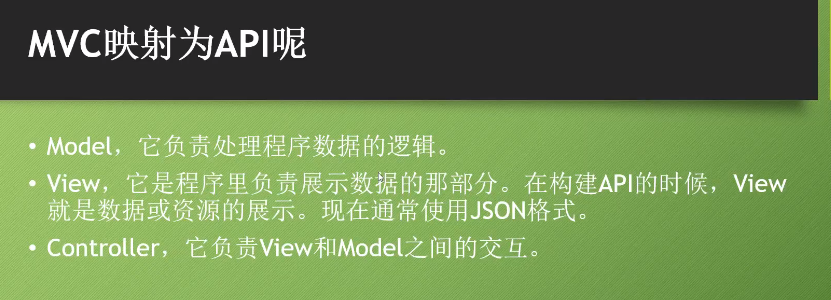
# 基础

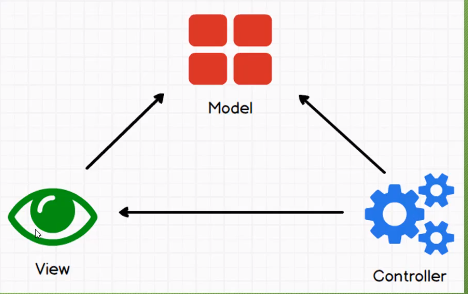
## 简介

### Web api

* 大多数web api并不是RESTful API：【只有符合REST约束规范/架构风格的API才是】
* Web api通常是指“使用HTTP协议并且通过网络调用的api”，由于它使用了HTTP协议，所以需要通过URL信息来指定端点。
* Web api就是一个we系统，通过访问URL可以与其进行信息交互。
* MVC模式最关注的两点:松耦合、关注点分离(separation of concerns)



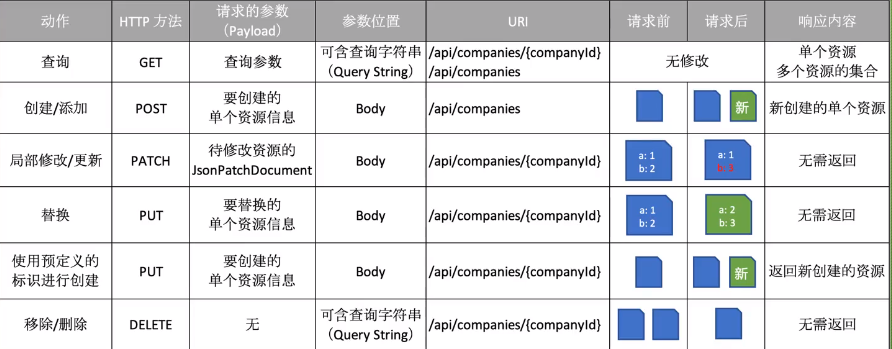
依赖关系



### RESTful Api

* REST，表示性状态转移（representation state transfer）。简单来说，就是用URI表示资源，用HTTP方法(GET, POST, PUT, DELETE)表征对这些资源的操作。
* RESTful 是典型的基于HTTP的协议
* Resource: 资源，即数据，存在互联网上的可被访问的实体
* Representation： 数据的某种表现形式，如HTML, JSON。
* State Transfer：状态变化，HTTP方法实现
* RESTful API 就是REST风格的API。现在终端平台多样，移动、平板、PC等许多媒介向服务端发送请求后，如果不适用RESTful API，需要为每个平台的数据请求定义相应的返回格式，以适应前端显示。但是RESTful API 要求前端以一种预定义的语法格式发送请求，那么服务端就只需要定义一个统一的响应接口，不必像之前那样解析各色各式的请求。
* 关于RESTful api简介【
* 资源。首先要明确资源就是网络上的一个实体，可以是文本、图片、音频、视频。资源总是以一定的格式来表现自己。文本用txt、html；图片用JPG、JPEG等等。而JSON是RESTful API中最常用的资源表现格式。
* 统一接口。对于业务数据的CRUD，RESTful 用HTTP方法与之对应。

|  |  |
| --- | --- |
| CRUD | HTTP Methods |
| Create | POST |
| Read | GET |
| Update | PATCH(1-多个)/PUT(全部) |
| Delete | DELETE |

