ASP.NET Core 面试题汇总

ASP.NET Core 面试题汇总

目录

1.如何在 controller 中注入 service?

在 config services 方法中配置这个 service。

在 controller 的构造函数中，添加这个依赖注入。

---->详解

2.ASP.NET Core 比 ASP.NET 更具优势的地方是什么?

跨平台，ASP.NET Core 可以运行在 Windows 、Linux 和 MAC 系统上;

对框架本安装没有依赖，所有依赖都跟程序本身在一起;

ASP.NET Core 处理请求的效率更高，进而可以处理更多的请求;

ASP.NET Core 具有更多的安装配置方法。

3.asp.net core 主要的特性有哪些?

依赖注入。

日志系统架构。

引入了一个跨平台的网络服务器，kestrel。

可以没有 iis, apache 和 nginx 就可以单独运行。

可以使用命令行创建应用。

使用 AppSettings.json 来配置工程。

使用 start up 来注册服务。

更好的支持异步编程。

支持 web socket 和 signal IR。

对于跨网站的请求的预防和保护机制。

----> 详解

4.ASP.NET Core Filter 如何支持依赖注入?

可以通过全局注册，支持依赖注入

通过 TypeFilter(typeof(Filter)) 标记在方法，标记在控制器

通过 ServiceType(typeof(Filter))标记在方法，标记在控制器，必须要注册 Filter 这类；

TypeFilter 和 ServiceType 的本质是实现了一个 IFilterFactory 接口；

---->详解

5. ASP.NET Core 中有哪些异常处理的方案?

1）.继承 Controller，重写 OnActionExecuted

默认都会继承一个 Controller 类，重写 OnActionExecuted，添加上异常处理即可。一般情况下我们会新建一个 BaseController，让所有 Controller 继承 BaseController。代码如下

2）.使用 ActionFilterAttribute。

```
public class BaseController : Controller
{
    public override void OnActionExecuted(ActionExecutedContext context)
    {
        var exception = context.Exception;
        if (exception != null)
        {
            context.ExceptionHandled = true;
            context.Result = new ContentResult
            {
                Content = $"BaseController错误 : { exception.Message }"
            };
        }
        base.OnActionExecuted(context);
    }
}
```

ActionFilterAttribute 是一个特性，本身实现了 IActionFilter 及 IResultFilter，所以不管是 action 里抛错，还是 view 里抛错，理论上都可以捕获。我们新建一个 ExceptionActionFilterAttribute，重写 OnActionExecuted 及 OnResultExecuted，添加上异常处理，完整代码如下:

```
public class ExceptionActionFilterAttribute:ActionFilterAttribute
{
    public override void OnActionExecuted(ActionExecutedContext context)
    {
        var exception = context.Exception;
        if (exception != null)
        {
            context.ExceptionHandled = true;
            context.Result = new ContentResult
            {
                Content = $"错误 : { exception.Message }"
            };
        }
        base.OnActionExecuted(context);
    }

    public override void OnResultExecuted(ResultExecutedContext context)
    {
        var exception = context.Exception;
        if (exception != null)
        {
            context.ExceptionHandled = true;
          context.HttpContext.Response.WriteAsync($"错误 : {exception.Message}");
        }
        base.OnResultExecuted(context);
    }
}
```

使用方式有两种，

在 controller 里打上 [TypeFilter(typeof(ExceptionActionFilter)] 标签;

在 Startup 里以 filter 方式全局注入。

```
services.AddControllersWithViews(options =>
{
    options.Filters.Add<ExceptionActionFilterAttribute>();
})
```

3）.使用 IExceptionFilter

我们知道, Asp.Net Core 提供了 5 类 filter, IExceptionFilter 是其中之一，顾名

思义，这就是用来处理异常的。Asp.net Core 中 ExceptionFilterAttribute 已经实现了 IExceptionFilter，所以我们只需继承 ExceptionFilterAttribute，重写其中方法即可。同样新建 CustomExceptionFilterAttribute 继承 ExceptionFilterAttribute，重写 OnException ，添加异常处理，完整代码如下：

```
public class CustomExceptionFilterAttribute :
ExceptionFilterAttribute
{
    public override void OnException(ExceptionContext context)
    {
        context.ExceptionHandled = true;
        context.HttpContext.Response
       .WriteAsync($"CustomExceptionFilterAttribute错误
       :{context.Exception.Message}");
        base.OnException(context);
    }
}
```

4）.使用 ExceptionHandler.

在 startup 里，vs 新建的项目会默认加上.

```
if (env.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}
else
{
    app.UseExceptionHandler("/Home/Error");
}
```

5）.自定义 Middleare 处理

通过 middleware 全局处理。

```
public class ErrorHandlingMiddleware
{
    private readonly RequestDelegate next;

    public ErrorHandlingMiddleware(RequestDelegate next)
    {
         this.next = next;
    }

    public async Task Invoke(HttpContext context)
    {
         try
         {
            await next(context);
         }
         catch (System.Exception ex)
         {
            //处理异常
         }
    }
}
```

---->详解

6.介绍 ASP.NET Core 中服务的生命周期?

