**c#知识复习**

C#是我真正学习第一个编程语言，当时被传智播客的一个老师讲课方式所吸引了，很幽默，然后一边看视频一边做题，学得很认真，学了有半年时间，后面也写了几个系统，有c/s的，也有b/s，掌握了三层结构，mvc，一个系统还用上了ef，应该说当时基础还是满不错的。

但因为种种原因，电脑的配置比较低、运行个VS都卡卡的，还有免费虚拟机只支持PHP，还有一个就是.net我感觉它太会封装了，把一些复杂的东西都封装成点几下鼠标就搞定或者写几句代码就解决的事情，例如EF，我写了一个系统后，我发现我sql语言都不会写了。

我就想学一门脚本语言，开放一点的，不用编译的，就转学了php。

但学了php后，用了thinkphp框架后，我发现我写的代码都跟一坨屎一样，仅仅只是能实现功能而已，什么分层都没用上，可能是php太自由了，我太懒惰了。

现在终于发现.net的好，它实现的都是我现在想要改变的。有visual studio的约束下，写起代码也会漂亮点的。

不过隔了差不多3年了，.net的知识都快忘光了，现在重新捡起c#方面的知识，写下笔记，希望以后不会再忘了。

**using的作用**

using 命名空间 //引用命名空间

using mySpace = System; //可以用别名代替命名空间或者类名

namespace 命名空间 //定义命名空间

class 类 //定义类

static void 方法 //定义一个静态方法

cons double pi = 3.14; //定义一个常量，定义时必须得赋值，赋值后就不能修改了

命名都是用骆驼命名法，高级点的像命名空间、类，每个单词首字母大母，其余小写，例如 MyClass，像函数、变量就首单词字母小写，其余单词首字母大写，例如 myHome

**@的作用**

1、定义一个很长的字符串时可以不用换行

string str1 = “一个很长很长很长很长很

长很长的

字符串“;

2、可以忽略字符串里同的转义符 Console.WriteLine(@“这是一个\符号”);

3、可以定义关键字变量字，例如class

string @class = “一个关键字的变量名”

**数据类型**

值类型是存在在栈中的，引用类型是把真实内容存在堆中，变量及内在地址存放在栈

**值类型**

整数值 int

浮点数值 float double

布尔值 bool

字符值 char

**引用类型**

字符串 string

对象 object

需要注意的是string是引用类型，如果string str1=”hello”; string str2 = str1;这样的话，str1跟str2是指向同一个字符串，但因为字符串是不可能修改的，所以就算我把str1=”hello2”也不会改变str2的值；

**唯一一个三元运算符**

?””:””

d=a?b:c;

**数学运行符**

% 取余数

int a=5;

int b;

b = ++a;先a加1，再赋值给b

b=a++;先赋值给b，a再加1

**布尔运算符**

^ var1^var2，只有一个为true时，为true

**位运算符**

~ 位取反

& 与

| 或

^ 异或

<< 左移

>> 右移

其他运算符

is 判断类型

bool isStr = a is string;

**分支语句**

if(){

}else if (){

}else{

}

switch(表达式){

case 表达式:

语句;

break;

case :

case:

语句;

break;

default:

语句;

break;

}

**迭代语句**

while(){}

do{

}while();

for(int I;i<=10;i++){}

foreach( in 集合){}

**字符串比较**

bool b = string.Equals();

int a = string.compare()

字符串格式化

string.Format()

**一维数组**

数组定义后长度就不能改变了

string[] strArray = new int[3]{“1”,”2”,”3”};

int[] intArray1 = {1,3,4};

int[] intArray2 = new int[3];

**二维数组**

int[,] intArray= { {1,2},{3,4} }

int[,] intArray2 = new int[5,2];5行2列

Array.IndexOf(数组,8);//在数组里查找元素的索引位置

Array．BinarySearch(数组,8);//二分法查找，速度快

Array.Sort();//排序

Array.Reverse();//倒序

Array.Copy() //数组合并与拆分

ArrayList al =new ArrayList(); //动态数组

al.Add();

al.AddRange();

al.Remove(元素);//删除

al.RemoveAt();//删除

al.RemoveRange();

