MICROSOFT ARCHITECT
By Joydip Kanjilal, Columnist, InfoWorld
MAR 2, 2020 3:00 AM PST

How to schedule jobs using Quartz.NET in ASP.NET Core

Take advantage of a Quartz.NET hosted service to schedule background jobs in your ASP.NET Core application

When working on web applications, you will often need to execute certain tasks in the background. In some cases, these will be tasks that should be executed at predefined intervals of time.

Quartz.NET is an open source .NET port of the popular Java job scheduling framework. It has been in use for a long time and provides excellent support for working with Cron expressions. You can learn more about Quartz.NET from an earlier post here.

[Also on InfoWorld: When to use an abstract class vs. interface in C#]

This article presents a discussion of how we can work with Quartz.NET in ASP.NET Core to schedule background jobs.

To work with the code examples provided in this article, you should have Visual Studio 2019 installed in your system. If you don't already have a copy, you can download Visual Studio 2019 here.

Create an ASP.NET Core API project

First off, let's create an ASP.NET Core project in Visual Studio. Assuming Visual Studio 2019 is installed in your system, follow the steps outlined below to create a new ASP.NET Core project in Visual Studio.

1. Launch the Visual Studio IDE.
2. Click on "Create new project."
3. In the "Create new project" window, select "ASP.NET Core Web Application" from the list of templates displayed.
4. Click Next.
5. In the "Configure your new project" window shown next, specify the name and location for the new project.
6. Click Create.
7. In the "Create New ASP.NET Core Web Application" window, select .NET Core as the runtime and ASP.NET Core 2.2 (or later) from the drop-down list at the top. I'll be using ASP.NET Core 3.0 here.
8. Select "API" as the project template to create a new ASP.NET Core API application.
9. Ensure that the check boxes "Enable Docker Support" and "Configure for HTTPS" are unchecked as we won't be using those features here.
10. Ensure that Authentication is set as "No Authentication" as we won't be using authentication either.
11. Click Create.

This will create a new ASP.NET Core API project in Visual Studio. Select the Controllers solution folder in the Solution Explorer window and click “Add -> Controller...” to create a new controller named DefaultController.

Next, to work with Quartz, you should install the Quartz package from NuGet. You can do this either via the NuGet package manager inside the Visual Studio 2019 IDE, or by executing the following command at the NuGet package manager console:


```
Install-Package Quartz
```

Quartz.NET jobs, triggers, and schedulers

The three main concepts in Quartz.NET are jobs, triggers, and schedulers. A job contains the code to execute a task or a job to be performed. A job is represented by a class that implements the IJob interface. A trigger is used to specify the schedule and other details of a job. You can take advantage of a trigger to specify how the job should be executed. The scheduler is the component that is responsible for polling and executing jobs based on pre-defined schedules.

Create a scheduler using Quartz.NET

It should be noted that you can have multiple schedulers in an application. However, we'll use just one scheduler here for the sake of simplicity. The following code snippet illustrates how you can create a scheduler instance.

```
var scheduler = StdSchedulerFactory.GetDefaultScheduler().GetAwaiter().GetResult();
```

Once the scheduler has been created you can use the following code in the ConfigureServices method of the Startup.cs file to add the scheduler instance as a singleton service.

```
services.AddSingleton(scheduler);
```

[Also on InfoWorld: How to pass multiple parameters to Web API controller methods]

Start and stop a scheduler using Quartz.NET

To start and stop the scheduler we'll take advantage of a hosting service. To do this, you need to create a class that implements the IHostingService interface as shown in the code snippet given below.

```
public class CustomQuartzHostedService : IHostedService
{
        private readonly IScheduler _scheduler;
        public CustomQuartzHostedService(IScheduler scheduler)
        {
            _scheduler = scheduler;
        }
        public async Task StartAsync(CancellationToken cancellationToken)
        {
            await _scheduler?.Start(cancellationToken);
        }
        public async Task StopAsync(CancellationToken cancellationToken)
        {
            await _scheduler?.Shutdown(cancellationToken);
        }
 }
```

Note that you should register the hosted service in the services collection in the ConfigureServices method using the code snippet given below.

```
services.AddHostedService<QuartzHostedService>();
```

Here is the updated ConfigureServices method for your reference:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();
    var scheduler =
    StdSchedulerFactory.GetDefaultScheduler().GetAwaiter().GetResult();
    services.AddSingleton(scheduler);
    services.AddHostedService<QuartzHostedService>();
}
```

Create a job using Quartz.NET

As I said earlier, a job is a class that implements the IJob interface and contains the Execute() method. The Execute() method accepts an instance of type IJobExecutionContext.

The following code snippet illustrates a job class that contains an asynchronous Execute() method as well. This method contains the code that corresponds to the task that your job should perform.

