 알아서 해줄 수는 없을까? 바로 그런 목적으로 탄생한 것이 async/await 예약어이다.

```
using System;
using System.IO;
using System.Text;
using System.Threading;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            AwaitRead();
            Console.ReadLine();
        }

        private static async void AwaitRead()
        {
            using (FileStream fs = new FileStream(@"C:₩windows₩system32₩drivers₩etc₩HOSTS", FileMode.Open,
FileAccess.Read, FileShare.ReadWrite))
            {
                byte[] buf = new byte[fs.Length];
                Console.WriteLine("Before ReadAsync: " + Thread.CurrentThread.ManagedThreadId);
                await fs.ReadAsync(buf, 0, buf.Length);
                Console.WriteLine("After ReadAsync: " + Thread.CurrentThread.ManagedThreadId);

                // 아래의 두 라인은 C# 컴파일러가 분리해, ReadAsync 비동기 호출이 완료된 후 호출
                string txt = Encoding.UTF8.GetString(buf);
                Console.WriteLine(txt);
            }
        }
    }
}
```

* FileStream 타입은 await 비동기 호출에 사용되는 ReadAsync 메서드를 새롭게 제공한다.

Async 류의 비동기 호출에 await 예약어가 함께 쓰이면 C# 컴파일러는 이를 인지하고 그 이후의 코드를 묶어서 ReadAsync 의 비동기 호출이 끝난 후에 실행되도록 코드를 변경해서 컴파일 한다. 그 덕분에 비동기 호출을 동기 호출처럼 코드를 작성할 수 있다.

* 코드가 호출되기 전 스레드 ID와 호출된 후의 스레드 ID가 다르게 출력되는 것을 확인할 수 있다.

4. 닷넷 4.5 BCL에 추가된 Async 메서드

Async / await 의 도입으로 비동기 호출 코드가 매우 간결해졌다.

기존의 BCL 라이브러리에 제공되던 복잡한 비동기 처리에 async / await 호출이 가능한 메서드를 추가했다.

예제1) 동기 코드

```
using System;
using System.Net;
namespace ConsoleApplication1{
    class Program    {
        static void Main(string[] args)        {
            WebClient wc = new WebClient();
            string text = wc.DownloadString("http://www.microsoft.com");
            Console.WriteLine(text);
        }
    }
}
```

예제2) 비동기 코드

```
using System;
using System.Net;
namespace ConsoleApplication1{
    class Program    {
        static void Main(string[] args)        {
            WebClient wc = new WebClient();

            // DownloadStringAsync 동작이 완료됐을 때 호출할 이벤트 등록
            wc.DownloadStringCompleted += wc_DownloadStringCompleted;

            // DownloadString의 비동기 메서드
            wc.DownloadStringAsync(new Uri("http://www.microsoft.com"));
            Console.ReadLine();
        }

        static void wc_DownloadStringCompleted(object sender, DownloadStringCompletedEventArgs e)        {
            Console.WriteLine(e.Result); // e.Result == HTML 텍스트
        }
    }
}
```

예제3) async/ await를 사용한 비동기 코드

```
using System;
using System.Net;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            AwaitDownloadString();

            Console.ReadLine();
        }

        private static async void AwaitDownloadString()
        {
            WebClient wc = new WebClient();
            string text = await wc.DownloadStringTaskAsync("http://www.microsoft.com");
            Console.WriteLine(text);
        }
    }
}
```

5. Task, Task<TResult> 타입

지금까지 await 와 함께 사용된 메서드를 보면 한 가지 공통점이 있다.

```
FileStream 타입
  public Task<int> ReadAsync(byte[] buffer, int offset, int count);
WebClient 타입
   public Task<string> DownloadStringTaskAsync(string address);
```

즉, Async 메서드의 반환값이 모두 Task<TResult> 유형이라는 점이다.

Task 타입은 반환 값이 없는 경우에 사용되고, Task<TResult> 타입은 TResult 형식 매개변수로 지정된 반환 값이 있는 경우로 구분된다. 그리고 await 비동기 처리와는 별도로 원래부터 병렬 처리 라이브러리에 속한 타입이다.

