**启用Razor**

var builder = WebApplication.CreateBuilder(args);

// 注册Razor依赖

builder.Services.AddRazorPages();

var app = builder.Build();

if (!app.Environment.IsDevelopment())

{

    app.UseExceptionHandler("/Error");

    app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

// 注册Razor中间件

app.MapRazorPages();

app.Run();

**Razor**

Razor文件已.cshtml作为后缀，以下时Razor的示例

Pages/Index2.cshtml 文件

@page

@using RazorPagesIntro.Pages

@model Index2Model

<h2>Separate page model</h2>

<p>

    @Model.Message

</p>

一个Razor页面具有一个对应的Model，如下

Pages/Index2.cshtml.cs

using Microsoft.AspNetCore.Mvc.RazorPages;

using Microsoft.Extensions.Logging;

using System;

namespace RazorPagesIntro.Pages

{

    public class Index2Model : PageModel

    {

        public string Message { get; private set; } = "PageModel in C#";

        public void OnGet()

        {

            Message += $" Server time is { DateTime.Now }";

        }

    }

}

1. Razor文件的一些约定

* 页面必须已@page开头，表示不需要经过控制器的处理
* 页面放在Pages文件下
* @model标识页面使用的model

1. Razor路由与文件关系

* /Pages/Index.cshtml 对应的路由为 / 或 /Index
* /Pages/Contact.cshtml对应的路由为 /Contact
* /Pages/Store/Contact.cshtml对应的路由为 /Store/Contact
* /Pages/Store/Index.cshtml对应的路由为 /Store 或 /Store/Index

**Razor页面不需要控制器参与**

Razor页面的操作是由Model直接处理的，如下一个示例

.cshtml页面

@page

@model RazorPagesContacts.Pages.Customers.CreateModel

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

<p>Enter a customer name:</p>

<form method="post">

    Name:

    <!-- 绑定 model 的 Customer.Name 属性 -->

    <input asp-for="Customer!.Name" />

    <input type="submit" />

</form>

Model代码

public class CreateModel : PageModel

{

    private readonly Data.CustomerDbContext \_context;

    // 依赖注入注入上下文

    public CreateModel(Data.CustomerDbContext context)

    {

        \_context = context;

    }

    // Get 请求

    public IActionResult OnGet()

    {

        // 返回页面

        return Page();

    }

    // 页面使用的属性

    [BindProperty]

    public Customer? Customer { get; set; }

    // Post 请求

    public async Task<IActionResult> OnPostAsync()

    {

        // 验证模型状态

        if (!ModelState.IsValid)

        {

            // 返回页面

            return Page();

        }

        // 添加数据库

        if (Customer != null) \_context.Customer.Add(Customer);

        await \_context.SaveChangesAsync();

        // 重定向到 index

        return RedirectToPage("./Index");

    }

}

知识点在注释里

[BindProperty(SupportsGet = true)]：如果想要在Get请求上绑定属性，则使用SupportsGet = true

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers：添加该Helper，可使我们识别一些特殊的标签，如asp-for="Customer!.Name"

**示例 2**

继续看一个例子

如下是首页列表的一个示例

.cshtml页面

@page

@model RazorPagesContacts.Pages.Customers.IndexModel

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

<h1>Contacts home page</h1>

<form method="post">

    <table class="table">

        <thead>

            <tr>

                <th>ID</th>

                <th>Name</th>

                <th></th>

            </tr>

        </thead>

        <tbody>

        @if (Model.Customers != null)

        {

            foreach (var contact in Model.Customers)

            {

                <tr>

                    <td> @contact.Id </td>

                    <td>@contact.Name</td>

                    <td>

                        <!-- 跳转到编辑页，携带路由参数id (asp-route-{参数名}) -->

                        <a asp-page="./Edit" asp-route-id="@contact.Id">Edit</a> |

                        <!-- 处理方法名 OnPostDelete，并携带路由参数 id -->

                        <!-- 处理方法名 = On + methdo + asp-page-handler -->

                        <button type="submit" asp-page-handler="delete" asp-route-id="@contact.Id">delete</button>

                    </td>

                </tr>

            }

        }

        </tbody>

    </table>

    <a asp-page="Create">Create New</a>

</form>

Model代码

public class IndexModel : PageModel

{

    private readonly Data.CustomerDbContext \_context;

    public IndexModel(Data.CustomerDbContext context)

    {

        \_context = context;

    }

    public IList<Customer>? Customers { get; set; }

    // GET 请求

    public async Task OnGetAsync()

    {

        // 从数据库获取数据，放到 Customers 中

        Customers = await \_context.Customer.ToListAsync();