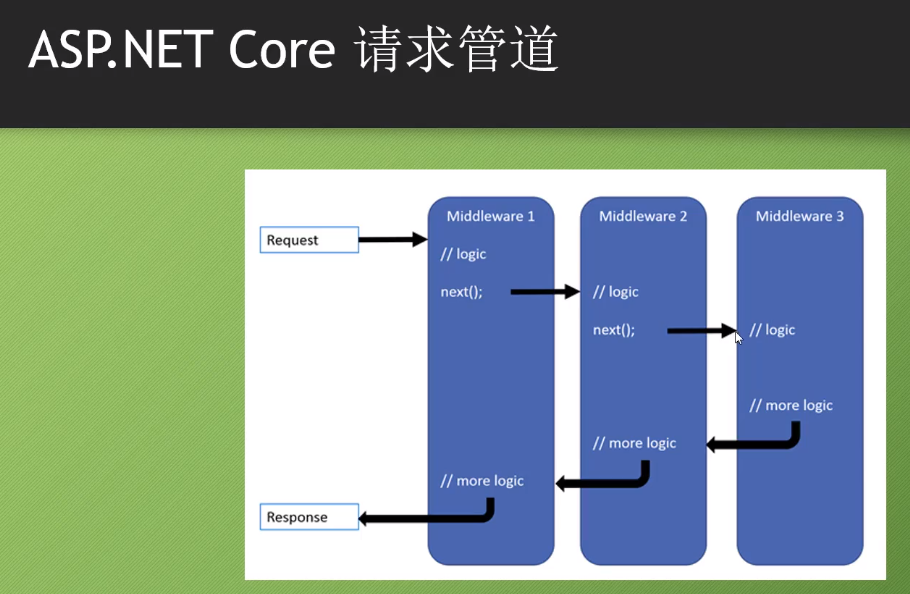


* URI。统一资源标识符，它可以唯一标识一个资源。注意到，URL(统一资源定位符)是一种URI，因为它可以唯一标志资源。但URL != URI。应该说URL 是URI的子集。因为URL使用路径来唯一标识资源，这只是唯一标识资源的一种方式。还可以用一个唯一编号来标识资源，如example.html.fuce2da23。只不过这种方式并不被广泛使用。总之，要在概念上对URL和URI有所区分。
* 无状态。 所谓无状态是指所有资源都可以用URI定位，而且这个定位与其他资源无关，不会因为其他资源的变动而变化。这里引入一个幂等性的概念：无论一个操作被执行一次还是多次，执行后的效果都相同。比如对某资源发送GET请求，如果访问一次和访问十次获得的数据一样，那么就说这个请求具有幂等性。
* URL中只能有名词，不能出现动词。这是因为在REST要求对资源的操作由HTTP 方法给出，而方法是由HTTP 请求报文头部给出的，自然不需要在URL中暴露操作方式。
* 】
* 【
* REST背后的主要思想就是：采用RESTful架构风格进行组织的分布式系统，将在以下几个方面得到改善：
* 性能。REST的通信风格应该是简单并且高效的，采用它的系统性能应该得以提升。
* 组件交互的可扩展性。其实任何分布式系统都允许这种扩展性，而REST所提出的简单交互方式更是如此。
* 组件的可修改性。分布式系统的分布式本质和REST提出的关注点分离，使得组件得以以最小的成本和最低的风险彼此独立的进行修改。
* 可移植性。REST与技术和语言无关，所以使用任何技术都可以实现REST。
* 可靠性。REST所提出的无状态约束允许在系统发生故障后轻松的恢复系统。
* 可视性。REST所提出的无状态约束为所述请求添加了完整的状态。
* 】

## 项目结构简介

### Startup.cs

* ConfigureServices:注册服务，已经注册的服务可以在项目其它地方通过依赖注入使用
* Configure：指定asp.net core web程序 是如何响应每一个http请求=》在这里配置请求的管道【在此处添加中间件】=>中间件的顺序很重要。



