.net core 简介

1、历史

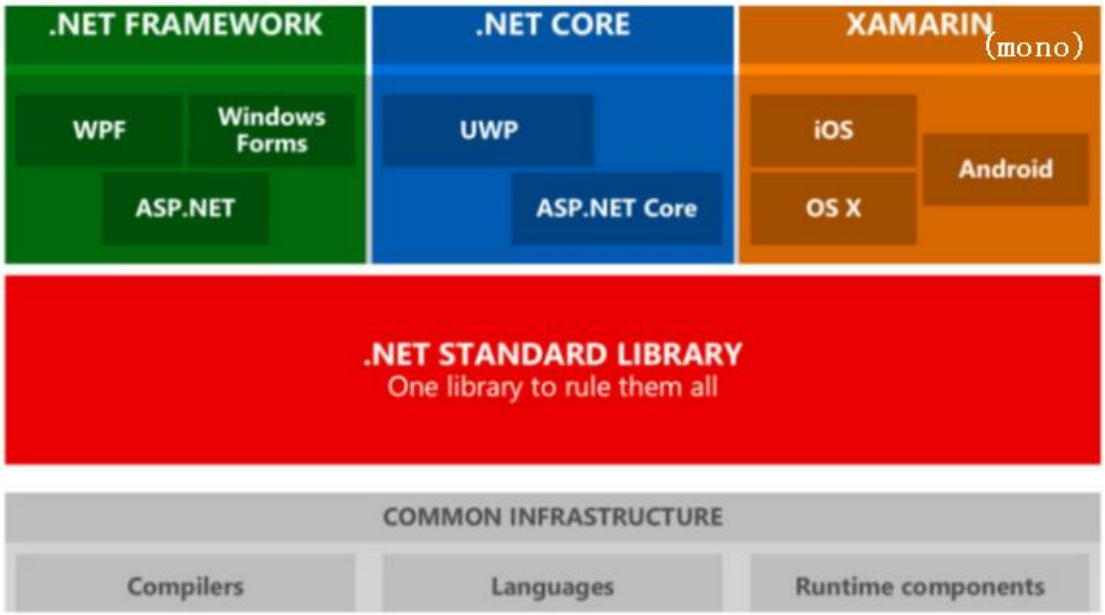
.net 设计之初就是考虑像 java 一样跨平台的，.net framework 是在 windows 下运行的，大部分类是可以兼容移植到 linux 下的，但是没人做这个工作。后来 novell 公司开发了 mono，把大部分.net framework 功能移植到 linux 下。mono 也成为 xamarin（使用.net 开发 Android、IOS app 的技术）和 unity3d（使用.Net 开发 android、ios 游戏的技术）的基础。

.Net core 是微软开发的另外一个可以跨 Linux、windows、mac 等平台的.Net。

2016 年初，微软收购 mono 的公司 xamarin。为什么微软已经收购了 mono，还要搞出来一个.net core？因为 mono 完全兼容.net framework，架构太陈旧，不有利于现在云计算、集群等现在新的架构理念。因此微软推翻重写了.net core。

2、.net framework、.net core、mono 的关系

.net framework、.net core、xamarin 有通用的类，也有特有的。为了保证代码通用，微软定义了公共的.Net Standard Library(.net 标准库，像 FileStream、List 等这些)，按照.Net Standard Library 编写的代码可以在几个平台下通用。



大部分.net framework 中的类在.net core 中还有，方法也还有，只是 namespace 可能变了，有些方法也有不一样，部分 api 缺失（注册表等 windows 平台特有的 api），后面会讲区别。

之前那些在.net framework 中调用的 dll，不一定都能用在.net core(linux)中。

如何判断 dll 是否支持.net core，举例：https://www.nuget.org/packages/NPOI/ 、https://www.nuget.org/packages/Npoi.Core/ 、https://www.nuget.org/packages/ImageProcessor/、https://www.nuget.org/packages/Autofac/

3、.net core 文档

https://github.com/dotnet/docs.zh-cn/tree/master/docs/core github 不定期被封

4、.net core 的版本

在讲课时稳定版本是.net core 1.1，现在有.Net core 2.0 Preview 版，微软计划在 2017 年 Q3 发布.net core 2.0 正式版。Preview 版 api 还不稳定，不建议学习。

5、.net core 在 linux 下的安装

从文档中点击链接安装方法：https://www.microsoft.com/net/core （安装过程中网络可能有问题，会报错，那么重试几次或者尝试翻墙）不要完全按照我上课讲的操作来，根据最新的文档操作。万一执行错了脚本版本怎么办？从 firefox 复制命令的时候一定不要多复制一个结尾的回车。否则可能会在输入完 y 之后“中止”。

找对版本，使用“uname -a”可以查看 linux 的发型版本和内核版本

6、.net core 在 Windows 下的安装

两种安装方法：

1） 按照文档单独安装.net core sdk

2） 安装 visual studio2017 自带.net core

VS2015 不支持.net core 1.1 正式版，只支持预览版。安装 VS2017 的方法（win7 要装 sp1 补丁），可以和其他 VS 版本同时安装。大家请安装，后续要用 VS2017，下载地址：https://www.visualstudio.com/zh-hans/downloads/

.Net Core 简单项目的创建

1、 创建控制台项目

1) dotnet new console -o test1 在当前目录建创建目录 test1，并且初始化控制台(console)类型的项目项目结构（第一次运行 dotnet new 比较慢）。使用命令创建项目，是目前很流行的风格。
2) 也可以手动创建 test1 目录，进入目录，再执行 dotnet new console。也就也就相当于在当前目录下创建控制台项目。
3) dotnet restore 通过 nuget 还原安装当前目录的项目用到的包，一定要 cd 到项目根目录下执行。restore 执行消息里给出了 nuget 下载目录的路径。
4) 改一下项目的代码；
5) dotnet run 编译并运行当前目录的项目，一定要在项目根目录下执行。如果编译报错就根据报错信息改代码或者是忘了 dotnet restore 了
6) 注意.net core 下的控制台程序不是生成 exe 的，要通过“dotnet 入口 dll 文件名”方式运行

三个步骤：new、restore、run

2、 asp.net mvc core 项目

1） dotnet new mvc -o test1 或者 dotnet new mvc
2） dotnet restore
3） 改一下项目的代码；
4） dotnet run web 项目自带嵌入式服务器，测试阶段不用 IIS 等单独的服务器，部署阶段再部署到 iis、Nginx 等上（后面讲）Owin-SelfHost
5） 在 Linux 的服务器的浏览器中打开 http://127.0.0.1:5000（只能本机访问，以后讲通过 Nginx 配置远程访问）。
6） ctrl+c 停止服务器。
7） 修改默认绑定的端口的方法：在 Program.cs 的 Build 之前加入 UseUrls("http://*:5001");

```
public class Program
{
    public static void Main(string[] args)
    {
        var host = new WebHostBuilder()
            .UseKestrel()
            .UseContentRoot(Directory.GetCurrentDirectory())
            .UseIISIntegration()
            .UseStartup<Startup>()
            .UseUrls("http://*:5001")
            .Build();


        host.Run();
    }
}
```