따라서 await 없이 Task 타입을 단독으로 사용하는 것이 가능하다.

```
using System;
using System.Threading;
using System.Threading.Tasks;
namespace ConsoleApplication1{
    class Program    {
        static void Main(string[] args)          {
            // 기존의 QueueUserWorkItem으로 별도의 스레드에서 작업을 수행
            ThreadPool.QueueUserWorkItem(
                (obj) =>
                {
                    Console.WriteLine("process workitem");
                }, null);
            // .NET 4.0의 Task 타입을 이용해 별도의 스레드에서 작업을 수행
            Task task1 = new Task(
                () =>
                {
                    Console.WriteLine("process taskitem");
                });
            task1.Start();
            Task task2 = new Task(
                (obj) =>
                {
                    Console.WriteLine("process taskitem(obj)");
                }, null);
            task2.Start();
            Console.ReadLine();
        }
    }
}
```

예제) Task / Task<TResult> 사용 예

```
using System;
using System.Threading;
using System.Threading.Tasks;

class Program
{
    static void Main(string[] args)
    {
        Task taskSleep = new Task(() => { Thread.Sleep(5000); });
        taskSleep.Start();
        taskSleep.Wait(); // Task의 작업이 완료될 때까지 현재 스레드를 대기한다.

        //StartNew 메서드를 사용하면 Action 델리게이트를 전달하자 마자 곧바로 작업을 시작하게 할 수 있다.
        Task.Factory.StartNew(() => { Console.WriteLine("process taskitem"); });

        Task.Factory.StartNew((obj) => { Console.WriteLine("process taskitem(obj)"); }, null);

        //실행이 완료된 후 원한다면 반환값까지 처리 할 수 있도록 개선했다.
        Task<int> task = new Task<int>(
            () =>
            {
                Random rand = new Random((int)DateTime.Now.Ticks);
                return rand.Next();
            }
        );

        task.Start();
        task.Wait();

        Console.WriteLine("무작위 숫자 값: " + task.Result);

        Task<int> taskReturn = Task.Factory.StartNew<int>(() => 1);
        taskReturn.Wait();
        Console.WriteLine(taskReturn.Result);
    }
}
```

6. Async 메서드가 아닌 경우의 비동기 처리

C#의 await 예약어가 Task, Task<TResult> 타입을 반환하는 메서드를 대상으로 비동기 처리를 자동화 했다는 점은 또 다른 활용 사례를 낳는다.

즉, Async 처리가 적용되지 않은 메서드에 대해 Task를 반환하는 부가 메서드를 만드는 것으로 await 비동기 처리를 할 수 있다.

예제1) ReadAllText 메서드를 비동기로 처리

```
using System;
using System.IO;
namespace ConsoleApplication1{
    class Program    {
        public delegate string ReadAllTextDelegate(string path);

        static void Main(string[] args)
        {
            //ReadAllText는 비동기 버전의 메서드를 제공하지 않는다.
            //그래서 이 작업을 비동기로 처리하려면 별도의 스레드를 이용하거나 델리게이트의 BeginInvoke
            //로 처리해야 했다. 그 결과 매우 복잡해 진다.
            //string text = File.ReadAllText(@"C:₩windows₩system32₩drivers₩etc₩HOSTS");
            //Console.WriteLine(text);

            string filePath = @"C:₩windows₩system32₩drivers₩etc₩HOSTS";

            // 델리게이트를 이용한 비동기 처리
            ReadAllTextDelegate func = File.ReadAllText;
            func.BeginInvoke(filePath, actionCompleted, func);

            Console.ReadLine(); // 비동기 스레드가 완료될 때까지 대기하는 용도
        }

        static void actionCompleted(IAsyncResult ar)          {
            ReadAllTextDelegate func = ar.AsyncState as ReadAllTextDelegate;
            string fileText = func.EndInvoke(ar);

            // 파일의 내용을 화면에 출력
            Console.WriteLine(fileText);
        }
    }
}
```

예제2) 위 코드를 Task<TResult> 활용 코드로 변경

아래 방법을 이용하면 BCL뿐만 아니라 사용자가 만드는 메서드도 비동기로 변환이 가능하다.

```
using System;
using System.IO;
using System.Threading.Tasks;

class Program
{
    static void Main(string[] args)
    {
        AwaitFileRead(@"C:₩windows₩system32₩drivers₩etc₩HOSTS");
        Console.ReadLine();
    }

    private static async void AwaitFileRead(string filePath)
    {
        string fileText = await ReadAllTextAsync(filePath);
        Console.WriteLine(fileText);
    }

    static Task<string> ReadAllTextAsync(string filePath)
    {
        return Task.Factory.StartNew(() =>
        {
            return File.ReadAllText(filePath);
        });
    }
}
```

7. 비동기 호출의 병렬 처리

await 와 Task 의 조합으로 할 수 있는 것 중 하나는 병렬로 비동기 호출을 할 수 있다.

예를 들어 작업이 완료되는 데 각각 5초와 3초가 걸리는 두 개의 메서드가 있다고 가정해 보자.

