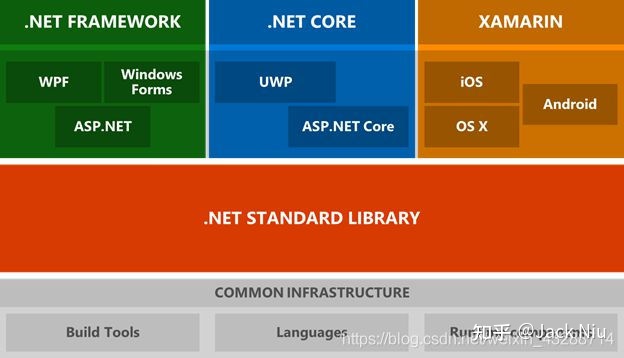
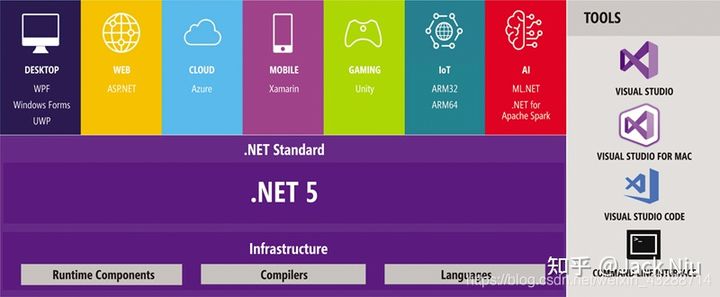
1. NET/dotnet core/.NET framework 的关系

3.x



Dotnet5



* 2016 年开始，分开了.NET Framework和.NET Core，后者开源、面向跨平台应用；
* 2019年，.NET Framework和.NET Core被.NET5完成了统一。未来只会有一个.NET，可以支持 Windows、Linux、macOS、iOS、Android、tvOS、watchOS 和 WebAssembly，等等
* .Net standard library 是之前.NET Framework和.NET Core公共的部分；

1. corefx, coreclr 都是什么？

dotnet 组织包含了.NET Core的核心代码, 包括coreclr和corefx等.

* + coreclr是运行时代码,
  + corefx是各种基础类库的代码. CoreFX主要包含数个公共库，例如 System.Collections, System.IO, System.Xml等。

1. Kestrel

Kestrel是http://asp.net core引入的跨平台的网络服务器，是默认配置。可以没有iis, apache和nginx就可以单独运行。

Kestrel基于libuv实现；（node.js 也是基于libuv）

1. asp.net core 主要的特性

* 跨平台
  + 依赖注入。(扩展什么是依赖注入，以及好处和坏处)
  + 日志系统架构。(日志门面)
  + OWIN(Host、Server、Middleware、Application) ，OWIN除了可以托管web服务，还可以托管其他自定义服务，通过实现IHostedServer接口
  + 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。配置读取简单，可以使用依赖注入来获取
  + 使用start up来注册服务。(x)
  + 更好的支持异步编程。(其实都支持，注意探讨请求处理，dotnetcore是多线程处理，framework可以异步，但还是单线程处理)
  + 支持web socket和signal IR。
  + 对于跨网站的请求的预防和保护机制(跨域？)

1. ASP.NET Core跟ASP.NET比较有哪些更好的地方？

可以说上面的特点

1. 什么是meta packages？

Meta packages是指包含所有ASP dot net code依赖的一个包。叫做Microsoft.AspNetCore.All

1. ASP.NET Core应用能够跟ASP.NET4.x架构一起工作吗？

可以。ASP.NET Core应用可以跟标准的dot net 库一起工作。

1. Aop支持

中间件和filter

1. 什么是中间件？

中间件就是一个个的组件，可以穿插在一个请求的生命周期中，对请求进行处理，多个中间件会构建成一个请求管道，依次执行，也可以提前返回结果。

做认证授权、静态资源处理、跨域、扩展(swagger、websocket)、拦截、日志

(延伸Use, Run, 和 Map)

1. Use, Run, Map和MapWhen

使用中间件的三种方式，Use能够通过next来调用下一个中间件，而Run不能，Map是以路径的方式来添加中间件，指定路径的请求才能进入中间件(类似webapi和控制器)，MapWhen会有一些条件筛选

1. app.UseEndpoints中MapControllers、MapControllerRoute的区别

MapControllers是映射Route配置的路径，也就是webapi的路径

MapControllerRoute是映射控制器的路径，控制器路径一般以控制器名称和控制器的方法(GET、POST等)来定义

1. app.UseRouting、app.UseEndpoints和app.UseMvc
2. 使用cookie、session和memorycache
3. 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。

