.NET Core笔试题

.NET Core笔试题

1.如何在ASP.NET Core中激活Session功能?
2.什么是中间件?
3.Applicationbuilder的Use和Run方法有什么区别?
4.如何使taghelper在元素这一层上失效?
5.什么是ASP.NET Core?
6.ASP.NET Core 中AOP的支持有哪些?
7.ASP.NET Core Filter的注册方式有哪些?
8.ASP.NET Core Filter如何支持依赖注入?
9.ASP.NET Core 如何和读取配置文件中的内容?
10.ASP.NET Core有哪些好的功能?
11.ASP.NET Core跟ASP.NET比较有哪些更好的地方?
12.什么是meta packages?
13.ASP.NET Core应用能够跟ASP.NET4.x架构一起工作吗?
14.什么是ASP.NET Core的startup 类?
15.startup 类的configservice方法有什么作用?
16.startup 类的configure方法有什么作用?
17.ASP.NET Core管道里面的map拓展有什么作用?
18.ASP.NET Core里面的路径是如何处理的?
19.ASP.NET Core工程里面有多少个工程文件?
20.什么是ASP.NET Core里面的taghelper
21.说说.NET5中 _ViewImports文件的作用。
22.什么是Razor页面?
23.说说.NET5中 __ViewStart文件的作用
24.如何在Razor页面中实现数据模型绑定?
25.如何在Controller中注入service?
26.描述一下依赖注入后的服务生命周期?
27.说说ASP.NET Core内置容器的特点;
28.ASP.NET Core中如何读取静态文件?
29.ASP.NET Core项目如何设置IP地址和端口号?
30.ASP.NET Core项目中，wwwroot文件夹内包含什么内容?
31.谈谈对ASP.NET Core kestrel的理解。
32.谈谈对Autofac的理解;
33.ASP.NET Core 如何支持Log4Net扩展?
34.说说脚本启动ASP.NET Core Web项目
35.说说Core WebApi的Swagger。
36.说说Core WebApi特性路由。
37.说说RESTful是什么。
38.说说脚本在请求Web CoreApi的时候，为什么会发生跨域问题?
39.如何解决跨域问题?
40.说说你了解到的鉴权授权技术。
41.请问对gRPC有了解吗，说说gRPC。
42.gRPC有几种模式?
43.说说如何使用C#实现简单模式gRPC
44.说说gRPC的拦截器有哪些?
45.gPRC作为一种被调用的服务，有什么保护安全的措施吗?
46.请问对EFCore有了解吗?
47.说说EFCore查询的性能调优小技巧。
48.EFCore 如果通过数据生成实体和DbContext?
49.说说对SaveChanges的理解。
51.说说对EFCore中EntityState的理解。
52.说说什么是导航属性和引用属性。

1.如何在ASP.NET Core中激活Session功能?

首先要添加session包. 其次要在configservice方法里面添加session。然后又在configure方法里面调用usesession。。

2.什么是中间件?

中间件在这里是指注入到应用中处理请求和响应的组件。是通过多个委托来嵌套形成的一个俄罗斯套娃!

3.Applicationbuilder的Use和Run方法有什么区别?

这两个方法都在startup 类的configure方法里面调用。都是用来向应用请求管道里面添加中间件的。Use方法可以调用下一个中间件的添加，而run不会。run是终结式的;

4.如何使taghelper在元素这一层上失效?

使用叹号。

5.什么是ASP.NET Core?

首先ASP.NET Core可以说是 ASP.NET的升级版本。它遵循了.NET的标准架构，是一个基于.NET Core的Web开发框架， 可以运行于多个操作系统上。它更快，更容易配置，更加模块化，可扩展性更强。

6.ASP.NET Core 中AOP的支持有哪些?

通过Filter来支持；分别有IResourceFilter AuthorizeFilter ActionFilter ExceptionFilter ResultFilter，Filter也被称为拦截器!

7.ASP.NET Core Filter的注册方式有哪些?

方法注册：只对方法生效

控制器注册：对控制器中的所有方法生效

全局注册：对整个项目生效;

8.ASP.NET Core Filter如何支持依赖注入?

