net core 官方文档地址 <https://docs.microsoft.com/en-us/aspnet/core>

安装net core

curl https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > microsoft.gpg

sudo mv microsoft.gpg /etc/apt/trusted.gpg.d/microsoft.gpg

sudo sh -c 'echo "deb [arch=amd64] https://packages.microsoft.com/repos/microsoft-ubuntu-xenial-prod xenial main" > /etc/apt/sources.list.d/dotnetdev.list'

sudo apt-get install apt-transport-https

sudo apt-get update

sudo apt-get install dotnet-sdk-2.1.105

apt-get install aspnetcore-store-2.0.0

linux 安装mysql

sudo apt-get install mysql-server

suto apt-get install mysql-client

mysql –u账号 –p密码 连接mysql数据库

show databases; 查看有哪些数据库 quit 退出MySql客户端

mysql默认只能本机连接,如果想通过其他电脑连接,需要配置

1. 编辑mysql配置文件 mysqld.cnf,由于mysql是系统服务,因此需要以sudo运行vi才有权限,执行 sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf. 命令 status 查看MySql版本信息
2. 把 bind-address=127.0.0.1注释掉 ,使用#号注释.
3. 执行 sudo /etc/init.d/mysql restart 重启mysql 服务
4. 执行ifconfig 查看主机ip,执行 grant all privileges on \*.\* to root @"192.168.0.172" identified by "Lm160713" with grant option; 再执行 flush privileges;

linux安装 OpenSSH

Ubuntu默认只安装了客户端

安装服务器:sudo apt-get install openssh-server

查看服务器是否开启:netstat –tlp

启动sudo /etc/init.d/ssh start 重启 restart

ssh服务默认端口22,可通过修改配置文件修改端口 /etc/ssh/sshd\_config

linux 远程软件 putty

linux 安装ftp服务:

ftp上传工具 FileZilla

sudo apt-get install vsftpd

默认没有上传权限,需要”sudo vi /etc/vsftpd.conf” 将 “wirte\_enable=YES”前面的#取消,然后 重启服务 sudo /etc/init.d/vsftpd restart

dotnet core 反编译软件 JustDecompile

dotnet core 如果使用反射: Using System.Reflection

dotnet new console/classlib(类库)/webapi/sln(解决方案) –o myConsole

dotnet restore

dotnet run

dotnetcore的mvc 项目的静态文件必须放到 wwwroot下面

修改默认绑定端口的方法:在 program.cs的build之前加入.UseUrls(“http://\*:5001”);

解决方案添加项目

sudo dotnet sln web.sln add Common/Common.csproj

项目之间添加引用

sudo dotnet add MyMvc/MyMvc.csproj reference Common/Common.csproj

添加第三方引用

dotnet add package Newtonsoft.json

shell*脚本*

mkdir $1

cd $1

dotnet new mvc –o $1.web

dotnet new classlib –o $1.model

dotnet new classlib –o $1.Bll

dotnet new classlib –o $1.Dal

dotnet new sln

dotnet sln $1.sln add $1.web/$1.web.csproj

…..

….

dotnet add $1.web/$1.web.csproj reference $1.model/$1.model.csproj

….

dotnet restore

保存成 sc.sh

调用: bash ./sc.sh Test11

dotnet core 读取配置文件 appsettings.json

1. 添加 Microsoft.Extensions.Configuration和Microsoft.Extensions.Configuration.Json
2. Using Microsoft.Extensions.Configuration
3. IConfigurationBuilder builder = new ConfigurationBuilder().SetBasePath(Directory.GetCurrentDirectory()).AddJsonFile("appsettings.json");

var configRoot = builder.Build();

var value = configRoot.GetSection("Logging").GetSection("LogLevel").GetSection("Default").Value;

使用mysql efcore

1. 创建数据库
2. install-package pomelo.entityframeworkcore.mysql
3. 编写和数据表对应的类
4. 编写自己的Dbcontext,需要引用命名空间using Microsoft.EntityFrameworkCore;

EfDbContext.cs

using Microsoft.EntityFrameworkCore;

using System.Reflection;

namespace Model

{

public class EfDbContext : DbContext

{

public DbSet<UserInfo> UserInfos { get; set; }

public DbSet<Role> Roles { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

//需要引用nugut包 install-package RuPeng.EFCore.Ext

modelBuilder.ApplyConfigurationsFromAssembly(Assembly.Load("Model"));