ASP.NET Core 支持依赖注入软件设计模式，它允许在不同的组件中注入我们的服务，并且控制服务的初始化。有些服务可以在短时间内初始化，并且只能在某个特别的组件，以及请求中才能用到；而还有一些服务，仅仅只用初始化一次，就可以在整个应用程序中使用。

Singleton --单例模式：

只有一个服务的实例被创建，这个实例，存储在内存中，可以在整个应用程序中使用。我们可以对一些初始化代价比较大的服务，使用 Singleton 模式。在代码中可以这样：

```
services.AddSingleton<IProductService, ProductService>();
```

Scoped --作用域

这种模式下，将会为每一个请求，都创建一个服务的实例。所有同一个请求中的中间件、MVC 控制器，等等，都会得到一个相同的服务实例。Entity Framework Context 就是一个 Scoped 服务最好的参考例子。我们可以通过使用 AddScoped 方法来使用 Scoped 模式：

```
services.AddScoped<IProductService, ProductService>();
```

Transient --短暂的、瞬变的

Transient 模式下，服务每次被请求的时候，都会创建一个服务实例，这种模式特别适合轻量级、无状态的服务。我们可以使用 AddTransient 方法，来注入服务：

```
services.AddTransient<IProductService, ProductService>();
```

---->详解

7.什么是依赖注入？

依赖注入是一个过程，就是当一个类需要调用另一个类来完成某项任务的时候，在调用类里面不要去 new 被调用的类的对象，而是通过注入的方式来获取这样一个对象。具体的实现就是在调用类里面有一个被调用类的接口，然后通过调用接口的函数来完成任务。比如 A 调用 B，而 B 实现了接口 C，那么在 A 里面用 C 定义一个变量 D，这个变量的实例不在 A 里面创建，而是通过 A 的上下文来获取。这样做的好处就是将类 A 和 B 分开了，他们之间靠接口 C 来联系，从而实现对接口编程。

---->详解

8.依赖注入有哪几种方式？

setter 注入：

就是在类 A 里面定义一个 C 接口的属性 D，在 A 的上下文通过 B 实例化一个对象，然后将这个对象赋值给属性 D。主要就是 set 与 get

构造函数注入：

就是在创建 A 的对象的时候，通过参数将 B 的对象传入到 A 中。

还有常用的注入方式就是工厂模式的应用了，这些都可以将 B 的实例化放到 A 外面，从而让 A 和 B 没有关系。还有一个接口注入，就是在客户类（A）的接口中有一个服务类(B)的属性。在实例化了这个接口的子类后，对这个属性赋值，这和 setter 注入一样。

接口注入：

相比构造函数注入和属性注入，接口注入显得有些复杂，使用也不常见。具体思路是先定义一个接口，包含一个设置依赖的方法。然后依赖类，继承并实现这个接口。

---->详解

9.控制反转是什么?

控制反转（Inversion of Control，缩写为 IoC），是面向对象编程中的一种设计原则，可以用来减低计算机代码之间的耦合度。其中最常见的方式叫做依赖注入（Dependency Injection，简称 DI），还有一种方式叫"依赖查找"（Dependency Lookup）。

通过控制反转，对象在被创建的时候，由一个调控系统内所有对象的外界实体将其所依赖的对象的引用传递给它。也可以说，依赖被注入到对象中。

---->详解

10.依赖注入有哪些著名的框架?

Unity、autofac、http://spring.net、MEF、Injection、Asp.Net Core 的 ServiceCollection。

11.什么是 dotnet core 的 startup class?

Startup class 是 dot net core 应用的入口。所有的 dot net core 应用必须有这个 class。这个类用来配置应用。这个类的调用是在 program main 函数里面进行配置的。类的名字可以自己定义。

---->详解

12.startup class 的 configure 方法有什么作用?

这个方法来定义整个应用如何响应 HTTP 请求。它有几个比较重要的参数，application builder，Hosting environment, logo factory， 在这里我们可以配置一些中间件用来处理路径，验证和 session 等等。

---->详解

13.什么是.NET Core 中间件（Middleware）？

中间件是组装到应用程序管道中以处理请求和响应的软件。每个组件：

选择是否将请求传递给管道中的下一个组件。

可以在调用管道中的下一个组件之前和之后执行工作。

请求委托（Request delegates）用于构建请求管道，处理每个 HTTP 请求。

请求委托使用 Run，Map 和 Use 扩展方法进行配置。单独的请求委托可以以内联匿名方法（称为内联中间件）指定，或者可以在可重用的类中定义它。这些可重用的类和内联匿名方法是中间件或中间件组件。请求流程中的每个中间件组件都负责调用流水线中的下一个组件，如果适当，则负责链接短路。

14.ASP.NET Core 中间件的使用场景有哪些?