可以通过全局注册，支持依赖注入

通过TypeFilter(typeof(Filter)) 标记在方法，标记在控制器

通过ServiceType(typeof(Filter))标记在方法，标记在控制器，必须要注册Filter这类;

TypeFilter和ServiceType的本质是实现了一个IFilterFactory接口;

9.ASP.NET Core 如何和读取配置文件中的内容?

可以有两种方式，可以通过IConfiguration接口来读取;

有可以定义根据配置文件结构一致的实体对象，来绑定到对象中去；或者通过1写入，2注入读取

必须保证：DBConnectionOption和配置文件的内容结构一致;

```
1. services.Configure<DBConnectionOption>
(Configuration.GetSection("ConnectionStrings"));//注入多个链接
```

```
2.private DBConnectionOption dBConnections = null;

        private DbContext _Context = null;

        public DbContextFactory(DbContext context, IOptions<DBConnectionOption>
options)
        {
            _Context = context;
            dBConnections = options.Value;
        }
```

10.ASP.NET Core有哪些好的功能?

第一是依赖注入。
第二是日志系统架构。
第三是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。
第四是可以使用命令行创建应用。
第五是使用appsettings来配置工程。
第六是使用startup来注册服务。
第七是更好的支持异步编程。
第八是支持web socket和signal IR。
第九是对于跨网站的请求的预防和保护机制。

11.ASP.NET Core跟ASP.NET比较有哪些更好的地方?

第一是跨平台，它可以运行在三大操作系统上面，windows， Linux和MAC。
第二是对架构本身安装没有依赖，因为所有的依赖都跟程序本身在一起。
第三是ASP.NET Core处理请求的效率更高，能够处理更多的请求。
第四是ASP.NET Core有更多的安装配置方法。

12.什么是meta packages?

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

13.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。

14.什么是ASP.NET Core的startup 类?

startup 类是ASP.NET Core应用的入口。所有的ASP.NET Core应用必须有这个类。这个类用来配置应用。这个类的调用是在program main函数里面进行配置的。类的名字可以自己定义。

15.startup 类的configservice方法有什么作用?

在这个方法里我们可以添加一些service进入依赖注入容器。

16.startup 类的configure方法有什么作用?

这个方法来定义整个应用如何响应HTTP请求。它有几个比较重要的参数，applicationbuilder，Hosting environment, logfactory， 在这里我们可以配置一些中间件用来处理路径，验证和session等等。

17.ASP.NET Core管道里面的map拓展有什么作用?

可以针对不同的路径添加不同的中间件。

18.ASP.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。

19.ASP.NET Core工程里面有多少个工程文件?

launchsetting，appsettings，Program, Startup

20.什么是ASP.NET Core里面的taghelper

Taghelper用来在服务器端使用Razor视图引擎创建html元素的。

21.说说.NET5中 _ViewImports文件的作用。

在.NET5中可以支持组件化编程，定义的各种组件，在项目中使用的时候，需要在_ViewImports文件中引入进来。

22.什么是Razor页面?

是ASP.NET Core中支持ASP网页表格的一种开发模型。@page 作为页面的起始标志。。
Stringbulider的使用，最好制定合适的容量值，否则优于默认值容量不足而频繁的进行内存分

23.说说.NET5中 __ViewStart文件的作用

在控制器在返回视图的时候，开始替换视图引擎的时候，从_ViewStart.cshtml 开始，来初始化展示的视图界面；

24.如何在Razor页面中实现数据模型绑定?

使用bindproperty属性。

25.如何在Controller中注入service?

在config services方法中配置这个service。
在controller的构造函数中，添加这个依赖注入。

26.描述一下依赖注入后的服务生命周期?

在ASP.NET Core中，我们不需要关心如何释放这些服务, 因为系统会帮我们释放掉。有三种服务的生命周期。
单实例服务， 通过add singleton方法来添加。在注册时即创建服务, 在随后的请求中都使用这一个服务。
短暂服务, 通过add transient方法来添加。是一种轻量级的服务，用于无状态服务的操作。
作用域服务，一个新的请求会创建一个服务实例。使用add scoped方法来添加。

27.说说ASP.NET Core内置容器的特点;

ASP.NET Core内置容器IServiceCollection，只支持构造函数注入；支持三种声明周期：单例、瞬时、Scoped三种声明周期管理;

28.ASP.NET Core中如何读取静态文件?