3、创建其他项目

.net core 目前只支持控制台和 asp.net mvc core，不支持 winform 和 webform，估计永远

都不会支持。

创建类库项目使用 dotnet new classlib；webapi 项目：dotnet new webapi；解决方案：dotnet new sln

还有哪些可用？dotnet new --help

4、历史问题

旧版本.Net core 曾经用 project.json 做项目描述文件，后来又改回了 project.csproj

5、.Net Core 的两种开发方式：

用 vi 编辑代码没有自动提示，开发麻烦，所以只是进行原理演示。不会真的用 vi 写代码。

方式 1：Windows 下用 VS 开发，然后部署到 Linux 下部署运行；

方式 2：Linux 下使用 VSCode 开发，然后 Linux 下部署运行，VSCode 还是没有 VS 强大。

6、dotnet 命令文档 https://docs.microsoft.com/en-us/dotnet/articles/core/tools/dotnet

(*)Visual Studio Code 使用

1、 VSCode 介绍

Visual Studio Code（VSCode）是微软出的跨平台的轻量级开发工具，支持 Windows、Linux、Mac 等系统。VSCode 只是借着 VS 的名气，和 VS 没有直接的关系，不像 VS 那么庞大、全面，很轻量级，要装插件。

可以安装插件进行 C#、java、python、js 等开发。口碑不错。

如果做.net core 开发，尽量还是选择用 VS，只有在不支持 VS 的操作系统下才选择 VSCode。尽量还是 windows 下用 VS 开发，然后部署到 Linux 下运行。

微软官网 的话：if you want to have a lightweight tool to edit a file - VS Code has you covered. If you want the best possible experience for those projects and development on Windows in general, we recommend you use Visual Studio Community.

如果 VSCode 用的实在装不上、用着有问题，就不用了，用 VS。

2、 Linux 下安装 VSCode

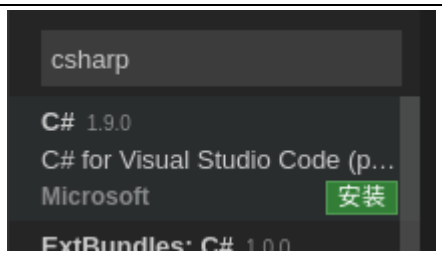
官网地址 https://code.visualstudio.com/

因为 vscode 是图形界面程序，因此需要在桌面环境下安装，不能在终端中装和使用。

使用  搜索计算机，输入“visual studio code”搜索就可以打开，可以把图标拖到启动器中。用不到的图标可以点右键【从启动器解锁】

3、 配置 C#开发环境

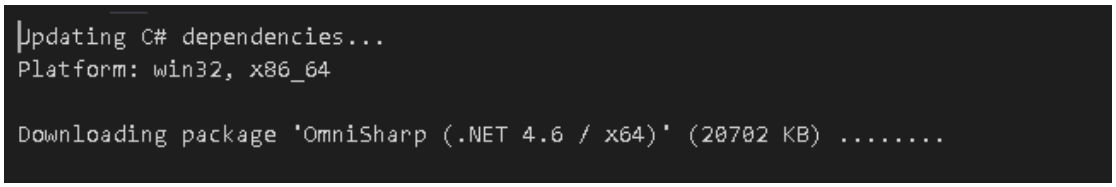
vscode 默认没有 C#开发插件，需要单独安装。根据 https://marketplace.visualstudio.com/items?itemName=ms-vscode.csharp 这里的说明，在 VSCode 中按 ctrl+p，执行 ext install csharp

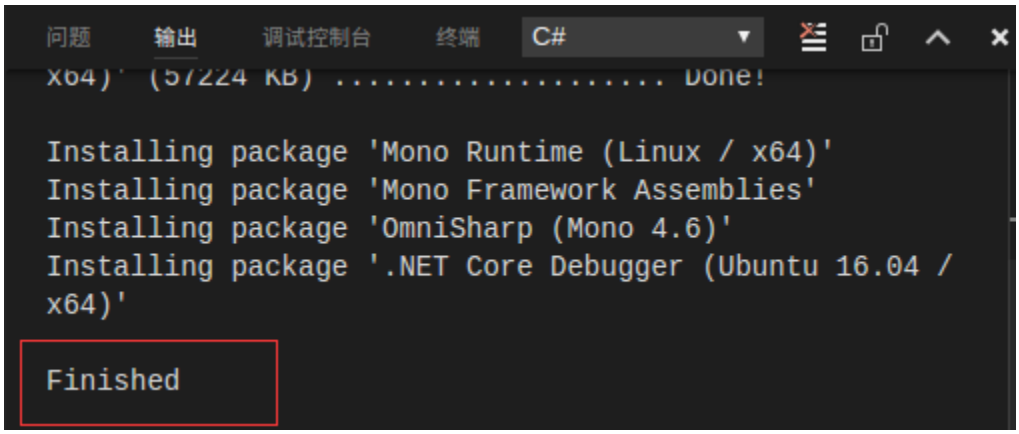


选这个 C#点【安装】

VSCode 中没有 vs 中新建项目的对话框，要使用 dotnet new 命令创建项目文件，然后使用打开文件夹的功能打开项目根目录（之前不要忘了先 dotnet restore）。

没有自动提示？打开一下 csproj 文件、再打开 C#文件，等一会下载加载插件





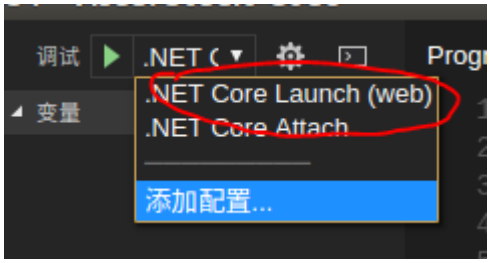
等他下载完成，重启一下 VSCode，就可以了。

如果网络原因下载不了，就跳过，直接用 VS。

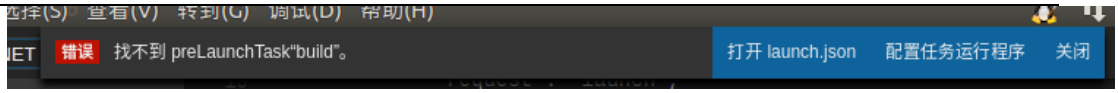
类、cshtml 等也没有模板向导，要自己建空文件。

4、 运行项目

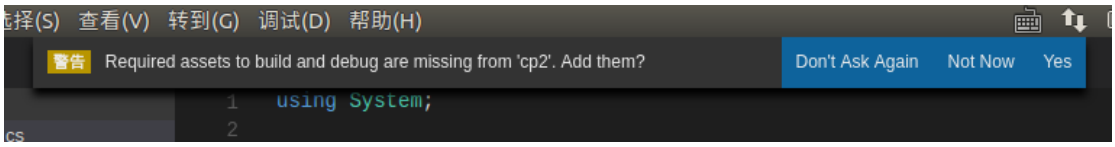
如果是 Web 项目，直接点虫子图标  中的



调试可能会报错



打开 Program.cs 稍等片刻会有如下提示，点击【Yes】



可以在底部的“终端”视图执行命令，避免了切换到单独的终端窗口。

5、 窗口多开

一个窗口只可以打开一个文件夹，可以 ctrl+shift+n 打开多个窗口。使用主菜单打开其他文件夹（Linux 的主菜单和 Windows 不一样，独立于窗口，显示在最顶端）。

