## Урок 4. Routing и Bundles

Цель урока: Изучить инициализацию маршрутизации. Деление на Areas в приложении. Принципы создания маршрутизации.

### Controller и Action.

Веб-сайт состоит из страниц. Вообще веб-сайт состоит не из страниц, а из ответов на запросы, но какую-то определенную структуру мы ходим иметь.

Собственно, у нас есть маршрутизатор, который должен определить, какой метод у какого контроллера вызвать. Поэтому 2 основных параметра, которые обязательно должны быть – это controller и action. Рассмотрим в App\_Start/RouteConfig.cs как задаюется шаблон маршрутов:

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

);

*Примечание: инициализацию api маршрутизации мы рассмотрим позже.*

Таким образом, url = «/Role/Create/2» будет означать, что мы находим контроллер RoleController, в этом контроллере находим Create метод, который может принимать (а может и не принимать) параметр id. И если он принимает параметр id, то id = 2 или даже id = “2” в зависимости что за тип будет.

Defaults обозначает, что если строка будет “/Role/Create” – то в случае что Create метод с параметром id и по умолчанию не стоит значение, или не может быть создано default(), то по возможности будет выбран другой метод (мы же полиморфны). Иначе будет сгенерирована ошибка, что не найден метод готовый принять такой запрос.

public ActionResult Index(int? id)

{

//ok

return View();

}

…

public ActionResult Index(int id = 0)

{

//ok

return View();

}

…

public ActionResult Index(int id)

{

//fail

return View();

}

В случае url = “/Role” будет вызван метод Index в контроллере RoleController.

В случае url = “/” будет вызван метод Index в контроллере HomeController.

Рассмотрим на примерах.

### BaseController

Создадим несколько контроллеров, но для того чтобы не создавать постоянно доступ к репозиторию, превоначально создадим базовый контроллер BaseController:

public abstract class BaseController : Controller

{

[Inject]

public IRepository Repository { get; set; }

}

И

public class HomeController : BaseController

{

public ActionResult Index()

{

return View();

}

}

View Home/Index.cshtml

@{

ViewBag.Title = "LessonProject";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>LessonProject</h2>

<p>

<div class="menu">

<a href="@Url.Action("Index", "Role")">Роли</a>

@Html.ActionLink("Пользователи", "Index", "User")

</div>

</p>

Добавим для просмотра RoleController:

public class RoleController : BaseController

{

public ActionResult Index()

{

var roles = Repository.Roles.ToList();

return View(roles);

}

}

И

@model IList<LessonProject.Model.Role>

@{

ViewBag.Title = "Roles";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Roles</h2>

<p>

@foreach (var role in Model)

{

<div class="item">

<span class="id">

@role.ID

</span>

<span class="name">

@role.Name

</span>

<span class="Code">

@role.Code

</span>

</div>

}

</p>

И такой же UserController собственно сделаем.

Рассмотрим как задаются маршруты с помощью Url.Action() и Html.ActionLink().

Url.Action() – принимает параметры, первым – action, потом – controller, потом через new {} – можно задавать и перечислять все остальные.

Html.ActionLink() – формирует тег <a>, первый параметр – наименование ссылки, второй – action, третьим – controller, четвертым(или пятым) параметром идут другие атрибуты тега, если мы хотим добавить другие параметры для маршрутизации – мы должны явно указать 5м параметром null. Т.е.

@Html.ActionLink(“Пользователь под номером 1”, “Item”, “User”, new {id = 1}, **null**)

Если не указать null:

@Html.ActionLink(“Пользователь под номером 1”, “Item”, “User”, new {id = 1})

то ссылка будет выглядеть так:

<a href=”/User/Item?length=4”>Пользователь под номером 1</a>

### Порядок в объявлении маршрутов

Создадим маршрут, который будет расположен ранее и относится только к RoleController:

routes.MapRoute(

name: "Role",

url: "roli/{action}/{id}",

defaults: new { controller = "Role", action = "Index", id = UrlParameter.Optional }

);

Однозначно его задавать, но не наименованием, а строкой “roli/{action}/{id}”, т.е. тут задается только action, а controller объявляется в defaults

Результат. Ссылка стала:

<a href="/roli">Роли</a>

Уберем из defaults action=”Index”:

<a href="/roli/Index">Роли</a>

Поместим после объявления “Defaults”:

<a href="/Role">Роли</a>

Так как с помощью вышестоящего правила уже можно однозначно задать путь маршрута.

Еще надо рассмотреть один метод ктороый называется IgnoreRoute, он указывает маршрутизатору, что если url подходит под шаблон, то нужно вернуть ресурс, который расположен по тому адресу, а не пытаться находить контроллер.

