UNITED STATES

MICROSOFT ARCHITECT

By Joydip Kanjilal, Columnist, InfoWorld
AUG 5, 2019 3:00 AM PDT

How to use conditional middleware in ASP.Net Core

Take advantage of advanced operations such as conditional branching when working with middleware in ASP.Net Core

ASP.Net Core is an open source, cross-platform, extensible, lean, and modular framework from Microsoft that can be used for building high-performance web applications. Middleware components can be used in the ASP.Net Core request pipeline to customize the way requests and responses are handled.

ASP.Net Core middleware components can also be used to inspect, route, or modify the request and response messages that flow through the pipeline. This article presents a discussion of how we can perform some advanced operations with middleware in ASP.Net Core.

[Microsoft .Net 5 unites the .Net Framework and .Net Core: Find out what the merger of .Net Standard and .Net Core means for developers. | Learn how to get the most from .Net Framework and .Net Core from InfoWorld's Microsoft Architect blog. | Keep up with hot topics in programming with InfoWorld's App Dev Report newsletter.]

Create an ASP.Net Core MVC project

First off, let's create an ASP.Net Core project in Visual Studio. Assuming that Visual Studio 2017 or 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, specify the name and location for the new project.
6. Click Create.
7. In the "Create New ASP.Net Core Web Application" shown next, select .Net Core as the runtime and ASP.Net Core 2.2 (or later) from the drop-down list at the top.
8. Select "Web Application (Model-View-Controller)" as the project template to create a new ASP.Net Core 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,

Following these steps should create a new ASP.Net Core project in Visual Studio. We'll use this project in the subsequent sections of this article.

The Use, Run, and Map methods in ASP.Net Core

The Use, Map, and Run methods are used to configure the HTTP pipeline in ASP.Net

Core. Here’s a glimpse at each of these methods and their purpose.

- Use — this method will execute the delegate and then move to the next step in the pipeline. The Use method can also be used to short-circuit the pipeline.
- Run — this method will execute a delegate and return the result.
- Map — this method will execute a delegate conditionally and return the result.

Register middleware in ASP.Net Core

A middleware component in ASP.Net Core is registered in the Configure method of the Startup class. The Use* extension methods are used to register middleware. Here is the syntax for registering a middleware component.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseMyCustomMiddleware<MyCustomMiddleware>();
}
```

It should be noted that middleware components are executed in the order in which they are registered.

The Invoke method in ASP.Net Core

Each middleware component contains an Invoke method. This method accepts a reference to an instance of HttpContext as an argument. A middleware component can perform operations before and after the next middleware component is called. Here is an example of a typical Invoke method:

```
public async Task Invoke(HttpContext context)
{
    // Write code here that will be executed before the
    // next middleware is called
        await _next.Invoke(context); // call next middleware
    // Write code here that will be executed after the
    //next middleware is called
}
```

Branching the HTTP pipeline in ASP.Net Core

The Map extension methods, i.e. Map and MapWhen, are used for branching the pipeline. While Map is used to branch based on a given request path, MapWhen is used to branch based on the result of a given predicate.

The following code snippet illustrates how the Map method can be used for branching the request pipeline.

```
public class Startup
{
    private static void MapRequestA(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            await context.Response.WriteAsync("This is MapRequestA");
        });
    }
    private static void MapRequestB(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            await context.Response.WriteAsync("This is MapRequestB");
        });
    }
    private static void MapRequestC(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            await context.Response.WriteAsync("This is MapRequestC");
        });
    }
    public void Configure(IApplicationBuilder app)
    {
        app.Map("/mapRequestPathA", MapRequestA);
        app.Map("/mapRequestPathB", MapRequestB);
        app.Map("/mapRequestPathB", MapRequestC);
        app.Run(async context =>
        {
            await context.Response.WriteAsync("Hello World!");
        });
    }
   //Other methods
}
```

The MapWhen method accepts two parameters:

- A Func<HttpContext, bool> predicate
- A delegate action

You can use the following code snippet in the Configure method of the Startup class to disallow content type "text/html".

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
       {
           if (env.IsDevelopment())
           {
               app.UseDeveloperExceptionPage();
           }
           app.MapWhen(context => context.Request.ContentType.Equals
           ("text/xml", StringComparison.InvariantCultureIgnoreCase),
           (IApplicationBuilder applicationBuilder) =>
           {
               applicationBuilder.Run(async context =>
               {
                   await Task.FromResult(context.Response.StatusCode = StatusCodes.Stat
               });
           });
           app.UseMvc();
       }
```

The UseWhen method in ASP.Net Core

The UseWhen method can be used to execute middleware conditionally. The following code snippet illustrates how the UseWhen method can be used to execute a middleware component if the request path starts with "/api".

```
app.UseWhen(context => context.Request.Path.StartsWithSegments("/api"), applicationBuil
{
    applicationBuilder.UseCustomMiddleware();
});
```

Note that unlike MapWhen, the UseWhen method continues to execute the later middleware irrespective of whether the UseWhen predicate was true or false. Let’s understand this with an example. Consider the following piece of code:

```
app.UseMiddlewareA();
app.UseWhen(context => context.Request.Path.StartsWithSegments("/api"), applicationBuil
{
    applicationBuilder.UseMiddlewareB();
});
app.UseMiddlewareC();
```

If there is no short-circuiting of the middleware, middleware A and C will always be executed. Middleware B will be executed only if the request path starts with "/api".