在启动器图标上点右键切换同一个程序的多个实例。

通过命令行创建解决方案

1、 如何创建多个项目的解决方案：
   1) 先创建解决方案文件夹 rupengbbs，然后在其中 dotnet new sln
   2) dotnet new mvc -o rupengbbs.web 说明：创建 web 项目
   3) dotnet new classlib -o rupengbbs.common 说明：创建 common 项目
   4) dotnet new sln 说明：解决方案，解决方案的名字默认是当前目录的名字
   5) dotnet sln rupengbbs.sln add rupengbbs.common/rupengbbs.common.csproj 说明：把 rupengbbs.common 项目中的 rupengbbs.common.csproj 添加到解决方案文件中。注意最后一个参数在/前后不要加空格，这指的是 rupengbbs.common 目录下的 rupengbbs.common.csproj 文件。
   6) dotnet sln rupengbbs.sln add rupengbbs.web/rupengbbs.web.csproj 说明：把 web 项目添加到解决方案中
   7) dotnet add rupengbbs.web/rupengbbs.web.csproj reference rupengbbs.common/rupengbbs.common.csproj 说明：rupengbbs.web.csproj 项目添加对 rupengbbs.common.csproj 项目的引用
   8) dotnet restore 说明：在解决方案下每个项目中执行 dotnet restore。如果是在某个项目下执行 dotnet restore 则只是 restore 某个项目。
2、 查看一下 csproj 和 sln 文件格式，知道如何手动改。
3、 vscode 打开解决方案文件夹即可。在 common 项目中建一个 Person.cs，写一个 Hello 方法，然后在 Web 项目中调用。
4、 (*)编译整个解决方案的方法，在解决方案文件夹下 dotnet build rupengbbs.sln
5、 手动创建三层项目：Web 项目、Model 项目、DAL 项目、BLL 项目。
6、 写一个创建三层项目解决方案的脚本，体现命令行的好处。脚本中的$1、$2 代表第 1、2 个参数的值。如果某一些操作需要反复执行，那么就可以写成脚本。

```
mkdir $1
```

```
cd $1
dotnet new mvc -o $1.web
dotnet new classlib -o $1.model
dotnet new classlib -o $1.dal
dotnet new classlib -o $1.bll
dotnet new sln
dotnet sln $1.sln add $1.web/$1.web.csproj
dotnet sln $1.sln add $1.model/$1.model.csproj
dotnet sln $1.sln add $1.dal/$1.dal.csproj
dotnet sln $1.sln add $1.bll/$1.bll.csproj
dotnet add $1.web/$1.web.csproj reference $1.model/$1.model.csproj
dotnet add $1.web/$1.web.csproj reference $1.bll/$1.bll.csproj
dotnet add $1.bll/$1.bll.csproj reference $1.dal/$1.dal.csproj
dotnet add $1.bll/$1.bll.csproj reference $1.model/$1.model.csproj
dotnet add $1.dal/$1.dal.csproj reference $1.model/$1.model.csproj
dotnet restore
```

比如保存成 cjsc.sh，然后要创建三层项目的时候执行“bash ./ cjsc.sh RuPeng”会自动把项目和解决方案创建起来了。

因为命令行所有脚本，因为脚本所以“自动化”。Windows 下的脚本语言有传统的 bat 以及新的 Powershell，Windows10 安装 Linux 子系统后也可以使用 Bash。

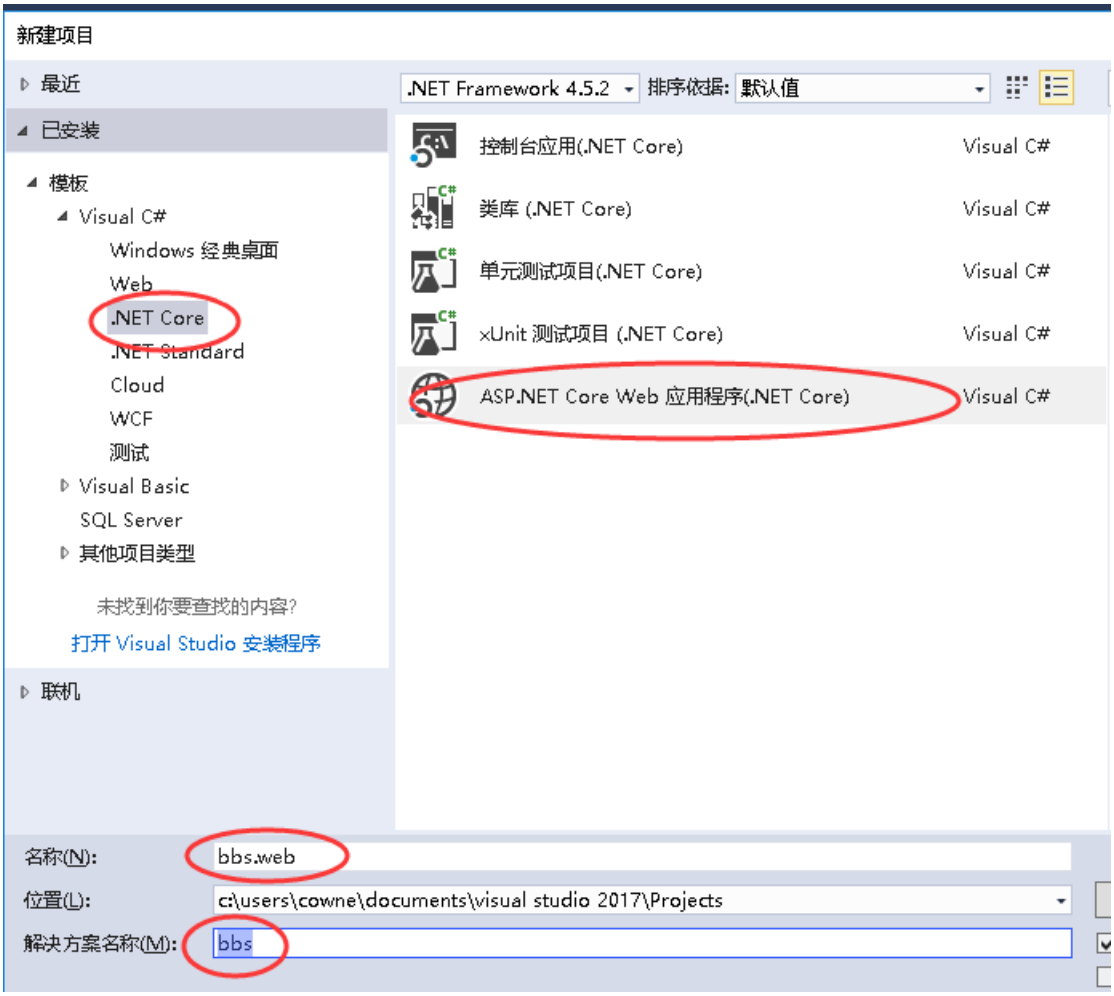
7、 项目添加 nuget 引用的方法：在项目下执行 dotnet add package Newtonsoft.Json，然后 dotnet restore。dotnet add package 就等价于 nuget 的 Install-Package。(*)还可以通过 dotnet addnew nugetconfig 创建一个 nuget 配置文件，指定源。