份验证，Session 存储，日志记录等。其实我们的 Asp.net core 项目中本身已经包含了很多个中间件。比如 身份认证中间件 UseAuthorization()等系列

---->详解

15.列举官方常用的中间件?

异常/错误处理 当应用在开发环境中运行时：开发人员异常页中间件(UseDeveloperExceptionPage) 报告应用运行时错误。数据库错误页中间件报告数据库运行时错误。当应用在生产环境中运行时：异常处理程序中间件(UseExceptionHandler) 捕获以下中间件中引发的异常。HTTP 严格传输安全协议 (HSTS) 中间件 (UseHsts) 添加 Strict-Transport-Security 标头。

HTTPS 重定向中间件 (UseHttpsRedirection) 将 HTTP 请求重定向到 HTTPS。

静态文件中间件 (UseStaticFiles) 返回静态文件，并简化进一步请求处理。

Cookie 策略中间件 (UseCookiePolicy) 使应用符合欧盟一般数据保护条例 (GDPR) 规定。

用于路由请求的路由中间件 (UseRouting)。

身份验证中间件（UseAuthentication）尝试对用户进行身份验证，然后才会允许用户访问安全资源。

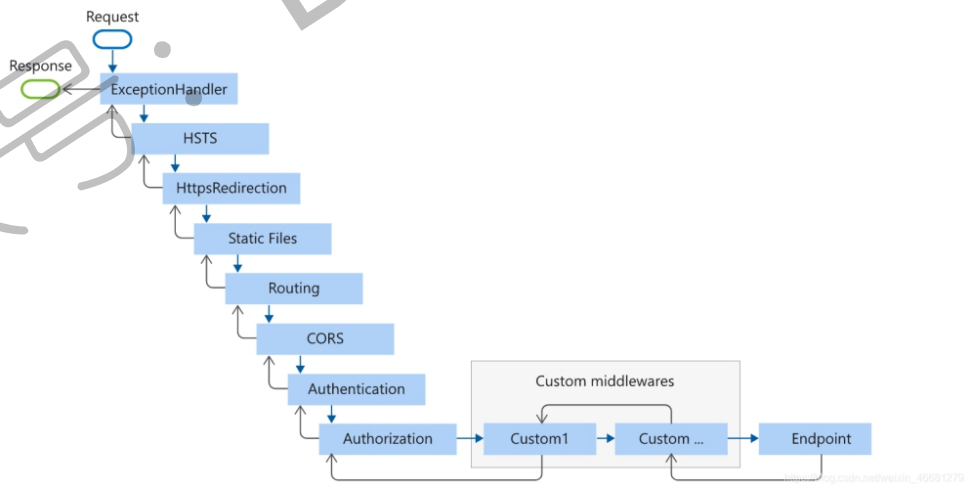
用于授权用户访问安全资源的授权中间件（UseAuthorization)。

会话中间件（UseSession）建立和维护会话状态。如果应用使用会话状态，请在 Cookie 策略中间件之后和 MVC 中间件之前调用会话中间件。

用于将 Razor Pages 终结点添加到请求管道的终结点路由中间件（带有 MapRazorPages 的 UseEndpoints）。

16.ASP.NET Core 中间件的执行顺序?

异常/错误处理
HTTP 严格传输安全协议
HTTPS 重定向
静态文件服务器
Cookie 策略实施
路由
身份验证
会话
MVC



17.application builder 的 use 和 run 方法有什么区别?

这两个方法都在 start up class 的 configure 方法里面调用。都是用来向应用请求管道里面添加中间件的。Use 方法可以调用下一个中间件的添加，而 run 不会。

---->详解

18.dotnet core 管道里面的 map 拓展有什么作用?

可以针对不同的路径添加不同的中间件。

```
public void Configure(IApplicationBuilder app)
{
  app.Map("/path1", Middleware1);
  app.Map("/path2", Middleware2);
}
```

---->详解

19.dot net core 里面的路径是如何处理的?

路径处理是用来为进入的请求寻找处理函数的机制。所有的路径在函数运行开始时进行注册。

主要有两种路径处理方式， 常规路径处理和属性路径处理。常规路径处理就是用 MapRoute 的方式设定调用路径，属性路径处理是指在调用函数的上方设定一个路径属性。

20.dotnet core 工程里面有哪些常见的工程文件?

global, launch setting，app settings，bundle config，bower, package。

21.ASP.NET Core 项目如何设置 IP 地址和端口号?

可以使用 Properties 文件夹下的 launchSettings 配置文件来配置不同的启动方式的时候，分别配置 IP 和端口号。

更多初中高面试题可以扫码关注公众号↓ ↓ ↓ ↓ ↓ ↓ ↓ 欢迎关注

北京群 群1：219690210，群2：377501688，群3：262827065，成都群：209844460
上海群 ：376029918 杭州群：376029918 广州群：344744167 深圳群：542733289
西安群：542733289 高级群：165150112
微信公众号:DotNet开发跳槽

欢迎关注