```
[DisallowConcurrentExecution]
public class NotificationJob : IJob
    {
        private readonly ILogger<NotificationJob> _logger;
        public NotificationJob(ILogger<NotificationJob> logger)
        {
            _logger = logger;
        }
        public Task Execute(IJobExecutionContext context)
        {
            _logger.LogInformation("Hello world!");
            return Task.CompletedTask;
        }
    }
```

Create a job factory using Quartz.NET

A job factory is a class that inherits the IJobFactory interface and implements the NewJob() and ReturnJob() methods. The following code snippet can be used to create a factory class that can create and return a job instance.

```
public class CustomQuartzJobFactory : IJobFactory
    {
        private readonly IServiceProvider _serviceProvider;
        public CustomQuartzJobFactory(IServiceProvider serviceProvider)
        {
            _serviceProvider = serviceProvider;
        }
        public IJob NewJob(TriggerFiredBundle triggerFiredBundle,
        IScheduler scheduler)
        {
            var jobDetail = triggerFiredBundle.JobDetail;
            return (IJob)_serviceProvider.GetService(jobDetail.JobType);
        }
        public void ReturnJob(IJob job) { }
    }
```

Note that this implementation doesn't take advantage of job pooling. If you want to use job pooling, you should change the NewJob() method and then implement the ReturnJob() method.

[Also on InfoWorld: TypeScript vs. JavaScript: Understand the differences]

Create a JobMetadata class to store your job metadata

We'll use a custom class to store the metadata related to a job, i.e., the job Id, name, etc. The following class represents the job metadata class.

```
public class JobMetadata
    {
        public Guid JobId { get; set; }
        public Type JobType { get; }
        public string JobName { get; }
        public string CronExpression { get; }
        public JobMetadata(Guid Id, Type jobType, string jobName,
        string cronExpression)
        {
            JobId = Id;
            JobType = jobType;
            JobName = jobName;
            CronExpression = cronExpression;
        }
    }
```

Create a hosted service to start and stop the Quartz.NET scheduler

Next, we'll need to implement a hosted service. A hosted service is a class that implements the IHostedService interface and starts the Quartz scheduler. The following code listing illustrates a custom hosted service class.

```
public class CustomQuartzHostedService : IHostedService
    {
        private readonly ISchedulerFactory schedulerFactory;
        private readonly IJobFactory jobFactory;
        private readonly JobMetadata jobMetadata;
        public CustomQuartzHostedService(ISchedulerFactory
            schedulerFactory,
            JobMetadata jobMetadata,
            IJobFactory jobFactory)
        {
            this.schedulerFactory = schedulerFactory;
            this.jobMetadata = jobMetadata;
            this.jobFactory = jobFactory;
        }
        public IScheduler Scheduler { get; set; }
        public async Task StartAsync(CancellationToken cancellationToken)
        {
            Scheduler = await schedulerFactory.GetScheduler();
            Scheduler.JobFactory = jobFactory;
            var job = CreateJob(jobMetadata);
            var trigger = CreateTrigger(jobMetadata);
            await Scheduler.ScheduleJob(job, trigger, cancellationToken);
            await Scheduler.Start(cancellationToken);
        }
        public async Task StopAsync(CancellationToken cancellationToken)
        {
            await Scheduler?.Shutdown(cancellationToken);
        }
        private ITrigger CreateTrigger(JobMetadata jobMetadata)
        {
            return TriggerBuilder.Create()
            .WithIdentity(jobMetadata.JobId.ToString())
            .WithCronSchedule(jobMetadata.CronExpression)
            .WithDescription($"{jobMetadata.JobName}")
            .Build();
        }
        private IJobDetail CreateJob(JobMetadata jobMetadata)
        {
            return JobBuilder
            .Create(jobMetadata.JobType)
            .WithIdentity(jobMetadata.JobId.ToString())
            .WithDescription($"{jobMetadata.JobName}")
            .Build();
        }
    }
```

The following code snippet shows the complete code of the ConfigureServices method of the Startup class.