예제1) 8초가 걸리는 작업

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int result3 = Method3();
            int result5 = Method5();

            Console.WriteLine(result3 + result5);
        }

        private static int Method3()
        {
            Thread.Sleep(3000); // 3초가 걸리는 작업을 대신해서 Sleep 처리
            return 3;
        }

        private static int Method5()
        {
            Thread.Sleep(5000); // 5초가 걸리는 작업을 대신해서 Sleep 처리
            return 5;
        }
    }
}
```

예제2) Thread를 이용한 처리

```
using System;
using System.Collections.Generic;
using System.Threading;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Dictionary<string, int> dict = new Dictionary<string, int>();

            Thread t3 = new Thread((result) =>
            {
                Thread.Sleep(3000);
                dict.Add("t3Result", 3);
            });

            Thread t5 = new Thread((result) =>
            {
                Thread.Sleep(5000);
                dict.Add("t5Result", 5);
            });

            t3.Start(dict);
            t5.Start(dict);

            t3.Join(); // 3초짜리 작업이 완료되기를 대기
            t5.Join(); // 5초짜리 작업도 완료되기를 대기
                       // 약 5초 후에 모든 결괏값을 얻어 처리 가능

            Console.WriteLine(dict["t3Result"] + dict["t5Result"]);
        }
    }
}
```

예제3) Task<TResult> 타입으로 구현

2개의 작업을 병렬로 처리하지만 모든 작업이 완료될때까지 대기

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            // Task를 이용해 병렬로 처리
            var task3 = Method3Async();
            var task5 = Method5Async();

            // task3 작업과 task5 작업이 완료될 때까지 현재 스레드를 대기
            Task.WaitAll(task3, task5);

            Console.WriteLine(task3.Result + task5.Result);
        }

        private static Task<int> Method3Async()
        {
            return Task.Factory.StartNew(() =>
            {
                Thread.Sleep(3000);
                return 3;
            });
        }

        private static Task<int> Method5Async()
        {
            return Task.Factory.StartNew(() =>
            {
                Thread.Sleep(5000);
                return 5;
            });
        }
    }
}
```

예제4) Task<TResult> 와 await의 도움을 받아 비동기 호출로 처리

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
namespace ConsoleApplication1{
    class Program    {
        static void Main(string[] args)
        {
            // await을 이용해 병렬로 비동기 호출: 5초 소요
            DoAsyncTask();
            Console.ReadLine();
        }

        private static async void DoAsyncTask()          {
            var task3 = Method3Async();
            var task5 = Method5Async();

            await Task.WhenAll(task3, task5);

            Console.WriteLine(task3.Result + task5.Result);
        }

        private static Task<int> Method3Async()          {
            return Task.Factory.StartNew(() =>
            {
                Thread.Sleep(3000);
                return 3;
            });
        }

        private static Task<int> Method5Async()          {
            return Task.Factory.StartNew(() =>
            {
                Thread.Sleep(5000);
                return 5;
            });
        }
    }
}
```

8. Task의 7가지 사용법 정리

```
using System;
using System.Threading;
using System.Threading.Tasks;
namespace ConsoleApplication1{
    class Program    {
        //1. 직접 호출
        static void fun1()          {
            Task.Factory.StartNew(
                () =>
                {
                Console.WriteLine("Hello Task Library!");

                });
        }

        //2. Action 사용
        static void fun2()          {
            Task task = new Task(new Action(PrintMessage));
            task.Start();
        }

        static void PrintMessage()          {
            Console.WriteLine("Hello Task Library!");
        }

        //3. 델리게이트 사용
        static void fun3()          {
            Task task = new Task(
                delegate { PrintMessage();
                });
            task.Start();
        }

        //4. 람다 사용
        static void fun4()          {
            Task task = new Task(
                () => PrintMessage());
            task.Start();
        }
```

```
        //5. 람다와 익명함수
        static void fun5()          {
            Task task = new Task(
                () => { PrintMessage(); });
            task.Start();
        }

        //6. 닷넷 4.5 이상(Task.Run)
        static async Task fun6Async()           {
            await Task.Run(() => PrintMessage());
        }


        //7. 닷넷 4.5 이상(Task.FromResult)
        public async Task fun7Async()           {
            int res = await Task.FromResult<int>(GetSum(4, 5));
        }

        static int GetSum(int a, int b)
        {
            return a + b;
        }

        static void Main(string[] args)
        {
            // await을 이용해 병렬로 비동기 호출: 5초 소요
            fun1();
        }
    }
}
```