}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

base.OnConfiguring(optionsBuilder);

optionsBuilder.UseMySql("Server=192.168.0.172;database=mynetcore;uid=root;pwd=Lm160713");

}

}

}

UserConfig

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Model.Config

{

class UserConfig : BaseConfig,IEntityTypeConfiguration<UserInfo>

{

public void Configure(EntityTypeBuilder<UserInfo> builder)

{

builder.ToTable("T\_UserInfos");

//builder.Property(i => i.Age).IsRequired().HasMaxLength(3);

//执行sql时候过滤掉roleid=0的

//builder.HasQueryFilter(i => i.RoleId==0);

builder.HasOne(e=>e.Role).WithMany().HasForeignKey(i=>i.RoleId).IsRequired();

}

}

}

获取完整路径:mvc core没有server.mappath

1. 引用命名空间Microsoft.AspNetCore.Hosting
2. 在构造函数注入引用, IHostingEnvironment是框架自动注入的ioc容器的

public HomeController(TestService testService,IHostingEnvironment hostingEnvironment) {

this.\_testService = testService;

this.\_hostingEnvironment = hostingEnvironment;

}

3. //获取本地路径

var rootPath=\_hostingEnvironment.ContentRootPath;

//获取wwwroot静态文件的路径

var wwwPath = \_hostingEnvironment.WebRootPath;

//拼接获取对应文件的路径

var sitePath = Path.Combine(wwwPath,"js/site.js");

设置开发模式和生产模式

Program.cs 的main方法中 .UseEnvironment(“Development”)为开发模式

.UseEnvironment(“Production”)为生产环境

//是否为开发环境

var isDev = \_hostingEnvironment.IsDevelopment();

缓存的使用

1.在构造函数注入IMemoryCache

\_memoryCache.Set<UserInfo>("LoginUser",user,TimeSpan.FromSeconds(1000));

var user = \_memoryCache.Get<UserInfo>("LoginUser");

session使用

1. nuget安装Microsoft.AspNetCore.Session
2. ConfigureServices中services.AddSession();
3. Configure中app.UseSession();
4. TempData依赖于Session,所以也要配置Session.
5. HttpContext.Session,但是原始只有Void Set(string key, byte [] value) 和bool TryGetValue(string key,out byte[] value)这两个方法.如果using Microsoft.AspNetCore.Http;(需安装 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.
6. 推荐使用redis做进程外的session;

MVC Core 中的return Json( 对象) 会自动处理属性首字母小写

字符串html编码和url编码

HtmlEncoder.Default.Encode("<input type='button' value ='测试' />")

UrlEncoder.Default.Encode("http://www.no5.com.cn")

视图文件默认引用的类在 \_ViewImports.cshtml中配置

日志: NLog Https://github.com/Nlog/NLog.web/wiki

1. 声明 ILogger
2. 构造函数中注入ILoggerFactory类
3. this.\_logger = loggerFactory.CreateLogger(typeof (HomeController));

\_logger.LogDebug("这是一个调试信息");

\_logger.LogError("这是一个错误信息");

\_logger.LogWarning("警告");

图形化文本编辑器 emacs

Nginx 运行asp.net core

1. 安装install-package microsoft.aspnetcore.httpoverrides

startup 的configure 添加 app.UseForwardedHeaders(new ForwardedHeadersOptions {

ForwardedHeaders=Microsoft.AspNetCore.HttpOverrides.ForwardedHeaders.XForwardedFor|Microsoft.AspNetCore.HttpOverrides.ForwardedHeaders.XForwardedProto

});

1. 安装nginx sudo apt-get install nginx
2. 启动nginx sudo service nginx start
3. 配置nginx /etc/nginx/sites-available/default

server {

listen 80;

server\_name example.com \*.example.com;

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 $http\_host;

proxy\_cache\_bypass $http\_upgrade;

}

}

1. 执行 sudo nginx –t 验证配置文件是否正确
2. 执行 sudo nginx –s reload 使用修改后的配置文件
3. 把启动dotnet core程序注册成系统服务
4. sudo emacs /etc/systemd/system/MyNetCore.service
5. [Unit]

Description=Example .NET Web API App running on Ubuntu

[Service]