```
public void ConfigureServices(IServiceCollection services)
{
services.AddControllers();
services.AddSingleton<IJobFactory, CustomQuartzJobFactory>();
services.AddSingleton<ISchedulerFactory, StdSchedulerFactory>();
services.AddSingleton<NotificationJob>();
services.AddSingleton(new JobMetadata(Guid.NewGuid(), typeof(NotificationJob),"Notificat
services.AddHostedService<CustomQuartzHostedService>();
}
```


And that's all you have to do! When you execute the application you'll observe that the Execute() method of the NotificationJob class runs once every 10 seconds.

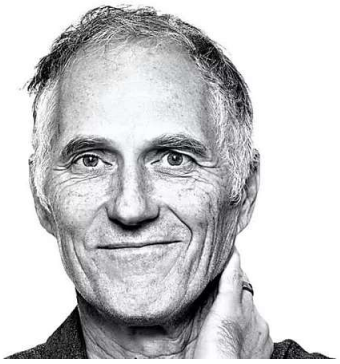
Quartz.NET is a good choice for implementing schedulers in your applications. You can take advantage of the persistence feature in Quartz.NET to store your jobs in a database such as SQL Server, PostgreSQL, or SQLite as well.

Joydip Kanjilal is a Microsoft MVP in ASP.Net, as well as a speaker and author of several books and articles. He has more than 20 years of experience in IT including more than 16 years in Microsoft .Net and related technologies.

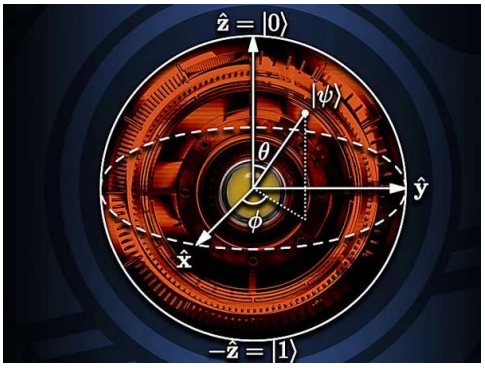
Follow

YOU MAY ALSO LIKE

Recommended by

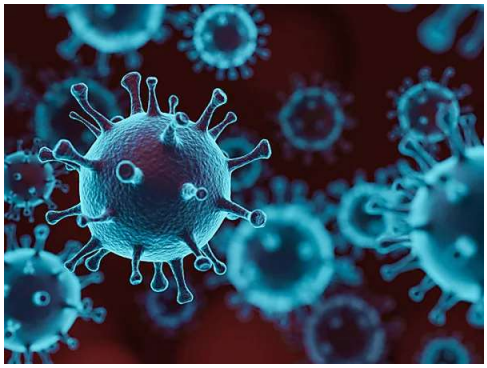


Tim O'Reilly: the golden age of the programmer is over



Amazon Braket: Get started with quantum computing

7 best practices for remote development teams



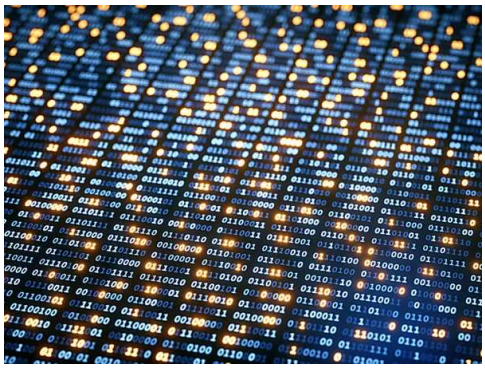
The COVID pandemic's lasting impact on cloud usage

Make the most of R colors and palettes

Oracle Database 21c review: The old RDBMS is new again

6 reasons to switch to managed Kubernetes

Red Hat OpenShift ramps up security and



What is a computational storage drive? Much-

When to use Task.WaitAll vs. Task.WhenAll in .NET

3 enterprise AI success stories