**方法修饰符**

static public private

**方法参数**

**params** 可指定个数不定的参数

public int Sum(params int[] nums){}

**ref 引用参数**

public int Test(ref int a){}

int b=0;

Test(ref b );

**out** 输出参数 跟ref用法差不多，不过可以传入没有赋值的参数

**in** 只读参数，不能在方法内部修改

virtual 当方法定义为虚方法后，继承该方法的对象后，可以用override重写此方法

abstract 抽象方法只能定义在抽象类，抽象类里面也只有抽象方法，

**协变 out** 用于泛型参数上，T可在接口或委托中用于返回类型，不能用于传入参数类型

将子类的泛型实例赋于声明基类的泛型变量，为了能限制返回一个基类对象

例如 Animal是一个基类，Dog是Animal的派生类

那么可以 I

**逆变 in** T可在接口或委托中用于传入参数类型，不能用于返回类型，为了能限制传入一个子类对象，觉得主要用于委托，用来限制输入参数的类型

将基类的泛型实例赋于声明了子类的泛型变量

**override** 只有基类方法为virtual abstract override修饰时，才能进行重写

重写的意义就是

基类 A=new 子类B();

A.Test();//这还是执行子类B的重写方法，如果没用重写的话，那会执行基类的Test方法

**as** 把变量转换为某个类型，转换失败就赋为null。跟()强制转换区别就是安全

**is** 判断变量是否为某个类型

**dynamic**

**什么是多态**

通过继承重写方法的形式，使同一基类声明的对象，使用同一方法，但有不同的执行效果。

**委托 delegate**　可以理解为方法的接口 Invoke跟直接用()效果是一样的

注意的是加了event的委托不能直接用类调用，只能出现在+=或-=左边

public delegate void TestDelegate();

class Persion()

{

public TestDelegate say;

public event TestDelegate say2;

}

Person p1 = new Person();

TestDelegate td =()=>{};

p1.say += td;

p1.say2+=td;

p1.say();//可以

p1.say();//不可以

**partial 局部类**，可以定义多个名字一样的类，编译时会合成为一个

**new的使用**

实例对象 new Class()

继承时其它对象时，隐藏实例同名方法或属性，不进行重写

public new void Test(){}

限制泛型的类型

where T :new()

**扩展方法**

在静态类中静态方法的参数 中用this 例如

public statis string Test(this string input)

{

return “123”;

}

string a = “456”;

a.Test();

**特性 Attribute** 自定义特性类必须继承于Attribute，可以用typeof(Class1).GetCustomAttributes获取，Attribute.GetCustomAttribute()也能获取

**implicit隐式转换** 可以类中定义一个

class Person

{

public string Name{get;set;}

public static implicit operator Person(string name)

{

return new Person{ Name=name};

}

}

Person p=”123”;//就可以这样定义了

**explicit 显式转换**

跟上面差不多，只不过得用强制转换 Person p = (Person)”123”;

**泛型**

**定义方法**

以<T>，T自定义字母

可以定义在类名后面，如下

class Person<T>{

}

也可以定义在方法名后面，如下

public T getData<T>(T params){}

定义后，就可以用这个自定义字母T去限制参数类型、返回值类型了

使用方法

如果是定义在类上的，可以在声明时明确T的真实类型，如下

Person<int> p = new Person<int>();

如果是定义在方法上面的，可以在调用时明确T的真实类型，如下

int a=10;

int b = getData(10);

**?的用法**

?. a?.ToString();//如果a变量为null，则返回null，不运行toString()

string b = a??”hello”;//如果a变量为null，则返回hello，不然返回a自己

a?[1] 如果a变量为null，则返回null，不然返回索引1的值

**Async/await 异步编程**

System.Threading.Tasks.Task.Run(()=>{

//do something

});

传统的Thread线程每次创建都是新线程，线程id是不同的，但Task创建的线程可能是重复的，带来性能上的优化

约定方法名后面都要加上Async

public async Task TestAsync()

{

await Task.Run(()=>{

Thread.sleep(5000);

Console.WriteLine(“hello,task!”);

})