In ASP.Net Core there is a chain of middleware components in the request processing pipeline. All requests and responses flow through this pipeline. When a new request arrives, these middleware components can either process the request or pass the request to the next component in the pipeline. For more complex request processing, we can use the Map and MapWhen methods to branch the pipeline and UseWhen to execute middleware conditionally.

—

Learn more about middleware and request and response handling in ASP.Net Core:

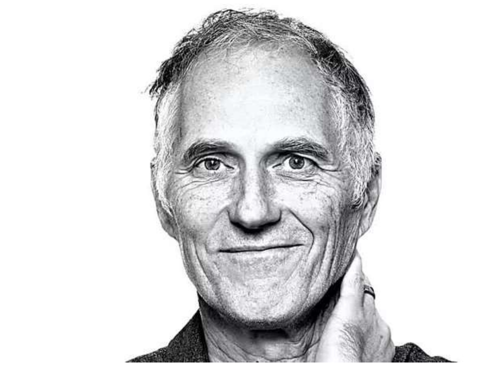
- How to build custom middleware in ASP.Net Core
- How to improve the performance of ASP.Net Core
- How to use response caching middlware in ASP.Net Core
- How to implement health checks in ASP.Net Core
- How to implement global exception handling in ASP.Net Core
- How to use action filters in ASP.Net Core MVC

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.

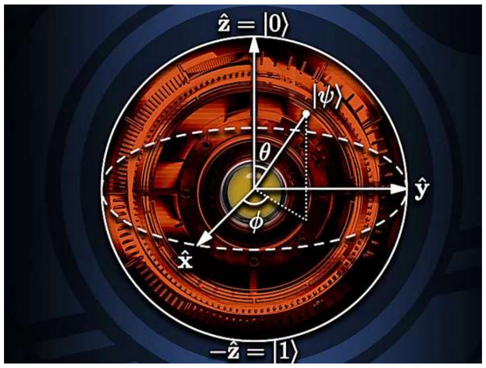
Follow

YOU MAY ALSO LIKE

Recommended by



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

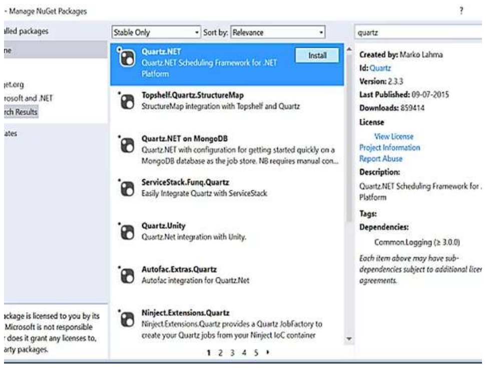


Amazon Braket: Get started with quantum computing

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

2 common cloud computing predictions for 2021 are wrong

JDK 17: The new features in Java 17



- Manage NuGet Packages ?

alled packages | Stable Only | Sort by: Relevance | quartz

ne
et.org
rosoft and .NET
rch Results
ates

Quartz.NET Install
Quartz.NET Scheduling Framework for .NET Platform

Topshelf.Quartz.StructureMap
StructureMap integration with Topshelf and Quartz

Quartz.NET on MongoDB
Quartz.NET with configuration for getting started quickly on a MongoDB database as the job store. NB requires manual con...

ServiceStack.Funq.Quartz
Easily Integrate Quartz with ServiceStack

Quartz.Unity
Quartz.Net integration with Unity.

Autofac.Extras.Quartz
Autofac integration for Quartz.Net

Ninject.Extensions.Quartz
Ninject.Extensions.Quartz provides a Quartz JobFactory to create your Quartz jobs from your Ninject IoC container

1 2 3 4 5 ›

Created by: Marko Lahma
Id: Quartz
Version: 2.3.3
Last Published: 09-07-2015
Downloads: 859414
License
View License
Project Information
Report Abuse
Description:
Quartz.NET Scheduling Framework for . Platform
Tags:
Dependencies:
Common.Logging (≥ 3.0.0)
Each item above may have sub-dependencies subject to additional licer agreements.

ckage is licensed to you by its Microsoft is not responsible does it grant any licenses to, arty packages.

How to work with Quartz.Net in C#

3 enterprise AI success stories

3 cloud architecture mistakes we all make, but

Red Hat OpenShift ramps up security and

The 24 highest paying developer roles in 2020

What’s new in Kubernetes 1.20