可以通过中间件UseStaticFiles来配置读取静态文件;

29.ASP.NET Core项目如何设置IP地址和端口号?

可以使用Properties文件夹下的launchSettings配置文件来配置不同的启动方式的时候，分别配置IP和端口号。

30.ASP.NET Core项目中，wwwroot文件夹内包含什么内容?

包含了css、js、js库、字体文件

31.谈谈对ASP.NET Core kestrel的理解。

Kestrel 是一个跨平台的适用于 ASP.NET Core 的 Web 服务器。 Kestrel 是 Web 服务器，默认包括在 ASP.NET Core 项目模板中。

Kestrel 支持以下方案：

- HTTPS
- 28
- 用于启用 WebSocket 的不透明升级
- 用于获得 Nginx 高性能的 Unix 套接字
- HTTP/2（除 macOS† 以外）

macOS 的未来版本将支持 †HTTP/2。

.NET Core 支持的所有平台和版本均支持 Kestrel。

32.谈谈对Autofac的理解;

Autofac是一个IOC容器，支持三种类型的DI依赖注入，配置文件配置映射关系，支持AOP扩展定制;

在ASP.NET Core的使用步骤如下：

1.Nuget引入Autofac程序集

2.在Program类中的CreateHostBuilder方法中，通过.UseServiceProviderFactory(new AutofacServiceProviderFactory())替换容器工厂，把容器替换到框架中;

3.在Startup中增加ConfigureContainer方法，用来配置映射关系

```
public void ConfigureContainer(ContainerBuilder builder)
        {

        }
```
使用了Autofac以后，在IServiceCollection中注入的服务，也能生效；因为Autofac是先接受了所有的来自于IServiceCollection的服务映射后，再去读取ConfigureContainer方法中配置的映射；

4.就可以在控制器中配置构造函数注入了

33.ASP.NET Core 如何支持Log4Net扩展?

就是一个日志组件的集成使用，大概分为以下步骤：

1.nuget引入log4net程序集；Microsoft.Extensions.Logging.Log4Net.AspNetCore程序集合

2.增加配置文件，配置文件内容如下

```
<?xml version="1.0" encoding="utf-8"?>
<log4net>
    <!-- Define some output appenders -->
    <appender name="rollingAppender"
type="log4net.Appender.RollingFileAppender">
        <file value="..\log\Customlog.txt" />
        <!--追加日志内容-->
        <appendToFile value="true" />

        <!--防止多线程时不能写Log,官方说线程非安全-->
        <lockingModel type="log4net.Appender.FileAppender+MinimalLock" />

        <!--可以为:Once|Size|Date|Composite-->
        <!--Composite为Size和Date的组合-->
        <rollingStyle value="Composite" />

        <!--当备份文件时,为文件名加的后缀-->
        <datePattern value="yyyyMMdd.TXT" />

        <!--日志最大个数,都是最新的-->
        <!--rollingStyle节点为Size时,只能有value个日志-->
        <!--rollingStyle节点为Composite时,每天有value个日志-->
        <maxSizeRollBackups value="20" />

        <!--可用的单位:KB|MB|GB-->
        <maximumFileSize value="3MB" />

        <!--置为true,当前最新日志文件名永远为file节中的名字-->
        <staticLogFileName value="true" />

        <!--输出级别在INFO和ERROR之间的日志-->
        <filter type="log4net.Filter.LevelRangeFilter">
            <param name="LevelMin" value="ALL" />
            <param name="LevelMax" value="FATAL" />
        </filter>
        <layout type="log4net.Layout.PatternLayout">
            <conversionPattern value="%date [%thread] %-5level %logger -
%message%newline"/>
        </layout>
    </appender>
```

```
    <root>
        <priority value="ALL"/>
        <level value="ALL"/>
        <appender-ref ref="rollingAppender" />
    </root>
</log4net>
```