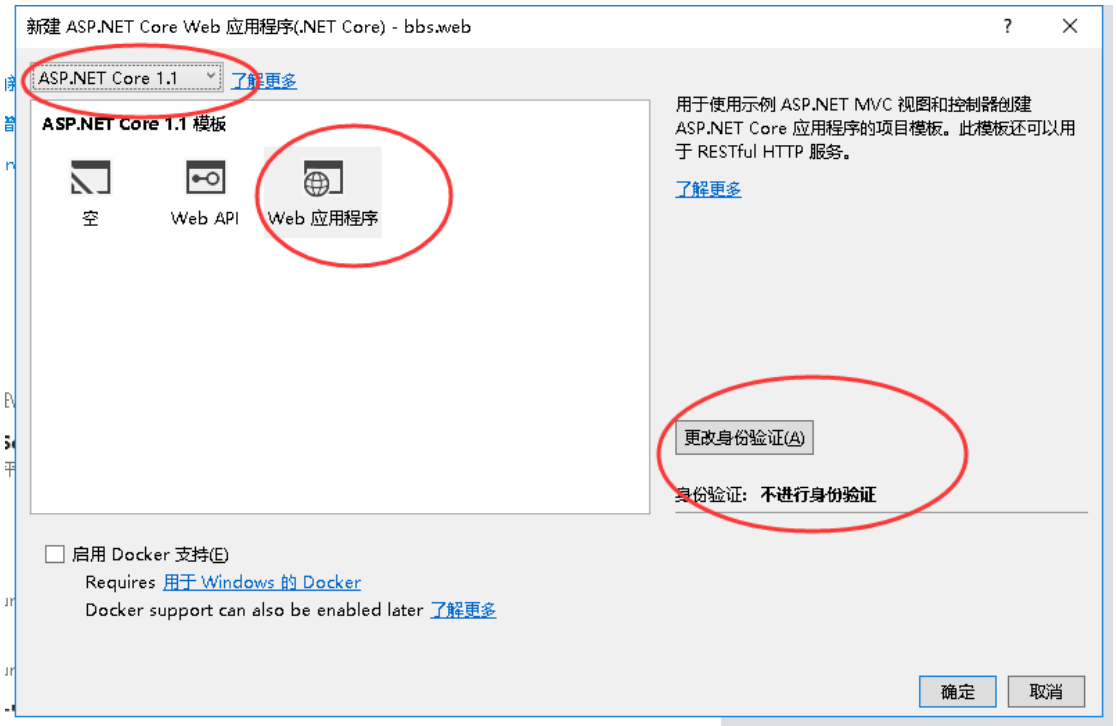
VisualStudio 创建.Net 解决方案

1、 .net core 开发还是在 Windows 下用 VS 开发最好，由于.net core 是跨平台的，所以之前讲的命令在 windows 下也能用（演示一下）。.sh 脚本是 Linux 特有的，不能直接在 windows 下用，Win10 可以安装“Linux 子系统”之后也可以用 bash 脚本（兼容性没测试过，知道即可）。

2、 至少使用 VS2017，确保安装了.net core 的组件，使用.net core 1.1。

3、 VS 创建项目的向导





4、 创建类库一定要创建“类库(.Net Core)”，不能引用.Net Framework 的类库

5、 引用：
   a) 项目之间引用，在项目的“依赖项”中点右键，选择【添加引用】
   b) 不能直接添加 dll 的引用，.net core 项目统一通过 nuget 添加对其他组件的引用。和.net framework 项目的“程序包管理器控制台”一样，也是用 Install-Package 指令，一定要确保添加的支持.net core。nuget 添加稍等一会自动提示才能用，有时候还要重启 VS。
   c) 可以搭建 nuget 私服。
6、 控制台、Web 程序部署到 Linux 等服务器的统一方式：服务器上先要安装.net core，然后在开发环境发布，然后把发布包上传到服务器，然后到目录下执行“dotnet 主程序 dll”即可。

.net core 和.net framework 对比

1、 .Net Core 缺失的类：不支持 DataTable 、DataSet；不支持 AppDomain；WinForm； WPF；Windows 特有类；不支持.Net Remoting；不支持二进制序列化；不支持 TransactionScope；不支持 System.Net.Mail 邮件发送（可以用 MailKit，http://www.cnblogs.com/pengze0902/p/6562447.html ）；串口通讯。
2、 .net core 如何使用 SqlConnection?要 nuget 安装：System.Data.Common（ADO.Net 基础类库） 和 System.Data.SqlClient（SqlServer）。
3、 反射：.net core 中大部分方法在 type.GetTypeInfo()里(需要 using System.Reflection，都属于扩展方法，不是 Type 类的成员)。而.net framework 下则主要是在 type 类下。
4、 .net core 1.x 中不支持 System.Drawing（验证码、水印、缩略图等使用），使用 ZKWeb.System.Drawing 等，（参考 http://www.cnblogs.com/niao/p/6063587.html）。linux 下要安装 libgdiplus（安装过程要用 sudo，安装完成后必须重启程序）.net core 2.0 后就要支持 System.Drawing 了。
   1) nuget 安装 ZKWeb.System.Drawing
   2) 正常写 GDI 代码：

```
using (Stream fs = File.OpenWrite("1.jpg"))
using (Bitmap bmp = new Bitmap(50, 50))
using (Graphics g = Graphics.FromImage(bmp))
{
    g.Clear(Color.White);
    g.DrawString("hello", new Font(FontFamily.GenericSansSerif, 20), Brushes.Black, new PointF(0, 0));
    bmp.Save(fs,System.DrawingCore.Imaging.ImageFormat.Jpeg);
}
```

   3) Windows 下可以直接运行。Linux 下需要运行：
      a) sudo apt-get install libgdiplus
      b) cd /usr/lib
      c) sudo ln -s libgdiplus.so gdiplus.dll
   b) 做图形验证码用 https://www.nuget.org/packages/CaptchaGen.NetCore/