一脸问号？ 静态文件路径、web api路径和控制器路径

1. 什么是依赖注入、依赖注入的好处和坏处

依赖注入是一个对象托管服务，托管对象的生命周期，从对象的生成到释放。

**坏处**：目前的依赖注入是依赖平台框架的，如果换了平台框架，需要做很大的改动，比如说从dotnet core换回.NET Framework；

**好处**：解耦，这个需要先说对象的依赖关系(拿日志举例)，很多服务都需要使用日志对象来记录日志，每个服务都需要创建一个日志对象，当更换日志驱动的时候，需要每个服务都做修改，而依赖注入是通过提供一个日志接口，让多种日志驱动(NLog,log4net)都实现接口，然后通过依赖注入方式来创建接口的实现类，并注入到服务中，这样当需要更换的时候，直接配置依赖注入的实现类就行了。

1. 依赖注入服务生命周期

Singleton(单例)、Scoped(作用域)、Transient(短暂)

这三个生命周期，单例模式也就是在整个运行周期里只有一个实例，作用于是在每个作用域里只有一个实例，短暂模式是每次获取对象时都会新生成一个实例，

* 但是如果一个单例模式的服务中注入一个短暂模式的对象时，这个对象也是一直都是同一个对象，因为依赖注入是在对象生成时获取依赖的对象来注入的，单例模式的服务在只创建一次，起依赖的短暂模式对象也就只注入一次，也就一直都是同一个对象。
* 作用域服务能注入单例模式服务，但是单例模式服务不能注入作用域服务，因为作用域服务只能在作用域里创建，在创建作用域是会生成一个子容器(IServiceProvider/ IServiceCollection)，用该容器来托管的作用域服务对象，当作用域结束时，子容器也就销毁，而子容器能注入主容器的服务对象，类似于父类和子类。

1. Filter注入

TypeFilter、ServiceType、FilterCollection.Add和FilterCollection.AddService

* TypeFilter 类型描述里说主要用于FilterCollection.Add调用，而FilterCollection.Add是使用ActivatorUtilities来构造filter对象的，所以TypeFilter配置的filter和FilterCollection.Add配置的filter一样，每次都会生成新的filter对象。
* ServiceType类型描述里说主要用于FilterCollection.AddService调用，而FilterCollection.AddService是从容器里获取filter对象的，所以filter对象的生命周期是自己注册filter时的生命周期。
* TypeFilter和ServiceType是在方法和控制器中进行配置，而FilterCollection.Add和FilterCollection.AddService是进行全局配置，所有控制器都会使用。

1. Redirect、Transfer和Execute区别（这个好像是ASP的，可能已经过时）

* Redirect

Redirect是跳转，会返回301/302状态码和location响应给浏览器，浏览器跳转到响应头的地址。整个跳转是由服务端和浏览器端共同完成的。

* Transfer

是后台跳转，浏览器端是看不到的，是将执行完全转移到指定页面

* Execute

是从当前页面转移到指定页面 , 并将执行返回到当前页面

1. 配置文件读取

Configuration.GetSection(“”)

Services.Configre<>

IOption

1. 服务注入方式

构造函数注入，很多第三方容器支持属性注入(Autofac),使用IServiceCollection进行服务注册，使用IServiceProvider进行服务获取，单例服务不能注入作用域服务对象，作用域服务可以注入单例服务对象。

1. Wwwroot

静态资源文件夹

1. 如何设置IP地址和端口号

开发状态配置launchSettings、iis托管时就iis配置，使用kestrel 的web服务来托管的时候就在appsettings.json配置文件中配置kestrel相关的东西。

1. 替换日志为nlog
2. RESTful
3. 什么是跨域

同源策略：是指协议，域名，端口都要相同，其中有一个不同都会产生跨域；

浏览器的同源策略，为了站点安全，不同源时候会受到浏览器的限制，但是script和link标签的资源不受影响。

1. 简单请求和复杂请求

简单请求：get和部分post，post时content-type属于application/x-www-form-urlencoded，multipart/form-data，text/plain中的一种

复杂请求：不是简单请求的请求。

1. 如何处理跨域

* 反向代理

原理，把api和页面放在同一个源下，这样就不会受到浏览器的限制，然后在服务器中使用反向代理来把api和页面代理到其真实的服务器中。

* 后端添加Respon头，原理就是浏览器发现请求是和站点不是同源时，看到Respon中的相关请求头，知道这个请求是支持跨域的，所以不会进行拦截，这个配置起来比较麻烦：