3.使用Log4net配置

```
public static IHostBuilder CreateHostBuilder(string[] args)
        {
            return Host.CreateDefaultBuilder(args) //创建默认主机的建造者;
            .ConfigureLogging(loggbuild =>
            {
                loggbuild = loggbuild.AddLog4Net("CfgFile/log4net.Config");
            })   ///配置logging(指定使用Log4net)
            .ConfigureWebHostDefaults(webBuilder =>
             {
                 webBuilder.UseStartup<Startup>(); //如何配置?  配置全交给Startup来完成;
             }).UseServiceProviderFactory(new AutofacServiceProviderFactory());
        }
```

4.就可以支持注入了，可以在控制器中使用了

34.说说脚本启动ASP.NET Core Web项目

介绍两种方式:

第一种：定位到Web项目的编译地址下，就是bin文件夹下的.NET5文件夹，然后在当前文件夹下打开命令提示窗口；dotnet dll文件 ---urls=http://ip地址：端口号 回车即可；

第二种：定位到Web项目的根目录下，然后在当前文件夹下打开命令提示窗口；dotnet run ---urls=http://ip地址：端口号 回车即可；

推按第二种，第二种方式，在启动的时候，会自动编译项目，然后启动dll文件；

35.说说Core WebApi的Swagger。

Swagger是一个Api说明文档，支持Api测试；现在CoreWebApi开发使用swagger还挺多的；

在.NET5中已经内置了Core WebApi；配置流程如下:

1.Nuget引入程序集：Swashbuckle.AspNetCore.SwaggerGen

2.配置服务:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();
    services.AddSwaggerGen(c =>
    {
     c.SwaggerDoc("v1", new OpenApiInfo { Title = "WebApplication1", Version = "v1" });
    });
 }
```

3.配置使用中间件

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
  app.UseSwagger();
  app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json",
"WebApplication1 v1"));
}
```

36.说说Core WebApi特性路由。

在Core WebApi中，每一个Api必须指定特性路由，即在Api或者控制器上标记特性Route("api/[Controller]/Api")；访问Api，就按照这个格式访问；

37.说说RESTful是什么。

在传统的服务中，比方说WebService,WCF，Remouting，都是通过调用方法来做到一个进程去调用另外一个进程的服务，在Core WebApi中是把要调用的服务资源化，比方说有图书资源，Books，学生资源Studentlist，每一个资源对应一个控制器，然后对外提供增删改查等操作；对外提供统一的Uri, 可以对资源Books，资源Studentlist做增删改查的操作；访问的是资源，可以根据不同的额访问方式，做不同的事儿；

38.说说脚本在请求Web CoreApi的时候，为什么会发生跨域问题?

跨域问题：本质是浏览器的行为，浏览器有一个同源策略，同源策略：协议、IP地址相同就认为是同源；否则就非同源；同源策略限定脚本请求只能请求同源的服务器返回的内容才给正常的使用；否则就会报跨域问题；其实我们在请求Core WebApi的时候，浏览器直接访问Api没有问题，如果是脚本请求，就会出现跨域问题；

39.如何解决跨域问题?

三种方式：

1.后台模拟Http请求，既然是浏览器的行为，就避开浏览器，先来一个同源的服务器去请求，然后由服务器模拟http请求去请求到Core WebApi的资源，然后响应给前端；

2.JSONP，思路：通过html部分标签发起请求，比方说 等等，发起请求是可以避开同源策略的，使用这些标签发起请求，然后带有一个回调函数，然后得到请求后，把回调函数之心一次，把数据解析后使用；

3.服务端允许跨域，多种方式，可以自己定义中间件支持跨域，只要把响应的Response的头信息Header中写入“Access-Control-Allow-Origin” 即可支持跨域；如果需要让所有的Api都支持跨域，就可以写一个中间件从管道处理模型中去支持跨域，如果要选择性的支持跨域，可以使用ActionFilter来完成，也可以通过Cors（ASP.NET Core中提供的中间件，可以支持配置不同的跨域规则）来配置支持跨域；

40.说说你了解到的鉴权授权技术。

1.传统的授权技术：通过Session、Cookie完成授权；实现特点: 让无状态的http请求，变的有状态，让第一次请求和第二次请求之间产生联系，第一次请求进入服务器，在服务器写入一组session，然后返回sessionId给客户端存在Cookie,第二次请求，从cookie中渠道SessionId,传递给服务器，服务器鉴别SessionId,如果是上一次来的SessionId,就认为之前来请求过；就认为有权限；

