What is the difference between Task.Run() and Task.Factory.StartNew()

Asked 3 years ago Active 1 month ago Viewed 70k times

137

38

I have Method :

```
private static void Method()
{
    Console.WriteLine("Method() started");

    for (var i = 0; i < 20; i++)
    {
        Console.WriteLine("Method() Counter = " + i);
        Thread.Sleep(500);
    }

    Console.WriteLine("Method() finished");
}
```

And I want to start this method in a new Task. I can start new task like this

```
var task = Task.Factory.StartNew(new Action(Method));
```

or this

```
var task = Task.Run(new Action(Method));
```

But is there any difference between `Task.Run()` and `Task.Factory.StartNew()`. Both of them are using ThreadPool and start Method() immediately after creating instance of the Task. When we should use first variant and when second?

c# multithreading task-parallel-library

edited Jul 12 '17 at 19:53
Christos
45.3k 8 50 81

asked Jul 17 '16 at 16:34
Sergiy Lichenko
782 2 7 10

6 Actually, StartNew does not have to use the ThreadPool, see the blog I linked to in my answer. The problem is `StartNew` by default uses `TaskScheduler.Current` which may be the thread pool but also could be the UI thread. – Scott Chamberlain Jul 17 '16 at 16:51

2 Possible duplicate of Regarding usage of Task.Start() , Task.Run() and Task.Factory.StartNew() – Ahmed Abdelhameed Oct 26 '17 at 4:39

6 Answers

148

The second method, `Task.Run` , has been introduced in a later version of the .NET framework (in .NET 4.5).

However, the first method, `Task.Factory.StartNew` , gives you the opportunity to define a lot of useful things about the thread you want to create, while `Task.Run` doesn't provide this.

For instance, lets say that you want to create a long running task thread. If a thread of the thread pool is going to be used for this task, then this could be considered an abuse of the thread pool.

One thing you could do in order to avoid this would be to run the task in a separate thread. A newly created thread *that would be dedicated to this task and would be destroyed once your task would have been completed.* You *cannot* achieve this with the `Task.Run` , while you can do so with the `Task.Factory.StartNew` , like below:

```
Task.Factory.StartNew(..., TaskCreationOptions.LongRunning);
```

As it is stated here:

> So, in the .NET Framework 4.5 Developer Preview, we've introduced the new Task.Run method. **This in no way obsoletes** Task.Factory.StartNew, **but rather should simply be thought of as a quick way to use** Task.Factory.StartNew **without needing to specify a bunch of parameters. It's a shortcut.** In fact, Task.Run is actually implemented in terms of the same logic used for Task.Factory.StartNew, just passing in some default parameters. When you pass an Action to Task.Run:

```
Task.Run(someAction);
```

> that's exactly equivalent to:

```
Task.Factory.StartNew(someAction,
    CancellationToken.None, TaskCreationOptions.DenyChildAttach, TaskScheduler.Default);
```

edited Jan 13 '18 at 7:26

TheGeneral
42.2k 8 46 76

answered Jul 17 '16 at 16:38

Christos
45.3k 8 50 81

4 I have a piece of code where the statemente `that's exactly equivalent to` does not hold. – Emaborsa Sep 11 '17 at 10:05

6 @Emaborsa I would appreciate If you could post this piece of code and elaborate your argument. Thanks in advance ! – Christos Sep 11 '17 at 10:10

3 @Emaborsa You could create a gist, gist.github.com, and share it. However, except from sharing this gist, please specify how did you get to the outcome that the phrase `tha's exactly equivalent to` does not hold. Thanks in advance. It would be nice to explain with comment on your code. Thanks :) – Christos Sep 12 '17 at 7:17

7 It's also worth mentioning that Task.Run unwrap nested task by default. I recommend to read this article about major differences: blogs.msdn.microsoft.com/pfxteam/2011/10/24/… – Pawel Maga Nov 7 '17 at 14:59

1 @The0bserver nope, it is `TaskScheduler.Default` . Please have a look here referencesource.microsoft.com/#mscorlib/system/threading/Tasks/…. – Christos Jan 21 at 18:13

25

See this blog article that describes the difference. Basically doing:

```
Task.Run(A)
```

Is the same as doing:

```
Task.Factory.StartNew(A, CancellationToken.None, TaskCreationOptions.DenyChildAttach,
TaskScheduler.Default);
```

edited Nov 28 '17 at 9:37

Bugs
4,190 9 27 37

answered Jul 17 '16 at 16:48

Scott Chamberlain
101k 25 203 336

21

The `Task.Run` got introduced in newer .NET framework version and it is recommended.

> Starting with the .NET Framework 4.5, the Task.Run method is the recommended way to launch a compute-bound task. Use the StartNew method only when you require fine-grained control for a long-running, compute-bound task.

The `Task.Factory.StartNew` has more options, the `Task.Run` is a shorthand:

> The Run method provides a set of overloads that make it easy to start a task by using default values. It is a lightweight alternative to the StartNew overloads.