    }

    public async Task<IActionResult> OnPostDeleteAsync(int id)

    {

        var contact = await \_context.Customer.FindAsync(id);

        if (contact != null)

        {

            \_context.Customer.Remove(contact);

            await \_context.SaveChangesAsync();

        }

        // 重定向到根索引页

        return RedirectToPage();

    }

}

**示例：编辑页面**

.cshtml代码

@page "{id:int}"

@model RazorPagesContacts.Pages.Customers.EditModel

@{

    ViewData["Title"] = "Edit";

}

<h1>Edit</h1>

<h4>Customer</h4>

<hr />

<div class="row">

    <div class="col-md-4">

        <form method="post">

            <div asp-validation-summary="ModelOnly" class="text-danger"></div>

            <input type="hidden" asp-for="Customer!.Id" />

            <div class="form-group">

                <label asp-for="Customer!.Name" class="control-label"></label>

                <input asp-for="Customer!.Name" class="form-control" />

                <span asp-validation-for="Customer!.Name" class="text-danger"></span>

            </div>

            <div class="form-group">

                <input type="submit" value="Save" class="btn btn-primary" />

            </div>

        </form>

    </div>

</div>

<div>

    <a asp-page="./Index">Back to List</a>

</div>

@section Scripts {

    @{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

Model代码

public class EditModel : PageModel

{

    private readonly RazorPagesContacts.Data.CustomerDbContext \_context;

    public EditModel(RazorPagesContacts.Data.CustomerDbContext context)

    {

        \_context = context;

    }

    [BindProperty]

    public Customer? Customer { get; set; }

    public async Task<IActionResult> OnGetAsync(int? id)

    {

        if (id == null)

        {

            return NotFound();

        }

        Customer = await \_context.Customer.FirstOrDefaultAsync(m => m.Id == id);

        if (Customer == null)

        {

            return NotFound();

        }

        return Page();

    }

    // To protect from overposting attacks, enable the specific properties you want to bind to.

    // For more details, see https://aka.ms/RazorPagesCRUD.

    public async Task<IActionResult> OnPostAsync()

    {

        if (!ModelState.IsValid)

        {

            return Page();

        }

        if (Customer != null)

        {

            \_context.Attach(Customer).State = EntityState.Modified;

            try

            {

                await \_context.SaveChangesAsync();

            }

            catch (DbUpdateConcurrencyException)

            {

                if (!CustomerExists(Customer.Id))

                {

                    return NotFound();

                }

                else

                {

                    throw;

                }

            }

        }

        return RedirectToPage("./Index");

    }

    private bool CustomerExists(int id)

    {

        return \_context.Customer.Any(e => e.Id == id);

    }

}

@page "{id:int}" 指令：告诉页面接受包含 int 路由数据的页面请求。如果页面请求未包含可转换为 int 的路由数据，则运行时返回 HTTP 404（未找到）错误。若要使 ID 可选，请将使用 @page "{id:int?}"

**示例：验证**

.cshtml代码

@page

@model RazorPagesContacts.Pages.Customers.CreateModel

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

<p>Validation: customer name:</p>

<form method="post">

    <div asp-validation-summary="ModelOnly"></div>

    <!-- 用于存放 Customer.Name 验证错误文本 -->

    <span asp-validation-for="Customer.Name"></span>

    Name:

    <input asp-for="Customer.Name" />

    <input type="submit" />

</form>

<!-- 需要添加这些库 -->

<script src="~/lib/jquery/dist/jquery.js"></script>

<script src="~/lib/jquery-validation/dist/jquery.validate.js"></script>

<script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js"></script>

数据模型代码

using System.ComponentModel.DataAnnotations;

namespace RazorPagesContacts.Models

{

    public class Customer

    {

        public int Id { get; set; }

        [Required, StringLength(10)]

        public string? Name { get; set; }

    }

}

通过标签在数据上指定验证规则

如下示例用于验证的特性

public class Movie

{

    public int ID { get; set; }

    [StringLength(60, MinimumLength = 3)]

    [Required]

    public string Title { get; set; }

    [Display(Name = "Release Date")]

    [DataType(DataType.Date)]

    public DateTime ReleaseDate { get; set; }

    [Range(1, 100)]

    [DataType(DataType.Currency)]

    [Column(TypeName = "decimal(18, 2)")]

    public decimal Price { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z\s]\*$")]

    [Required]

    [StringLength(30)]

    public string Genre { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z0-9""'\s-]\*$")]

    [StringLength(5)]

    [Required]

    public string Rating { get; set; }

}