2.流行鉴权授权方式：Token授权，在Core WebApi中主要就是JWT和IdentityServer4;都是独立的授权中心，授权后办法token，然后客户端带着token去请求Api,Api方验证Token，验证通过就有权限，验证不通过就没有权限；

41.请问对gRPC有了解吗，说说gRPC。

有了解，说gRPC可以先说RPC,PRC：所谓RPC(remote procedure call 远程过程调用)框架实际是提供了一套机制，使得应用程序之间可以进行通信，而且也遵从server/client模型。使用的时候客户端调用server端提供的接口就像是调用本地的函数一样。

所谓gRPC 是由谷歌开发的一个高性能、开源和通用的 RPC 框架，面向移动和 HTTP/2 设计。目前提供 C、Java 和 Go 语言版本，分别是：grpc, grpc-java, grpc-go. 其中 C 版本支持 C, C++, Node.js, Python, Ruby, Objective-C, PHP 和 C# 支持.

42.gRPC有几种模式?

四种模式：

1，简单模式：简单模式只是使用参数和返回值作为服务器与客户端传递数据的方式，最简单。

2，客户端流模式：即从客户端往服务器端发送数据使用的是流，即服务器端的参数为流类型，然而在服务器相应后返还数据给客户端，使用的也是流的send方法。一般在服务器端的代码，需要先recv再send，而客户端与此相反。但是在后面的双向模式中可以使用go的协程协作。

3，服务器端流模式：即服务器端返回结果的时候使用的是流模式，即传入的数据是通过参数形式传入的。但是在往客户端发送数据时使用send方法，与客户端返回数据的方式大同小异。

4，双向模式：客户端如果不适用协程，那么发送必须在接收之前。如果使用协程，发送与接收并没有先后顺序。为了保证协程的同步，可以使用互斥量进行约束。

43.说说如何使用C#实现简单模式gRPC

分为客户端和服务端；

服务端：

1.通过vs新建一个gRPC服务，会内置一proto文件；内容如下，可以理解成是一个模板，通过这个模板可以生成对应的类文件。

```
syntax = "proto3";  //规范---标准---工具生成C#

option csharp_namespace = "Zhaoxi.gRPCDemo.DefaultServer";

package greet;

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply);
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}
```

```
// The response message containing the greetings.
message HelloReply {
  string message = 1;
}
```

2.需要让这个文件生效，就必须要在项目文件中配置使用这个文件；GrpcServices="Server",这是服务端的配置；

```
  <ItemGroup>
    <Protobuf Include="Protos\CustomMath.proto" GrpcServices="Server" />
    <Protobuf Include="Protos\greet.proto" GrpcServices="Server" />
  </ItemGroup>
```

3.编译，就可以通过这个模板生成一些类，包含这些类的方法；

客户端：

1.Vs新建一个控制台，作为客户端

2.把服务端的那个proto文件，连同文件一起Copy到客户端来。

3.配置客户端的项目文件,如下。请注意 GrpcServices="Client"

```
  <ItemGroup>
    <Protobuf Include="Protos\greet.proto" GrpcServices="Client" />
    <Protobuf Include="Protos\CustomMath.proto" GrpcServices="Client" />
  </ItemGroup>
```

4.编译后，编写调用gRPC的方法如下：

```
private static async Task TestHello()
{
     using (var channel = GrpcChannel.ForAddress("https://localhost:5001"))
     {
        var client = new Greeter.GreeterClient(channel);
        var reply = await client.SayHelloAsync(new HelloRequest { Name = "朝夕教育" });
        Console.WriteLine("Greeter 服务返回数据: " + reply.Message);
     }
 }
```

44.说说gRPC的拦截器有哪些?