}

**async** 是用来修饰方法的，且此函数返回类型只能是void，Task，Task<T>，其实都是返回一个Task类型的

**await**就是执行Task类型的，只能用在有async修饰符的方法里，用来等待await后面代码执行的返回结果，如果await后面有开启子线程的代码就会立即返回一个Task类型，并将await下面的代码都扔给子线程执行，如果没有，就跟执行同步代码一样，之前以为一遇到await就开启子线程，但经过测试，如果await后面的方法，都没有开启子线程的操作，那就跟执行同步代码一模一样；

**Task.Result** 这也是用来得到子线程的返回结果的，这是阻塞主线程的，抛开线程来说，可以说跟await的效果一样

这个跟await的区别是，这个是阻塞主线路等待执行结果返回才执行下面的代码，而await是有开启子线程的立即返回一个Task类型，不阻塞主线程，在子线程中等待子线程执行结果，再执行await下面的代码，await下面的代码也是在子线程中执行的；

当遇到await关键字之前的代码都是同步执行的，但遇到await时，就会把函数后面的代码都扔到子线程中执行，然后在主线程中立即返回一个Task类型的对象，然后继续执行主线程的代码，等到Task.Run里面的执行结果得出后，会从await继续执行以下代码，然后再返回值，如果主线程中

**我猜的执行过程**

用await 执行一个Task的对象A时（这个Task对象A里面有个IsCompleted，这个bool值），await会检测这个IsCompleted（我猜的，这个IsCompleted值应该是代表Task里面的异步函数是否执行完），但IsCompleted是false时，await 会把await关键字下面的代码进行打包，编译成一个异步函数，再生成一个Task对象B（IsCompleted=false），把异步函数装进去，并在当前函数中返回这个Task对象B。（我猜应该会把对象B注入到对象A中一个属性）

然后每个Task里面的异步函数执行完后，IsComplete变成true，就会触发上级的Task执行异步函数（检测是否有注入上级Task对象），逐层传递，有点像递归

public Task Await(Task T)

{

if(T.IsCompleted)

{

return T.Result;

}

new Task()

{

IsCompleted=false

}

}

例如上面的例子，当Task对象A异步函数执行完后，会触发上级Task对象B执行

**ADO.NET**

1、先创建一个数据库连接对象，一个实现了IDbConnection接口的对象，然后使用open()打开数据连接；

2、再创建一个命令对象，一个实现了IDbCommand接口的对象

3、如果只是需要执行sql无需获取数据，可以用IDbCommand.ExcuteNonQuery()，会返回受到影响的行数；

4、如果需要获取查询结果里首行首列的字段，可以用IDbCommand.ExcuteScalar()，一般用来新增数据返回id

3、如果需要获取数据列表的话，那还要创建一个数据读取对象，一个继承了DbDataAdapter类的对象，在构造函数中传入命令对象，再创建一个DataSet对象，进行数据的填充，DbDataAdapter.Fill(DataSet)

|  |  |  |  |
| --- | --- | --- | --- |
| **数据库** | **IDbConnection实现类** | **IDbCommand实现类** | **DbDataAdapter继承类** |
| Sql Server | SqlConnection | Sql |  |
| Access | OleDbConnection |  |  |
| MySql | MySqlConnection |  | MySqlDataAdapter |

**ef core**

pmc命令

mysql db first

Scaffold-DbContext -Connection "Server=localhost;User Id=root;Password=root;Database=it\_db" -Provider "Pomelo.EntityFrameworkCore.MySql" -OutputDir "Models"

**vo** view object 就是controller传到前端的对象，就像我之前那个result对象一样，把要传给前端的数据再进行封装，再加上成功失败的字段跟描述字段，告诉前端，这个接口传输的状态

**dto** 就是将数据实体对象转化为一个前端可查看对象，其实就是隐藏一些不让前端可看到的字段

**IoC 依赖注入**

比如类A中需要new B，依赖注入就是让类A不用通过new使用B，而是通入构造函数或者属性传入B的对象，将new的功能交给外面处理

比如我controller要new一个bll对象，我可以自己new，但依赖得自己new，然后通过构造函数传进去，那我可以找一个非常专业的第三方框架对象，然后在对象的配置文件中说明依赖项是哪个，然后我让第三方对象给我一个bll对象，那它就会根据依赖对象，生成一个带有依赖对象的bll给我，我就不用在每个controller里面new依赖对象了