And by shorthand I mean a technical shortcut:

```
public static Task Run(Action action)
{
    return Task.InternalStartNew(null, action, null, default(CancellationToken),
TaskScheduler.Default,
        TaskCreationOptions.DenyChildAttach, InternalTaskOptions.None, ref stackMark);
}
```

edited Nov 16 '18 at 9:45

Rekshino
3,364 2 10 36

answered Jul 17 '16 at 16:36

Zein Makki
24.2k 4 33 49

19

According to this post by Stephen Cleary, Task.Factory.StartNew() is dangerous:

> I see a lot of code on blogs and in SO questions that use Task.Factory.StartNew to spin up work on a background thread. Stephen Toub has an excellent blog article that explains why Task.Run is better than Task.Factory.StartNew, but I think a lot of people just haven't read it (or don't understand it). So, I've taken the same arguments, added some more forceful language, and we'll see how this goes. :) StartNew does offer many more options than Task.Run, but it is quite dangerous, as we'll see. You should prefer Task.Run over Task.Factory.StartNew in async code.

Here are the actual reasons:

> 1. Does not understand async delegates. This is actually the same as point 1 in the reasons why you would want to use StartNew. The problem is that when you pass an async delegate to StartNew, it's natural to assume that the returned task represents that delegate. However, since StartNew does not understand async delegates, what that task actually represents is just the beginning of that delegate. This is one of the first pitfalls that coders encounter when using StartNew in async code.
> 2. Confusing default scheduler. OK, trick question time: in the code below, what thread does the method "A" run on?

```
Task.Factory.StartNew(A);

private static void A() { }
```

> Well, you know it's a trick question, eh? If you answered "a thread pool thread", I'm sorry, but that's not correct. "A" will run on whatever TaskScheduler is currently executing!

So that means it could potentially run on the UI thread if an operation completes and it marshals back to the UI thread due to a continuation as Stephen Cleary explains more fully in his post.

In my case, I was trying to run tasks in the background when loading a datagrid for a view while also displaying a busy animation. The busy animation didn't display when using `Task.Factory.StartNew()` but the animation displayed properly when I switched to `Task.Run()`.

For details, please see https://blog.stephencleary.com/2013/08/startnew-is-dangerous.html

edited Nov 15 '18 at 21:36

answered Jun 18 '18 at 15:36

user8128167

3,127 5 38 54

7

People already mentioned that

```
Task.Run(A);
```

Is equivalent to

```
Task.Factory.StartNew(A, CancellationToken.None, TaskCreationOptions.DenyChildAttach,
TaskScheduler.Default);
```

But no one mentioned that

```
Task.Factory.StartNew(A);
```

Is equivalent to:

```
Task.Factory.StartNew(A, CancellationToken.None, TaskCreationOptions.None,
TaskScheduler.Current);
```

As you can see two parameters are different for `Task.Run` and `Task.Factory.StartNew` :

1. `TaskCreationOptions` - `Task.Run` uses `TaskCreationOptions.DenyChildAttach` which means that children tasks can not be attached to the parent, consider this:

   ```
   var parentTask = Task.Run(() =>
   {
       var childTask = new Task(() =>
       {
           Thread.Sleep(10000);
           Console.WriteLine("Child task finished.");
       }, TaskCreationOptions.AttachedToParent);
       childTask.Start();

       Console.WriteLine("Parent task finished.");
   });

   parentTask.Wait();
   Console.WriteLine("Main thread finished.");
   ```

   When we invoke `parentTask.Wait()` , `childTask` will not be awaited, even though we specified `TaskCreationOptions.AttachedToParent` for it, this is because `TaskCreationOptions.DenyChildAttach` forbids children to attach to it. If you run the same code with `Task.Factory.StartNew` instead of `Task.Run` , `parentTask.Wait()` will wait for `childTask` because `Task.Factory.StartNew` uses `TaskCreationOptions.None`

2. `TaskScheduler` - `Task.Run` uses `TaskScheduler.Default` which means that the default task scheduler (the one that runs tasks on Thread Pool) will always be used to run tasks. `Task.Factory.StartNew` on the other hand uses `TaskScheduler.Current` which means scheduler of the current thread, it might be `TaskScheduler.Default` but not always. In fact when developing `Winforms` or `WPF` applications it is required to update UI from the current thread, to do this people use `TaskScheduler.FromCurrentSynchronizationContext()` task scheduler, if you unintentionally create another long running task inside task that used `TaskScheduler.FromCurrentSynchronizationContext()` scheduler the UI will be frozen. A more detailed explanation of this can be found here

So generally if you are not using nested children task and always want your tasks to be executed on Thread Pool it is better to use `Task.Run` , unless you have some more complex scenarios.

edited Jun 17 at 11:54 answered May 2 at 9:13

Mykhailo Seniutovych

736 1 8 21

-11

In my application which calls two services, I compared both Task.Run and **Task.Factory.StartNew**. I found that in my case both of them work fine. However, the second one is faster.

answered Dec 29 '17 at 18:52

Devendra Rusia

1 1