分为客户端拦截器，和服务端连接器，是AOP的编程思想的体现。

分别有：

BlockingUnaryCall：拦截阻塞调用

AsyncUnaryCall:　拦截异步调用

AsyncServerStreamingCall 拦截异步服务端流调用

AsyncClientStreamingCall 拦截异步客户端流调用

AsyncDuplexStreamingCall 拦截异步双向流调用

UnaryServerHandler 用于拦截和传入普通调用服务器端处理程序

ClientStreamingServerHandler 用户拦截客户端流调用的服务端处理程序

ServerStreamingServerHandler 用于拦截服务端流调用的服务器端处理程序

DuplexStreamingServerHandler 用于拦截双向流调用的服务器端处理程序

45.gPRC作为一种被调用的服务，有什么保护安全的措施吗?

有的，可以使用JWT，无论是对称可逆加密还是非对称可逆加密，都是可以支持的;

46.请问对EFCore有了解吗?

有了解。

Entity Framework Core是适用于.NET的新式物件资料库对应程式。其支援LINQ查询、变更追踪、更新以及结构描述移转。
EF Core透过[资料库提供者外挂程式模型]来搭配使用SQL Server/SQL Azure、SQLite、Azure Cosmos DB、MySQL、PostgreSQL及更多资料库。。、

47.说说EFCore查询的性能调优小技巧。

如果说查询出来的数据，只是做展示，不做增删改查，可以在查询的时候，增加AsNoTracking()方法，可以提高性能，可以避免在内存中存在副本;

建议在查询的时候，多使用Find()方法，会有限走内存缓存，如果内存已经存在，就不会去数据库中去操查询数据;

48.EFCore 如果通过数据生成实体和DbContext?

步骤如下:

1.Nuget引入 如下程序集

```
Install-Package Microsoft.EntityFrameworkCore
Install-Package Microsoft.EntityFrameworkCore.SqlServer
Install-Package Microsoft.EntityFrameworkCore.Tools
```

2.在Vs中打开工具--nuget包管理器--程序包管理器控制台：命令执行:
Scaffold-DbContext "Data Source=DESKTOP-63QE7M1;Initial Catalog=ZhaoxiEduDataBase;User ID=sa;Password=sa123" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Entity -Force -Context ZhaoxiDbContext -ContextDir /

注：命令参数应用如下:

```
命令参数:
-OutputDir *** 实体文件所存放的文件目录
-ContextDir *** DbContext文件存放的目录
-Context *** DbContext文件名
-Schemas *** 需要生成实体数据的数据表所在的模式
-Tables *** 需要生成实体数据的数据表的集合
-DataAnnotations
-UseDatabaseNames 直接使用数据库中的表名和列名（某些版本不支持）
-Force 强制执行，重写已经存在的实体文件
```

49.说说对SaveChanges的理解。

SaveChanges是以Context为维度的一次提交，对于数据库操作的一切动作，只要是在同一个Context实例，所有的操作，在调用SaveChanges方法后，统一体现到数据库中去；

51.说说对EFCore中EntityState的理解。

因为EFCore对于数据库的所有操作都是通过上下文DbContext来完成的，且是通过SaveChanges方法统一落实到数据库中去的；EntityState是EFCore 在对数据库操作增删改的时候，记录当前被操作的数据对象和Context的关系，针对与不同的操作，对应的一个状态信息，一共五种状态；一共五种：

Detached = 0， 当前对象和context没有任何关系，没有被上下文跟踪

Unchanged=1， 当前对象被context跟踪，数据没有做任何修改

Deleted=2， 当前对象被context跟踪，且标记是数据删除，调用SaveChanges后将会从数据中删除；

Modified=3， 当前对象被context跟踪，且有属性数据被修改过，调用SaveChanges后将会从数据中修改；

Added=4 当前对象被context跟踪，且数据并没有存在数据库中，调用SaveChanges后将会新增到数据库中去；

52.说说什么是导航属性和引用属性。

实体框架 中的导航属性提供了一种在两个实体类型之间导航关联的方法。 导航属性在概念模型中由 NavigationProperty 元素 (CSDL) 定义。 针对对象参与到其中的每个关系，各对象均可以具有导航属性。 使用导航属性，您可以在两个方向上导航和管理关系，如果重数为一或者零或一，则返回 EntityReference，或者如果重数为多个，则返回 EntityCollection。 也可以选择单向导航，这种情况下可以删除导航属性。