WorkingDirectory=/var/aspnetcore/hellomvc –项目工作路径

ExecStart=/usr/bin/dotnet /var/aspnetcore/hellomvc/hellomvc.dll --启动文件的完整路径

Restart=always

RestartSec=10 # Restart service after 10 seconds if dotnet service crashes

SyslogIdentifier=dotnet-example

User=limeng --用户名

Environment=ASPNETCORE\_ENVIRONMENT=Production

Environment=DOTNET\_PRINT\_TELEMETRY\_MESSAGE=false

[Install]

WantedBy=multi-user.target

1. systemctl enable MyNetCore.service
2. 重启Nginx服务 :sudo service nginx restart
3. systemctl start MyNetCore.service 启动MyNetCore服务
4. systemctl status MyNetCore.service 查看服务状态

Docker官网:https://hub.docker.com

安装docker

sudo apt-get install docker.io

sudo docker version 查看版本 sudo docker info 查看docker的状态信息

下载busybox的镜像,一个简单的linux发行版,执行sudo docker pull busybox

sudo docker run –it busybox

把当前的用户加入到root组

sudo groupadd docker 确保创建了docker用户组,应该是显示”docker用户组已存在”

sudo gpasswd –a 当前用户名 docker :把当前用户名加入docker组

sudo service docker restart 重启docker服务

注销:如果是服务器版执行 logout;桌面版点击右上角的注销

重新登录

docker 常用指令

docker ps 查看运行中的容器 –a 查看所有包含已停止的 –q只列出容器id

docker rm $(docker ps –a -q)删除所有容器

docker run –d 镜像名 方式在后台运行容器,返回的id就是容器的id

docker run -it --name test1 busybox 启动名称为test1的容器

docker start 容器名称/容器id 启动已存在的容器 stop停止容器

docker run –d –-name test123 busybox 启动一个后台运行的容器

docker attach TestDocker 进入TestDocker 容器,进入前必须先启动容器

docker images 列出所有镜像

sudo docker rmi 镜像的imageid 删除指定的镜像

安装ubuntu 镜像 sudo docker run –it Ubuntu

1. 安装微软内置dotnet core的镜像: sudo docker run –it microsoft/dotnet
2. 更新 sudo apt-get update
3. 在启动容器的时候执行: docker run –p ip:hostProt:containerPort 镜像名 这样就把主机的hostPort端口映射到容器的containerPort上,也就是外部用户通过hostPort端口可以访问容器中监听containerPort端口的程序.如果省略ip则表示绑定主机的所有网卡,一定要确保主机的端口没有被占用.还要设置UseUrls才可以
4. 执行如下的 docker run –it –p :81:5000 microsoft/dotnet 进行端口映射,然后进入容器命令行.
5. 可以通过docker ps –a查看端口映射信息
6. Microsoft/dotnet这个景象内置 apt-get ,如果需要安装vim就执行 sudo apt-get vim

制作docker镜像

1. 编写一个文件Dockerfile
2. 写入指令:

FROM ubuntu

RUN apt-get update

RuN apt-get install –y nginx

EXPOSE 80 //容器中的服务”可以”暴露80端口,后面run时候再绑定一下

1. 执行 docker build –t limeng/nginx . 然后就可以:docker run –it limeng/nginx这样 使用这个容器了(没有提交到DockerHub时候只能本机使用,也可以搭建自己的DockerBub私服)

测试端口是否通 wget <http://127.0.0.1:80> ,需安装 wget:apt-get install wget

制作一个包含网站的镜像

quartz.net定时任务框架

install-package quartz

try

{

//从工厂中获取调度程序实例

NameValueCollection props = new NameValueCollection() { { "quartz.serializer.type", "binary" } };

StdSchedulerFactory factory = new StdSchedulerFactory(props);

IScheduler scheduler = await factory.GetScheduler();

//开启调度器

await scheduler.Start();

//定义这个工作,并将其绑定到我们的IJob实现类

IJobDetail job = JobBuilder.Create<MyJob>().WithIdentity("job1", "group1").Build();

//触发作业立即运行,然后每十秒重复一次,无限循环

ITrigger trigger = TriggerBuilder.Create().WithIdentity("trigger1", "group1").StartNow().WithSimpleSchedule(x => x.WithIntervalInSeconds(10).RepeatForever()).Build();

//告诉Quartz使用我们的触发器来安排作业

await scheduler.ScheduleJob(job, trigger);

//等待60秒

await Task.Delay(TimeSpan.FromSeconds(60));

//关闭调度程序

await scheduler.Shutdown();

}

catch (Exception ex)

{

await Console.Error.WriteLineAsync(ex.ToString());

}