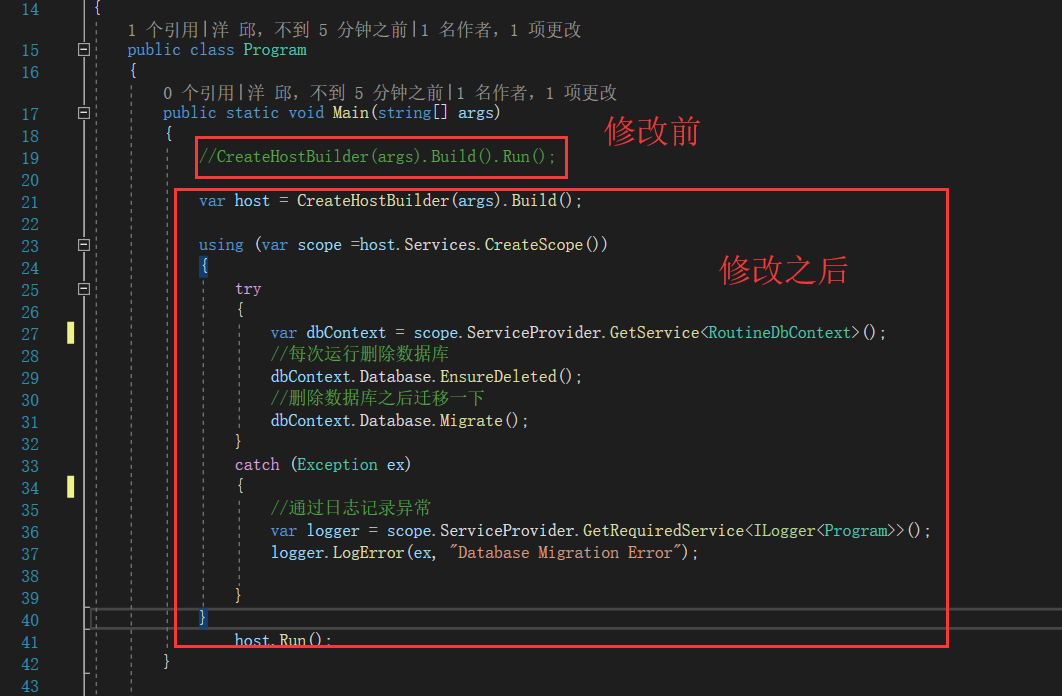
## 关于nuget

* Microsoft.EntityFrameworkCore.Tools：是数据迁移必须使用到的插件

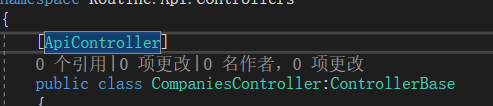
## 关于数据迁移

修改之后，每次只需要执行Add-Migration，不需要执行Update-Database

|  |
| --- |
| public static void Main(string[] args)  {  //CreateHostBuilder(args).Build().Run();  var host = CreateHostBuilder(args).Build();  using (var scope =host.Services.CreateScope())  {  try  {  var dbContext = scope.ServiceProvider.GetService<RoutineDbContext>();  //每次运行删除数据库  dbContext.Database.EnsureDeleted();  //删除数据库之后迁移一下  dbContext.Database.Migrate();  }  catch (Exception ex)  {  //通过日志记录异常  var logger = scope.ServiceProvider.GetRequiredService<ILogger<Program>>();  logger.LogError(ex, "Database Migration Error");    }  }  host.Run();  } |

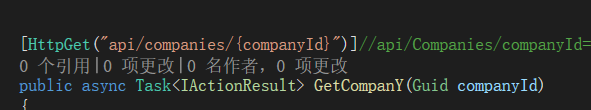


## 控制器

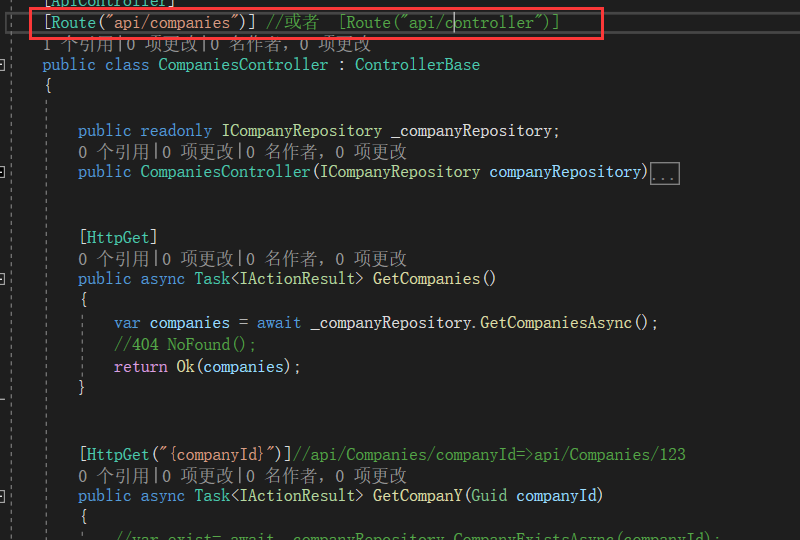




最上面写[Route("api/[controller]")]