AspNet.Core命令行操作

环境：window在cmd窗口下

可以用 dotnet –h 查看帮助 dotnet new –h 查看创建项目的帮助

dotnet new console –o MyConsole

创建项目console是模板 dotnet new –l 查看所有的项目模板

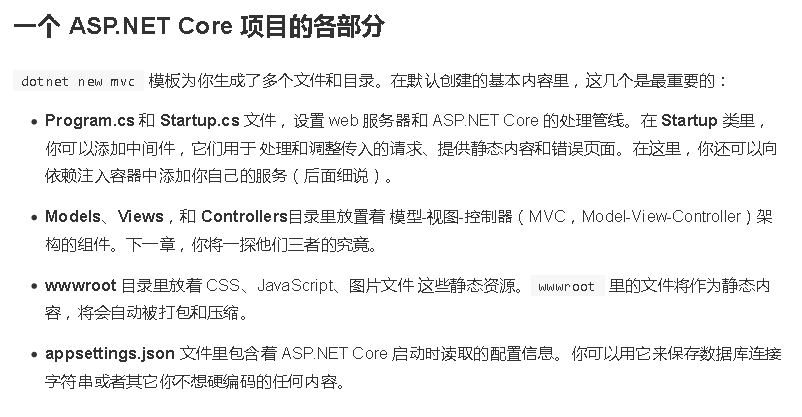
-o MyConsole 指定模板输出位置，就是指定项目放置的文件夹

dotnet add package Humanizer 添加一个依赖包Humanizer

dotnet new sln 创建一个解决方案文件

dotnet sln add 项目目录 将项目添加到解决方案文件中

-o 输出目录 –n 文件名称



**visual studio 2019 操作知识**

**创建项目一个很奇怪的地方**

“将解决方案和项目放在同一目录中”功能的作用

如果是.Core的项目的话，似乎不起什么作用，还是将创建一个项目目录及一个sln文件，勾上的话，只是将sln文件及目录放在位置的根目录，不勾的话，就会在位置上创建一个解决方案名称的目录，然后将sln文件及目录放进去

如果是.Framework的项目，就起作用，勾上的话，会有位置上创建项目目录，然后里面是目录文件及sln文件，不勾上的话，就会在位置上创建一个解决方案目录，然后再sln文件跟项目目录放进去

当创建过一次视图使用Entity Framework的MVC控制器后，如果配置错了，想重新创建一次的话，需将Pages目录删除

**快捷键**

ctrl+c 复制光标当前行，不用全选

ctrl+e,d 格式化代码

ctrl+j 强制进行代码提示

tab快速生成代码，例如for，try代码块

ctrl+k,s 嵌套代码块，以if、class、套起来

ctrl+k,ctrl+c 注释

ctrl+k,ctrl+u 取消注释

F12 转到光标所在的函数、类等等的定义，跟按住ctrl点击一样的效果

ctrl+g 跳转到行号

**Visual Studio Code** 操作知识

shift + alt +f 格式化代码

ctrl+ / 注释

Razor

如果不要使用布局的话

@{

Layout = null;

}

MVC

在Action里面，用ViewData进行传值

IServiceProvider GetRequiredService<T> 　从服务提供者返回某个服务T

加了ApiController 特性后，只能接受json[application/json]了，不支持form-data及x-www-form-urlencoded

[Route(“api/[controller]”)] 加在controller上，指定url前缀

[HttpGet(“getDetails/{id}”)] 定义在action上，指定url后缀

ConfigureServices

services.AddScoped<T,T2>()//每次请求的依赖类都是不同的

services.AddTransient<T,T2>(); //依赖类每次请求都会new一个新对象

services.AddSingleton<T,T2>()//每次请求的依赖类都是使用同一个实例

Configure 中间件，如果某个中间件终止了请求，则称为终端中间件

Use Run Map 配置http管道

public void Configure(IApplicationBuilder app)

{

//当url以mp1开头时

app.Map(“/Map1”,app=>{

app.Run(async(context)=>{

})

})

}