5、 (*)SHA1CryptoServiceProvider 换成了 SHA1.Create()

6、 (*)TimeZone 换成了 TimeZoneInfo

7、 HttpWebRequest、WebClient 不支持，必须用异步的 HttpClient（后面讲）

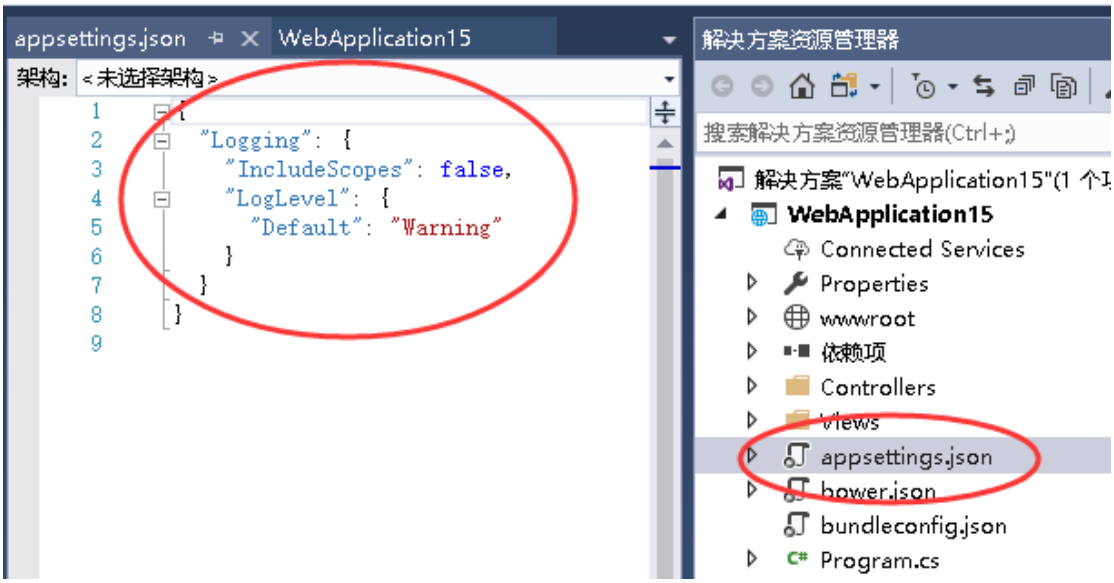
8、 Ilspy、reflector 都无法反编译.net core 程序，可以用 ConsoleApp3.zip c

9、 .Net Standard 和.net core、.Net framework 各版本的关系

http://www.cnblogs.com/zjoch/p/6696986.html

.Net Core 中配置文件的解析

.Net core 的配置文件，不再是配置在 web.config 中，而是单独的 json 配置文件（https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration）。



只要和合法的 Json 格式，怎么写随意，怎么写就怎么解析

解析方法：

1、 首先 Nuget 安装：Microsoft.Extensions.Configuration 和 Microsoft.Extensions.Configuration.Json

2、 然后 using Microsoft.Extensions.Configuration;

3、 下面的代码就把 Logging 下的 LogLevel 下的 Default 值读取出来了：

```
var builder =
    new ConfigurationBuilder().SetBasePath(Directory.GetCurrentDirectory())//SetBasePath 设置配置文件所在路径
    .AddJsonFile("appsettings.json");
var configRoot = builder.Build();
var value =
    configRoot.GetSection("Logging").GetSection("LogLevel").GetSection("Default").Value;
System.Console.WriteLine(value);
```

EFCore 的使用

EF Core（Entity Framework Core）是 EF 的.net core 版本。EF Core 对 SQLServer 支持很好，也可以在 Linux 下连接 SQLServer。不过如果在 Linux 下首选 MySQL，因此我们主要介绍 MYSQL 中使用 EF Core。SQLServer 用法几乎一样，只是换一个 EF Provider 的 NuGet 包而已。

EFCore 的 Nuget：Microsoft.EntityFrameworkCore

EFCore 官方文档：https://docs.microsoft.com/en-us/ef/core/index

1、官方的 mysql ef provider，还处于 preview 阶段；https://www.nuget.org/packages/MySql.Data.EntityFrameworkCore/

2、可以试试第三方的 EF core Provider。用 Nuget 安装 Pomelo.EntityFrameworkCore.MySql（依赖于 Pomelo.Data.MySql）用法参考 http://www.1234.sh/post/pomelo-data-mysql 如鹏的 UserCenter 接口用的这个，目前没发现问题。

3、Pomelo.EntityFrameworkCore.MySql 是使用数据库的默认字符集创建数据库，因此如果 mysql 是 latin1 字符集，那么数据库也会是 latin1 字符集，插入中文就会出错。因此要先手动创建 utf8 的数据库。

4、使用 MYSQL EF Core

a) 把 mysql 数据库、表创建起来；

b) Install-Package Pomelo.EntityFrameworkCore.MySql

c) 编写 Person 类

```
public class Person
{
    public long Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public bool? Gender { get; set; }
}
```

d) 编写 DbContext，ef core 的 DbContext 等 EF 的核心类在 using Microsoft.EntityFrameworkCore;这个 namespace 下

```
using Microsoft.EntityFrameworkCore;

namespace ConsoleApp4
{
    class MyDbContext:DbContext
    {
        public DbSet<Person> Persons { get; set; }
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            base.OnConfiguring(optionsBuilder);
            optionsBuilder.UseMySql("Server=127.0.0.1;database=zszdb;uid=root;pwd=root");
```

```
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            var etPerson = modelBuilder.Entity<Person>();
            etPerson.ToTable("t_persons");
        }
    }
}
```

通过 UseMysql 这样的扩展方法来配置连接字符串，这是.Net core 的风格！可以把连接字符串写到配置文件中，然后再读取。

5、 EF Core 和 EF 的区别：

a) 要使用 AsNoTracking、Include 等要 using Microsoft.EntityFrameworkCore，不能使用 System.Data.Entity

b) 目前还不支持 LazyLoad，需要显式的 Include

c) 没有内置 EntityTypeConfiguration（要么手动注册 Config 类，复习一下；要么后续课程会给大家提供一个）

d) 一对多关系配置从 builder.HasRequired(e => e.Author).WithMany();改成：HasOne(e => e.Author).WithMany().HasForeignKey(e=>e.AuthorId).IsRequired(); 或者 builder.HasOptional(e => e.Author).WithMany(); 改成： HasOne(e => e.Author).WithMany().HasForeignKey(e=>e.AuthorId);