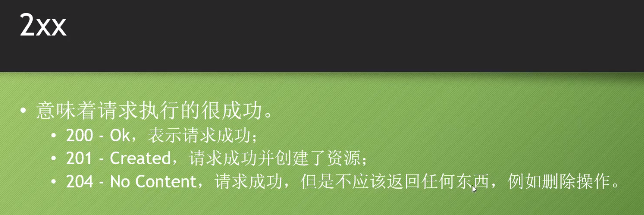


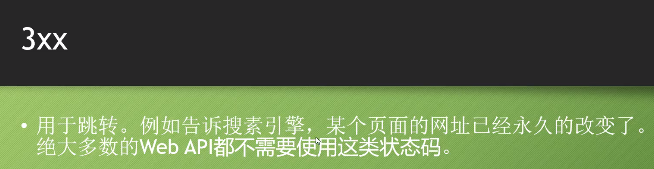
或者

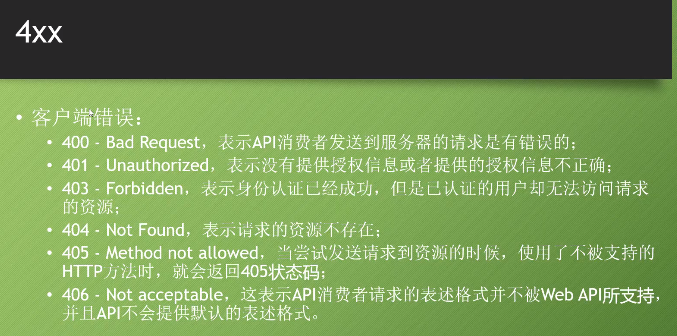


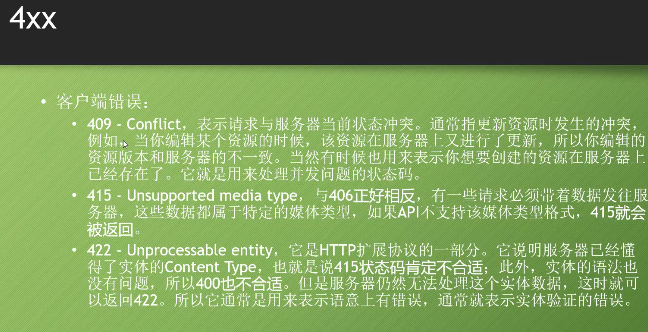
## http状态码

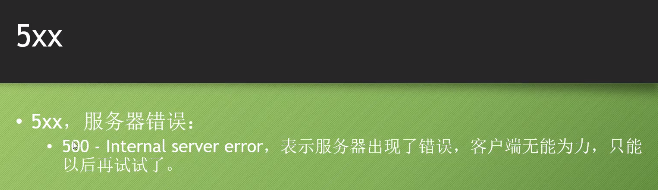
