**RouteBuilder路由生成器**

app.UseMvc ((Microsoft.AspNetCore.Routing.IRouteBuilder routes) =>

{

    routes.MapRoute (

        name: "default",

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

});

IRouteBuilder在UseMvc中被使用用于注册路由

**IRouteBuilder**

public interface IRouteBuilder

{

    // 注册的路由器

    IList<IRouter> Routes { get; }

    // 生成路由器

    IRouter Build();

}

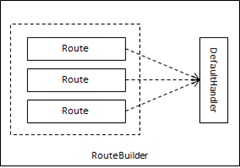
IRouteBuilder的Build方法将我们注册的路由器封装成一个路由器

Build返回的具体类型是一个RouteCollection，其包含了我们注册的IRouter并继承了IRouter

**IRouteBuilder.MapRoute**

扩展方法MapRoute实际向IRouteBuilder.Routes列表添加Route，

我们前面介绍，Route依赖RouteHandler，MapRoute使用的RouteHandler在哪？？

[](http://images2015.cnblogs.com/blog/19327/201612/19327-20161222091507667-769892869.png)

我们可以在IRouteBuilder中设置DefaultHandler，而我们使用MapRoute所注册的路由（Route）都使用DefaultHandler来设置请求处理器

app.UseRouter ((Microsoft.AspNetCore.Routing.IRouteBuilder routes) => {

    // 设置默认 RouteHandler

    routes.DefaultHandler = new RouteHandler (handler);

    // 添加路由

    routes.MapRoute (

        name: "default",

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

        handler);

});

**RouterMiddleware路由中间件**

与其他中间间一样

路由中间件是路由分析并映射到处理方法的入口

**中间件处理流程**

public class RouterMiddleware

{

    ...

    // 中间件处理流程

    public async Task Invoke (HttpContext httpContext)

    {

        // 生成路由上下文

        var context = new RouteContext (httpContext);

        context.RouteData.Routers.Add (\_router);

        // 使用路由器分析 路由数据（RouteContext.RouteData） 并指定 请求处理器（RouteContext.Handler）

        // 注：\_router 是我们通过 RouteBuilder.Build 生成的路由

        await \_router.RouteAsync (context);

        if (context.Handler == null) {

            \_logger.RequestDidNotMatchRoutes ();

            // 如果没有处理方法，调用后续中间件

            await \_next.Invoke (httpContext);

        } else {

            httpContext.Features[typeof (IRoutingFeature)] = new RoutingFeature () {

                RouteData = context.RouteData,

            };

            // 调用请求处理方法

            await context.Handler (context.HttpContext);

        }

    }

}

**UseRouter扩展方法**

RouterMiddleware提供了扩展方法，用于向依赖注入容器提供中间件

public static IApplicationBuilder UseRouter(this IApplicationBuilder builder, IRouter router)

{

    return builder.UseMiddleware<RouterMiddleware>(new object[] { router });

}

从这里我们可以猜到，UseMVC也是向路由中间件注册一个路由