首先需要在服务端返回的数据设置header

Access-Control-Allow-Origin：http://xxxxxxx

Access-Control-Allow-Credentials: true

Access-Control-Allow-Methods: POST,GET

Access-Control-Allow-Headers: x-requested-with,content-type, Authorization

这样浏览器才不会拦截跨域请求。

如果使用cookie/session认证，还需要在ajax中配置跨域携带cookie

xhrFields:{

withCredentials:true,

},

还有配置好cookie 的SameSite，设置为none

SameSite=Strict：设置了该属性的cookie将不会添加到跨域请求里。

SameSite=Lax：设置了该属性的cookie将不会添加到除了GET方法以外的跨域请

所以当使用这种方式需要最好使用token认证，在后端配置respon头access-control-allow-headers : Authorization,然后使用Authorization请求头来传递token

当跨域发送复杂请求的时候，浏览器还会先发送一次OPTIONS请求，options获得的回应是拒绝性质的，比如404\403\500等http状态，就会停止请求的发出。所以还需要处理好OPTIONS请求

* JSONP，这个原理就是利用script和link标签不受跨域影响。

1. 说说你了解到的鉴权授权技术

* Cookie/session,后台写入session时候，会产生一个sessionid，然后把sessionid写入到cookie中，返回给页面，页面每次访问都会携带cookie，后端获取到cookie的内容，根据sessionid获取到session
* Jwt 会把一些需要使用的信息安装jwt格式来构造一个对象，然后进行签名，再通过xxx编码加密为一个token对象，前端存储这个token，每次访问得时候通过Authorization请求头携带该token，后端能直接根据token解密出完整的用户信息，因为能获取到完整的用户信息，就不用再访问数据库进行权限认证。

**优点**：可以多端使用，用cookie/session很多非浏览器端会比较不友好。对集群友好，集群部署的时候cookie/session可能需要把session存储到一个能公共访问的存储媒介(数据库/redis等)，而jwt不用，集群中每个节点都能单独解析出完整的用户信息。

**缺点**：无法及时过期，要处理过期问题，还需要有状态，那样就没有了无状态的优点了

* AccessToken和RefreshToken，使用RefreshToken来获取AccessToken，用AccessToken来请求资源。AccessToken有效期比较短，RefreshToken有效期比较长。

1. 授权认证系统

* 单点登录

单点登录利用的是cookie/session机制

* Oauth

Oauth是利用的AccessToken来授权。Oauth其实没有认证功能，只是进行授权，让其他的服务能获取到自己服务的一些信息(用户名、头像等)，因此当使用第三方登录的时候，往往还是需要注册新的账户

1. Grpc是什么
2. grpc四种连接模式
3. gRPC的拦截器有哪些
4. gPRC安全认证
5. EFCore 如果通过数据生成实体和DbContext

Scaffold-DbContext -Force “Data Source=59.110.21.214; Initial Catalog=test\_DEV; Persist Security Info=True; User ID=sa; Password=123;” Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models/DataModels

1. 说说对SaveChanges的理解

这个要说到ef的跟踪功能，SaveChanges的时候会把跟踪的变化变更到数据库中，

1. EFCore中EntityState的理解

Detached、Unchanged、Added、Deleted、Modified

1. 导航属性和反向导航

在模型中引用其他模型就是一个引用导航，即便没有配置外键，efcore也会根据导航来自动创建外键。

有关系的两个模型都能创建导航，这两个导航互为反向导航

1. ForeignKey、InverseProperty

ForeignKey是指定外键，可以在外键字段上指定导航，也可以在导航上指定外键字段，当单一导航的时候，可以不使用这个，efcore会自动通过阴影导航来添加外键字段。

InverseProperty是用于多个导航的情况，多个导航就使用InverseProperty来指定反向导航。

1. 模型绑定

IModelBinder、ModelBinderProvider、IValueProvider、IValueProviderFactory

* IValueProvider、IValueProviderFactory是用来配置数据源的，从不同的数据源中获取数据，如FormValueProvider、RouteValueProvider、JQueryFormValueProvider等，把这些数据源的数据生成一个字典，提供给模型绑定器
* IModelBinder、ModelBinderProvider是模型绑定相关的，可以自定义模型绑定器，进行模型绑定，默认的几个BodyModelBinder、FormFileModelBinder、ServicesModelBinder、FormCollectionModelBinder
* 模型绑定一般使用IValueProvider就能获取到数据源然后进行数据绑定了，但是BodyModelBinder比较特殊，Body类型的数据会除了x-www-form-urlencoded和form-data这两种key-value的格式外，还可以传输复杂的格式，如json，这时候就需要使用IInputFormatters来解析这些复杂的格式
* FromForm、FromBody、FromQuery、FromService、FromHeader这些特性应该也是会创建成模型绑定器，而且这几个都是会限制数据源的，