路由功能负责将传入请求映射到路由处理程序

### **URL 匹配**

URL 匹配过程

1. 请求将进入 RouterMiddleware
2. 路由中间件具有IRouter 列表
3. 路由中间件循环调用每个IRouter 的RouteAsync方法
4. 如果RouteAsync方法设置了RouteContext.Handler，那么匹配结束，往下执行，否则回到步骤3
5. 路由中间件调用RouteContext.Handler

### **添加路由到路由中间件**

路由提供 Route 类，作为 IRouter 的标准实现。 Route 的 RouteAsync 方法使用 route template 语法来进行 url 匹配。

以下两个示例可创建等效路由：

routes.MapRoute(

    name: "default\_route",

    template: "{controller}/{action}/{id?}",

    defaults: new { controller = "Home", action = "Index" });

routes.MapRoute(

    name: "default\_route",

    template: "{controller=Home}/{action=Index}/{id?}");

此示例可添加路由约束和数据令牌：

routes.MapRoute(

    name: "us\_english\_products",

    template: "en-US/Products/{id}",

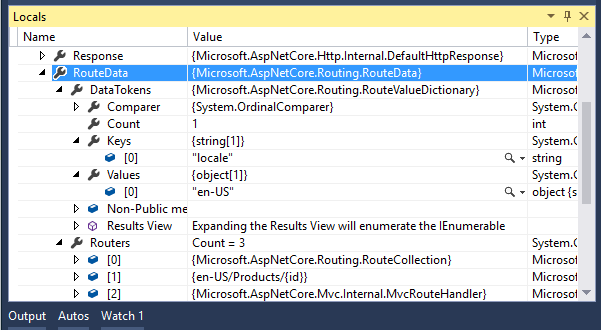
    defaults: new { controller = "Products", action = "Details" },

    // 约束 Id 的数据类型

    constraints: new { id = new IntRouteConstraint() },

    dataTokens: new { locale = "en-US" });

如下请求示例：



## **使用路由中间件**

添加 NuGet 包“Microsoft.AspNetCore.Routing”。

注册服务

public void ConfigureServices(IServiceCollection services){

    services.AddRouting();

}

添加中间件

public void Configure (IApplicationBuilder app, ILoggerFactory loggerFactory) {

    // IRouteHandler 路由处理器，其功能是根据 路由数据 设置请求委托

    // RouteHandler 其只包含一个请求委托，无论什么 路由数据，都将设计为这个委托

    // MvcAttributeRouteHandler 则根据 路由数据 选择合适的控制器方法设置请求委托

    var trackPackageRouteHandler = new RouteHandler (context => {

        var routeValues = context.GetRouteData ().Values;

        return context.Response.WriteAsync (

            $"Hello! Route values: {string.Join(", ", routeValues)}");

    });

    // 路由生成器

    // 将多个 路由 关联到一个默认 路由处理器

    // 默认 路由处理器 为 trackPackageRouteHandler

    var routeBuilder = new RouteBuilder (app, trackPackageRouteHandler);

    // 向路由生成器添加路由

    // 该 路由 关联的 路由处理器 为 默认路由处理器

    routeBuilder.MapRoute (

        "Track Package Route",

        "package/{operation:regex(^(track|create|detonate)$)}/{id:int}");

    // 向路由生成器添加路由

    // 该 路由 关联的 路由处理器 为 第二个参数

    routeBuilder.MapGet ("hello/{name}", context => {

        var name = context.GetRouteValue ("name");

        return context.Response.WriteAsync ($"Hi, {name}!");

});

    // 生成最终的路由集合

    var routes = routeBuilder.Build ();

    // 向路由中间件添加路由集合

    app.UseRouter (routes);

}

框架可提供一组扩展方法，用于将路由添加到路由生成器：

MapRoute

MapGet

MapPost

MapPut

MapDelete

MapVerb

## **保留的路由名称**

以下关键字是保留的名称，它们不能用作路由名称或参数：

* action
* area
* controller
* handler
* page