请求是否执行成功、如果请求失败，那么谁为它负责；



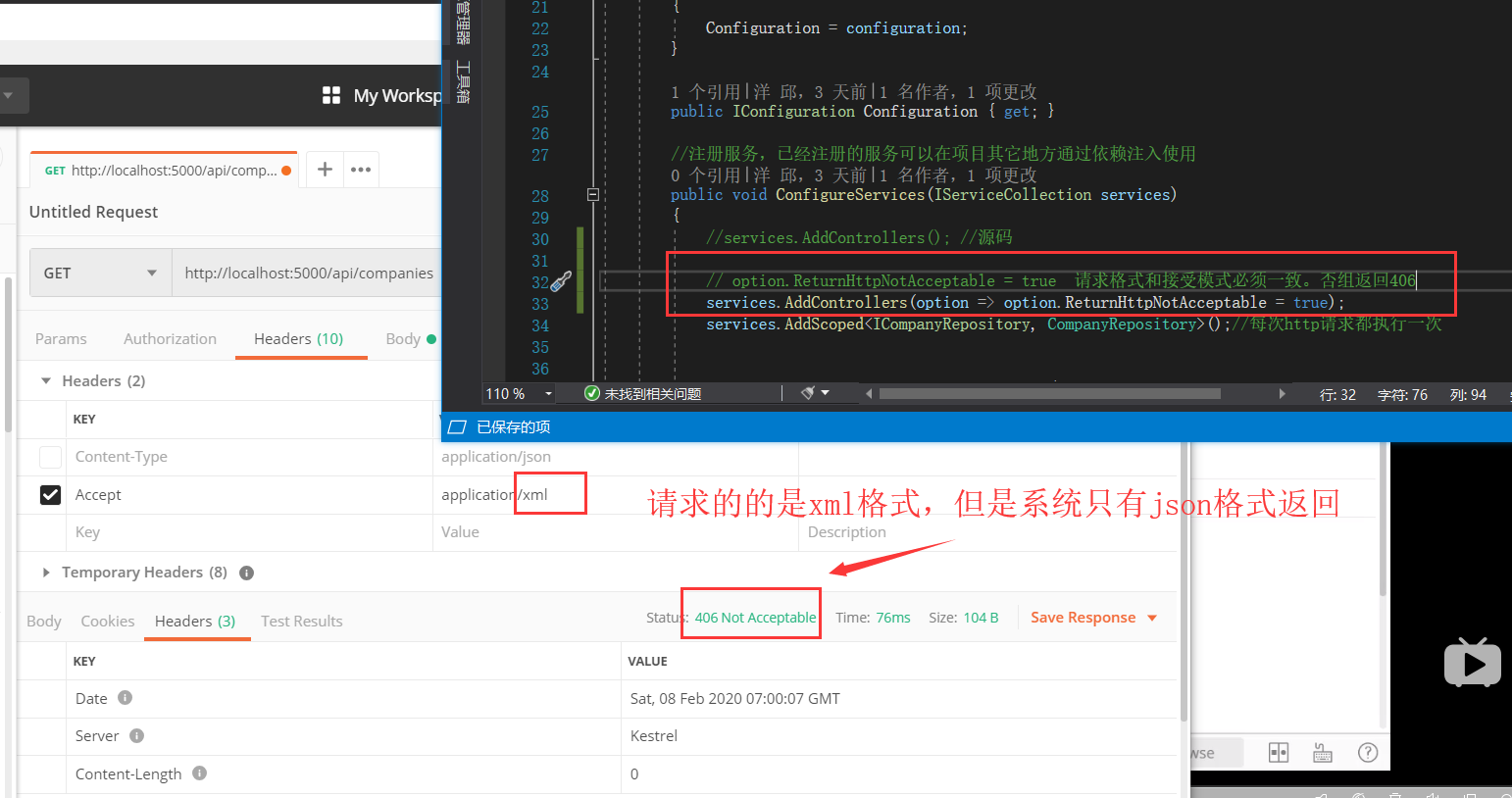






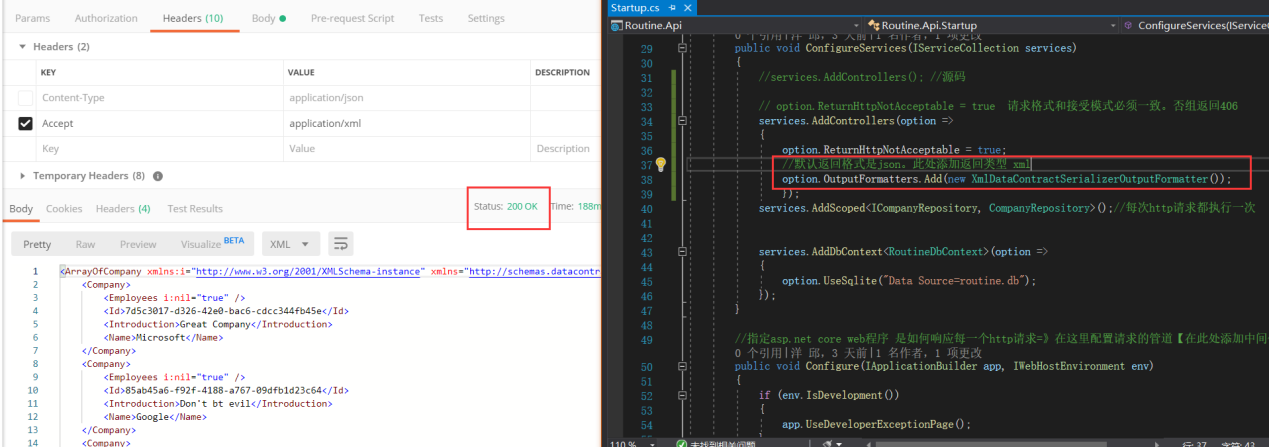


## 请求格式和返回格式



添加返回类型

|  |
| --- |
| option.OutputFormatters.Add(new XmlDataContractSerializerOutputFormatter()); |