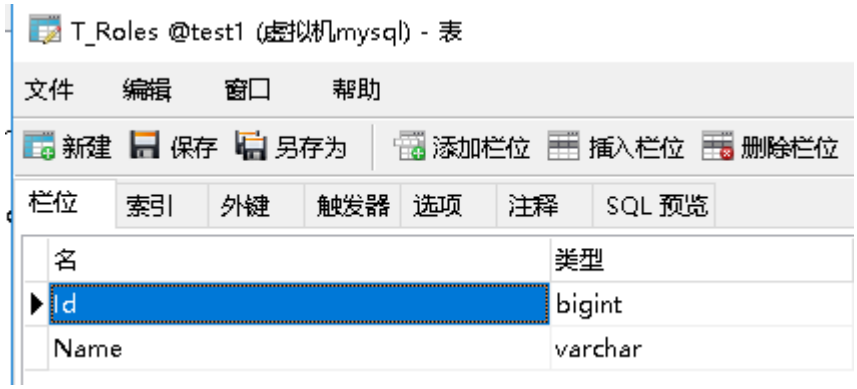
e) 不用中间实体的多对多还不支持，要自己拆成用中间实体的两对一对多。

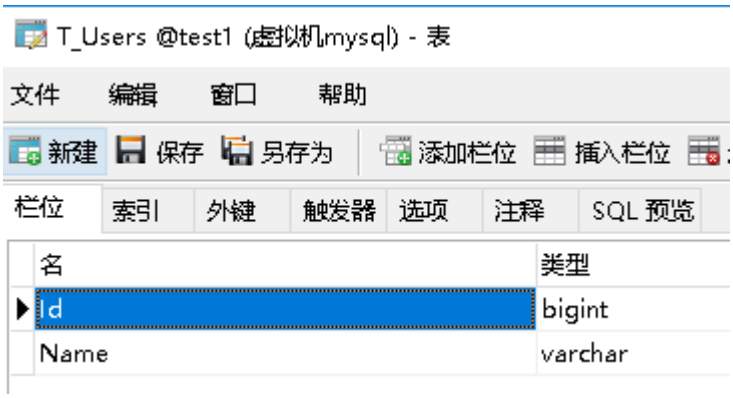
f) 只要把配置文件放到 UI 项目中即可，不再不需要在 UI 项目再 nuget 安装 EF

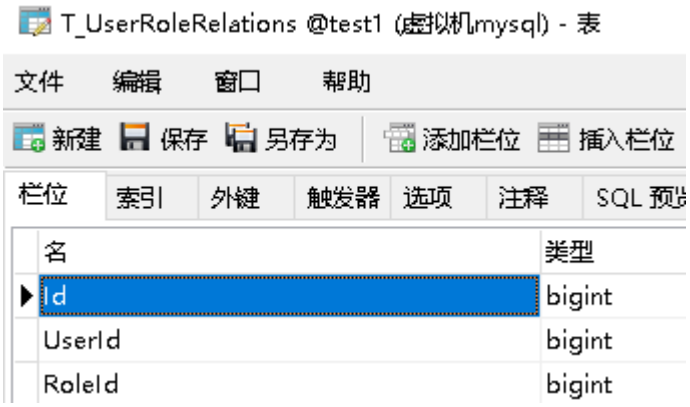
g) 控制台程序运行时候乱码：nuget 安装 System.Text.Encoding.CodePages，然后在最开始 Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

6、 多对多配置：

a) 创建表







b) 创建实体类

```
public class Role
{
    public long Id { get; set; }
    public string Name { get; set; }
}
public class User
{
    public long Id { get; set; }
    public string Name { get; set; }
}
public class UserRoleRelation
{
    public long Id { get; set; }
    public long UserId { get; set; }
    public long RoleId { get; set; }
    public User User { get; set; }
    public Role Role { get; set; }
}
```

c) 配置

```
class MyDbContext:DbContext
{
```

```
    public DbSet<User> Users { get; set; }
    public DbSet<Role> Roles { get; set; }
    public DbSet<UserRoleRelation> UserRoleRelations { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        base.OnConfiguring(optionsBuilder);
        optionsBuilder.UseMySql("Server=192.168.1.12;database=test1;uid=root;pwd=root");
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        var typeUser = modelBuilder.Entity<User>();
        typeUser.ToTable("T_Users");

        var typeRole = modelBuilder.Entity<Role>();
        typeRole.ToTable("T_Roles");

        var typeUserRoleRelation = modelBuilder.Entity<UserRoleRelation>();
        typeUserRoleRelation.ToTable("T_UserRoleRelations");

        typeUserRoleRelation.HasOne(e => e.Role).WithMany().HasForeignKey(e => e.RoleId).IsRequired();
        typeUserRoleRelation.HasOne(e => e.User).WithMany().HasForeignKey(e => e.UserId).IsRequired();
    }
}
```

d) 验证

```
using (MyDbContext ctx = new MyDbContext())
{
    var user = ctx.Users.First();
    long userId = user.Id;
    var relactions = ctx.UserRoleRelations//.Include(e => e.Role)
        .Where(r => r.UserId == userId);
    foreach (var relation in relactions)
    {
        Console.WriteLine(relation.Role.Name);
    }
}
```

7、 自己编写 EntityTypeConfiguration

1）文件 IEntityTypeConfiguration.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
```

```
namespace RuPengBBS.Service.Configs
{
    public interface IEntityTypeConfiguration
    {
        void Map(ModelBuilder builder);
    }

    public interface IEntityTypeConfiguration<T> : IEntityTypeConfiguration where T : class
    {
        void Map(EntityTypeBuilder<T> builder);
    }
}
```

2）文件 EntityTypeConfiguration.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace RuPengBBS.Service.Configs
{
    public abstract class EntityTypeConfiguration<T> : IEntityTypeConfiguration<T> where T : class
    {
        public abstract void Map(EntityTypeBuilder<T> builder);

        public void Map(ModelBuilder builder)
        {
            Map(builder.Entity<T>());
        }
    }

}
```

3）文件 ModelBuilderExtenions.cs

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Reflection;
using System.Linq;

namespace RuPengBBS.Service.Configs
{
    public static class ModelBuilderExtenions
    {
        private static IEnumerable<Type> GetMappingTypes(this Assembly assembly, Type
mappingInterface)
```

```
            {
                return assembly.GetTypes().Where(x => !x.GetTypeInfo().IsAbstract &&
    x.GetInterfaces().Any(y => y.GetTypeInfo().IsGenericType
                    && y.GetGenericTypeDefinition() == mappingInterface));
            }

            public static void AddEntityConfigurationsFromAssembly(this ModelBuilder modelBuilder,
    Assembly assembly)
            {
                var mappingTypes = assembly.GetMappingTypes(typeof(IEntityTypeConfiguration<>));
                foreach (var config in
    mappingTypes.Select(Activator.CreateInstance).Cast<IEntityTypeConfiguration>())
                {
                    config.Map(modelBuilder);
                }
            }
        }
    }
```

4）PostConfig.cs 单独定义