### Ограничения (Constrains)

Мы можем добавить в маршрутизацию ограничения запросов браузера, которые соответствуют особому маршруту. Например, id должен быть в нашем случае числовым:

routes.MapRoute(

name: "Role",

url: "roli/{action}/{id}",

defaults: new { controller = "Role", action = "Index", id = UrlParameter.Optional },

**constraints : new {id = @"\d+"}**

);

При передаче id, которое не соответствует данному условию, будет или выбран другой маршрут или метод не будет найден и ссылка будет битой:

Для:

<a href="@Url.Action("Index", "Role", new { id = "privet" })">Роли</a>

Будет:

<a href="/Role/Index/privet">Роли</a>

А для:

<a href="@Url.Action("Index", "Role", new { id = "1" })">Роли</a>

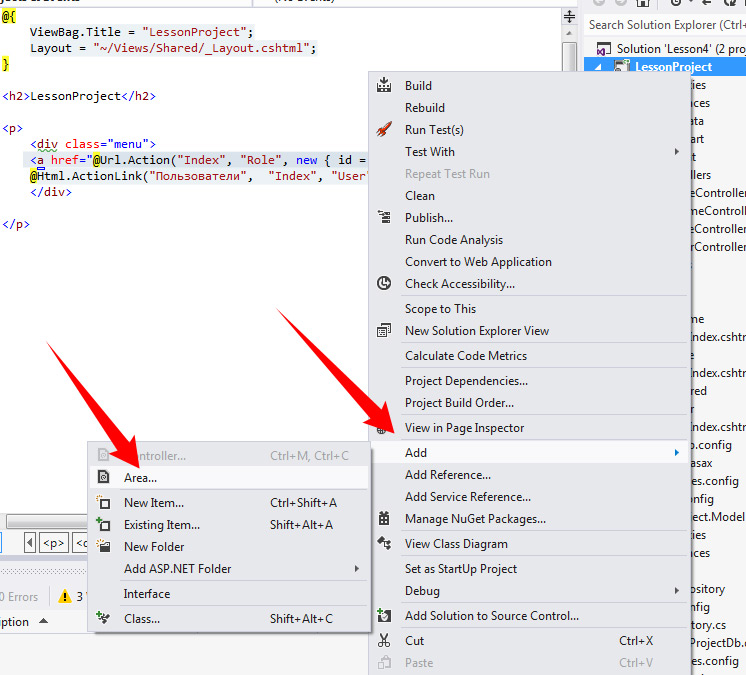
Будет:

<a href="/roli/Index/1">Роли</a>

*Примечание: Более подробно про ограничения можно узнать тут:* [*http://stephenwalther.com/archive/2008/08/07/asp-net-mvc-tip-30-create-custom-route-constraints.aspx*](http://stephenwalther.com/archive/2008/08/07/asp-net-mvc-tip-30-create-custom-route-constraints.aspx)

### Areas

Дабы разделить какие-то более общие по свойствам функциональные модули веб-приложения. Например, форум отдельно от всего сайта. Мы же поделим на часть Admin – где будет админка, и всё остальное, которое будет называться Default.



Сделаем следующие действия:

* Переименуем \_Default в Default везде.
* Перенесем свои контроллеры (кроме BaseController) в папку Areas/Default/Controllers
* Переименуем namespace для контроллеров в LessonProject.Areas.Default.Controllers
* Исправляем DefaultAreaRegistration:

Тут важно обратить внимание на новый параметр для задания маршрутов: namespaces, он указывает, из каких namespace можно выбирать контроллеры для разбора маршрута:

public class DefaultAreaRegistration : AreaRegistration

{

public override string AreaName

{

get

{

return "Default";

}

}

public override void RegisterArea(AreaRegistrationContext context)

{

context.MapRoute(

name : "default",

url : "{controller}/{action}/{id}",

defaults : new { controller = "Home", action = "Index", id = UrlParameter.Optional },

namespaces : new [] { "LessonProject.Areas.Default.Controllers" }

);

}

}

* В Global.asax есть строка AreaRegistration.RegisterAllAreas(); которая регистрирует все найденные объявления area, но она нам не подходит, так как если DefaultArea зарегистрировать быстрее AdminArea, то будет срабатывать маршрутизация Default, а в админку мы уже не сможем попасть поэтому исправляем:

var adminArea = new AdminAreaRegistration();

var adminAreaContext = new AreaRegistrationContext(adminArea.AreaName, RouteTable.Routes);

adminArea.RegisterArea(adminAreaContext);

var defaultArea = new DefaultAreaRegistration();

var defaultAreaContext = new AreaRegistrationContext(defaultArea.AreaName, RouteTable.Routes);

defaultArea.RegisterArea(defaultAreaContext);

* Регистрацию маршрутов убираем (не api).

Запускаем.

### API

Разобраться и описать как работает апи

<http://www.asp.net/web-api/overview/web-api-routing-and-actions/routing-in-aspnet-web-api>