### .netCore3.0之后新增方法

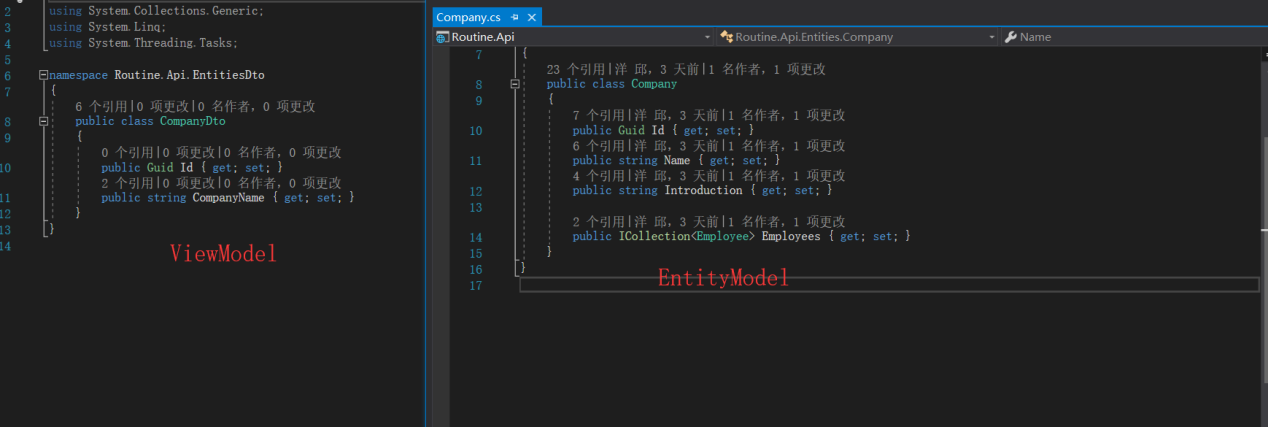
|  |
| --- |
| .AddXmlDataContractSerializerFormatters(); |



## 对象映射器(EntityModel和ViewModel)

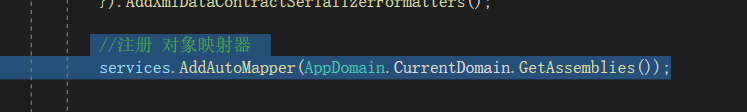
EntityModel是存入数据库的model

ViewModel 是显示到视图上面的数据



使用之前需要在nuget中引用【 AutoMapper.Extensions.Microsoft.DependencyInjection】，然后再startup.cs的ConfigureServices中注册

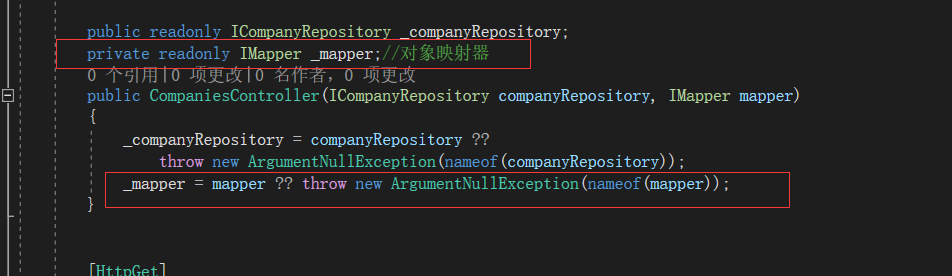
|  |
| --- |
| //注册 对象映射器  services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies()); |



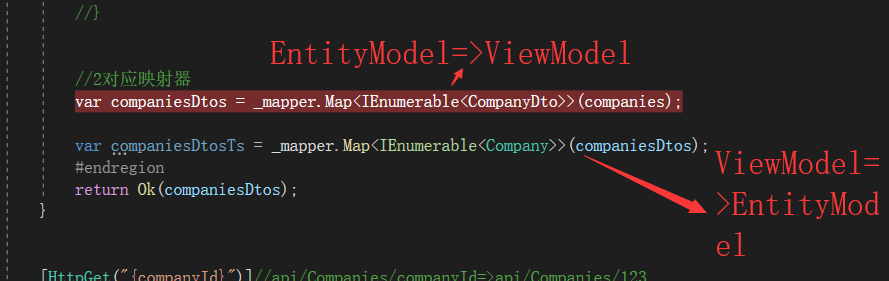
。然后创建相对应的类



在控制器中使用之前先要依赖注入



然后使用



# 项目

## Api格式

### 格式1

<http://localhost:5000/api/companies/{companyId}/employess> 例如

http://localhost:5000/api/companies/7d5c3017-d326-42e0-bac6-cdcc344fb45e/employess



### 注意

如果输入的值不存在。就连调试也不会进入此方法