```
    using RuPengBBS.Service.Entities;
    using Microsoft.EntityFrameworkCore.Metadata.Builders;
    using Microsoft.EntityFrameworkCore;//要加上

    namespace RuPengBBS.Service.Configs
    {
        class PostConfig:EntityTypeConfiguration<PostEntity>
        {
            public override void Map(EntityTypeBuilder<PostEntity> builder)
            {
                builder.ToTable("T_Posts");
                builder.Property(e => e.Content).IsRequired();
                builder.Property(e => e.Title).IsRequired().HasMaxLength(250);
                builder.HasOne(e =>
    e.Author).WithMany().IsRequired().HasForeignKey(e=>e.AuthorId);//和EF不一样的地方
            }
        }
    }
```

5）在 DbContext 的 **OnModelCreating 方法**中：

```
modelBuilder.AddEntityConfigurationsFromAssembly(Assembly.GetEntryAssembly());//参数表示 config
```

类所在的程序集

6）我们已经把这个组件发布到 Nuget 上，大家以后执行：Install-Package RuPeng.EFCore.EntityTypeConfig 就可以安装

8、 连接SQLServer：http://www.cnblogs.com/baobaodong/p/5870851.html 除了UseMySql改成

UseSqlServer，其他几乎没有变化。

asp.net mvc core 入门

1、 asp.net core的官方文档：https://docs.microsoft.com/en-us/aspnet/core/，部分文档有中文的：https://docs.microsoft.com/zh-cn/aspnet/core/tutorials/index

2、 asp.net core两种运行方式：SelfHost及IIS集成，在VS中调试的时候也有这两种方式。推荐用SelfHost。

3、 讲解项目结构：wwwroot是静态文件的路径，这和asp.net mvc不一样；appsettings.json是默认的配置文件；Startup.cs是项目初始化类，讲解一下 Startup.cs的代码，env.IsDevelopment()这个部分叫做“Staging集成”，在开发阶段、生产阶段程序可以有不同行为，通过在WebHostBuilder中调用UseEnvironment来设置Staging，app.UseMvc中在配置路由；Program.cs是以SelfHost方式运行的时候的配置代码：Kestrel是.net core内置的SelfHost服务器；UseIISIntegration支持集成运行到iis中；UseStartup设置Startup类。

asp.net core IOC

1、 asp.net mvc core 内置了 IOC 容器，不再需要 autofac 等，当然 autofac 也是支持.net core 的（http://www.open-open.com/lib/view/open1454127071933.html）。内置 IOC 是通过构造函数注入，而不是属性注入。

2、 在 Startup 的 ConfigureServices 中进行注入的准备工作。

3、 内置的 IOC 有三种生命周期：

Transient： Transient 服务在每次被请求时都会被创建。这种生命周期比较适用于轻量级的无状态服务。

Scoped： Scoped 生命周期的服务是每次 web 请求被创建。

Singleton： Singleton 生命能够周期服务在第一被请求时创建，在后续的每个请求都会使用同一个实例。如果你的应用需要单例服务，推荐的做法是交给服务容器来负责单例的创建和生命周期管理，而不是自己来走这些事情。

调用方法 services.AddSingleton(typeof(IMyService),new MyService());进行注册

但是最好 services.AddSingleton(typeof(IMyService),typeof(MyService));

因为这样的话可以在 MyService 中通过构造函数注入其他服务。

4、 Controller 注入

构造函数可以注入多个参数

在 Controller 中要使用构造函数注入（不是属性注入）。

下面的代码使用反射把所有服务接口进行了注入：

```
var serviceAsm = Assembly.Load(new AssemblyName("RuPengBBS.Service"));
foreach(Type serviceType in serviceAsm.GetTypes()
.Where(t=>typeof(IServiceTag).IsAssignableFrom(t)&&!t.GetTypeInfo().IsAbstract))
{
```

```
    var interfaceTypes = serviceType.GetInterfaces();
    foreach(var interfaceType in interfaceTypes)
    {
        services.AddSingleton(interfaceType, serviceType);
    }
}
```

5、 其他类注入

在其他类怎么使用注入？假如在 ExceptionFilter 中想调用 IUserService 怎么办？要确保 ExceptionFilter 不是 new 出来的，而是 IOC 创建出来

```
services.AddSingleton(typeof(ExceptionFilter));
//mvc core中注册filter要在AddMvc的回调方法中注册。
services.AddMvc(options => {
    var serviceProvider = services.BuildServiceProvider();
    var filter = serviceProvider.GetService<ExceptionFilter>();
    options.Filters.Add(filter);
});
```

在 Startup 的 ConfigureServices 方法中

```
services.AddSingleton(typeof(ExceptionFilter));//把 ExceptionFilter 注入
services.AddMvc(options => {
    var serviceProvider = services.BuildServiceProvider();
    var filter = serviceProvider.GetService<ExceptionFilter>();//获得 ExceptionFilter 对象，而不是 new
    options.Filters.Add(filter);
});
```

这样就可以在 ExceptionFilter 类中通过构造函数注入服务了。

6、 怎么手动获取对象

可以把 serviceProvider 搞成 static 的，这样其他类中可以通过 serviceProvider.GetService<IUserService>()这种手动获得对象了。

在 HttpContext 可用的时候，也可以通过这种方法来解析服务：IPostService ps = (IPostService)this.HttpContext.RequestServices.GetService(typeof(IPostService));

7、 内置服务

asp.net mvc core 中框架注入了很多服务，详细见 https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection

最有用的就是 IHostingEnvironment，其中主要成员有：WebRootPath 属性（wwwroot 文件夹的物理路径）；ContentRootPath 属性（网站根目录的物理路径）。Microsoft.AspNetCore.Hosting 下的 HostingEnvironmentExtensions 下还提供了一些扩展方法：IsDevelopment()是否是开发环境、IsProduction()是否是生产环境。

asp.net mvc core 中没有 Server.MapPath()方法，根据你要定位的文件是不是在 wwwroot 下，你可以使用 IHostingEnvironment. WebRootPath 或者 IHostingEnvironment. ContentRootPath 来进行拼接。

对比学习 asp.net mvc core

1、 缓存。asp.net mvc core不再支持HttpContext.Cache，是显式的通过服务的形式支持“内存Cache”（ https://docs.microsoft.com/en-us/aspnet/core/performance/caching/memory）、“分布式Cache（https://docs.microsoft.com/en-us/aspnet/core/performance/caching/distributed）”

2、 Session，需要添加对Session支持，否则会报错Session has not been configured for this application or request。使用方法 http://www.cnblogs.com/sword-successful/p/6243841.html

   a) nuget 安装 Microsoft.AspNetCore.Session
   b) ConfigureServices 中 services.AddSession();
   c) Configure 中 app.UseSession();
   d) TempData 依赖于 Session，所以也要配置 Session。
   e) HttpContext.Session，但是原始只有void Set(string key, byte[] value)、bool TryGetValue(string key, out byte[] value)这两个方法。如果using Microsoft.AspNetCore.Http;（需要安装Microsoft.AspNetCore.Http.Extensions）还可以使用SessionExtensions中的值是int、string类型的，其他类型只能自己使用json进行序列化处理。
   f) 推荐使用redis做进程外session：http://www.hossambarakat.net/2016/02/03/configuring-redis-as-asp-net-core-1-0-session-store/

3、 asp.net core 中的 Json()序列化会自动处理“开头字母小写”、“Date 问题”，所以不需要再自定义 Filter 使用 Newtonsoft.Json 来解决。

4、 asp.net core 中不会再自动检查请求中的 html 标签(因此也就没有[ValidateInput]了)，因此对于不确认安全性的内容输出的时候一定不要 @Html.Raw() http://stackoverflow.com/questions/39061401/enable-asp-net-core-request-validation

5、 asp.net mvc core 中文件上传对应的是 IFormFile 类型 https://www.mikesdotnetting.com/article/288/uploading-files-with-asp-net-core-1-0-mvc

6、 当需要使用 HtmlEncode 和 UrlEncode 的时候只要注入 HtmlEncoder、UrlEncoder 即可。如果没有的话需要 Install-Package System.Text.Encodings.Web

7、 参数用 IFormCollection 代替 FormCollection

8、 Views 中默认引入的 namespace 不再是写到 Views 的 web.config 中，而是_ViewImports.cshtml 中加入@using RuPengIMServer 这样的

ASP.Net Core 日志

1、 asp.net core 内置了日志机制，统一了日志操作的接口，提供了基础的日志 Provider，同时允许第三方日志组件。https://docs.microsoft.com/en-us/aspnet/core/fundamentals/logging

只要通过容器获得 ILoggerFactory 实例即可：

```
public class ExceptionFilter : IExceptionFilter
{
```

```
        private ILogger logger;

        public ExceptionFilter(ILoggerFactory logFactory)
        {
            this.logger = logFactory.CreateLogger(typeof(ExceptionFilter));
        }

        public void OnException(ExceptionContext context)
        {
            this.logger.LogError("未处理异常" + context.Exception);
        }
}
```

项目模板的Startup的Configure方法中

```
loggerFactory.AddConsole(Configuration.GetSection("Logging"));
loggerFactory.AddDebug();
```

就是在配置日志输出到什么地方。可以配置到 Debug 输出中（VS 中【输出】视图可以看到）

还内置了输出到 EventLog、Azure App Service 的 Provider。

还可以添加第三方日志 Provider：NLog 等。

2、 NLog 以在线文档为准

参考文档 https://github.com/NLog/NLog.web/wiki

3、 (*)Log4Net

.net core 中也可以使用 Log4Net 参考：http://www.cnblogs.com/linezero/p/log4net.html

但是目前也不支持 mvc core 的日志 api，还是自己的一套做法。可以用 https://github.com/RoamingLost/Chimera.Extensions.Logging.Log4Net 来适配.net core 的风格。

ASP.Net Core 网站部署

ASP.Net Core 底层和 IIS 完全隔离，可以使用 Kestrel 作 SelfHost，也可以 Host 其他 Web 服务器上。Kestrel 没有安全、缓存等功能，因此不应该把 Kestrel 暴露到公网。

经典的模式是 SelfHost，然后使用 IIS、Nginx 做反向代理服务器把请求转发给 Kestrel。由反向代理服务器进行缓存、安全等处理。

1、 ASP.Net Core 部署到 IIS 上 https://docs.microsoft.com/en-us/aspnet/core/publishing/iis 以在线文档为准

a) 首先正常安装配置 IIS；
b) 确保 IIS 上安装了 AspNetCoreModule 模块（IIS 的【模块】中查找），下载地址：https://go.microsoft.com/fwlink/?linkid=848766 如果是安装 AspNetCoreModule 模块之前新建的网站，则需要删掉重新建网站。
c) 把网站以【文件夹】方式发布，然后拷贝到服务器上
d) IIS 中新建一个网站，选择目录为发布目录
e) 设置网站的连接池的.Net CLR 版本为【无托管代码】，因为.Net Core 是以反向代理的方式部署的。
f) 部署到 IIS 上的时候，如果有用户正在访问，这时候无法覆盖 dll，因为 dotnet 正在使用 dll。需要关闭网站稍等 dotnet 进程退出后再覆盖 dll。
g) 如果运行有权限问题，就配置 EveryOne 用户有对网站文件夹全部权限就可以。

2、 ASP.Net Core 部署到 Linux 上（Nginx）

文档：https://docs.microsoft.com/en-us/aspnet/core/publishing/linuxproduction 以在线文档为准

1) Startup 的 Configure 中配置转发中间件

```
app.UseForwardedHeaders(new ForwardedHeadersOptions
{
    ForwardedHeaders = ForwardedHeaders.XForwardedFor | ForwardedHeaders.XForwardedProto
});
```

2) 安装 nginx 服务器：sudo apt-get install nginx

3) 启动 nginx 服务器：sudo service nginx start

4) sudo 运行编辑/etc/nginx/sites-available/default 文件，编辑成如下的配置。

```
server {
    listen 80;
    location / {
        proxy_pass http://localhost:5000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection keep-alive;
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

5) 执行以下命令，使 Nginx 配置生效：sudo nginx -s reload

6) dotnet aaa.dll 启动服务器，测试一下

7) 配置自动启动 dotnet aaa.dll，相当于 windows 的“服务”

Create the service definition file

```bash
sudo nano /etc/systemd/system/kestrel-hellomvc.service
```

An example service file for our application.

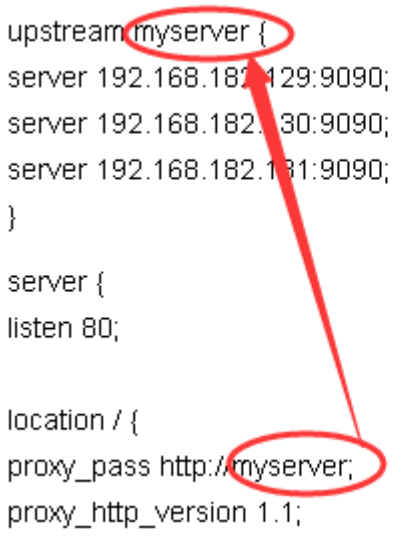
```ini
[Unit]
Description=Example .NET Web API Application running on Ubuntu

[Service]
WorkingDirectory=/var/aspnetcore/hellomvc
ExecStart=/usr/bin/dotnet /var/aspnetcore/hellomvc/hellomvc.dll
Restart=always
RestartSec=10  # Restart service after 10 seconds if dotnet service crashes
SyslogIdentifier=dotnet-example
User=www-data
Environment=ASPNETCORE_ENVIRONMENT=Production

[Install]
WantedBy=multi-user.target
```

```
systemctl enable kestrel-hellomvc.service
systemctl start kestrel-hellomvc.service
systemctl status kestrel-hellomvc.service
```

Nginx 复杂均衡的配置



nginx 负载均衡的策略 http://www.cnblogs.com/andashu/p/6377323.html

做一个登录注册的功能作为小案例，带图形验证码