

# View, Razor, страница ошибки

**Цель урока.** Научиться делать вывод данных в html, использование Razor. Helper-ы. PageableData. Динамические формы. RedirectToLogin, RedirectToNotFoundPage. Страница ошибки. RssActionResult.

## Основа

Итак, рассмотрим как устроена часть View.

В контроллере все action-методы возвращают тип ActionResult. И для вывода результата мы используем:

```
return View(modelData);
```

Основными параметрами View может быть:

- Имя, обычно оно совпадает с именем action-метода. В случае если надо вызвать иной по имени View, то используется конструкция `return View("ViewName", modelData)`.
- Данные для отображения во View. Необязательный параметр. При передаче во View этот объект данных будет обозначаться Model. Для связывания типа данных во View указывается ожидаемый тип данных:

```
@model LessonProject.Model.User
```

- Layout. Необязательный параметр. При указании этого параметра по данной строке найдется страница-контейнер и вызовется. View-часть будет обработана методом `RenderBody()`

Выбор, какой же View использовать происходит следующим образом:

- Ищется в папке `/Areas/[Area]/Views/[ControllerName]/`
- Ищется в папке `/Areas/[Area]/Views/Shared/`
- Ищется в папке `/Views/[ControllerName]/`
- Ищется в папке `/Views/Shared/`

приступим к изучению.

## Razor

При создании View есть выбор между двумя движками: ASPX и Razor. Первый мы не будем использовать в дальнейшем, поэтому поговорим о Razor.

View name:

View engine:

ASPX был громоздким движком с тегами `<% %>` для выполнения кода и `<%= %>` для вывода данных.

Razor использует конструкцию `@Model.Name`. Т.е. всё, что начинается с `@` переводит в режим или исполнения кода, или вывода данных `@foreach() { ... }`, или `@if() { ... } else { ... }`:

```
@if (Model.Any())
{
    <p>Список</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>
}
```

Внутри `{ }` находятся теги – это маркер того, что это шаблон. Для простого выполнения кода внутри шаблона используем структуру `@{ code }`, для корректного вывода данных внутри атрибутов или текстом конструкция — `@(string result)`:

```

@{
    int i = 0;
}
@foreach (var role in Model)
{
    <div class="item @(i % 2 == 0 ? "odd" : "")">
        <span class="id">
            @role.ID
        </span>
        <span class="name">
            @role.Name
        </span>
        <span class="Code">
            @role.Code
        </span>
    </div>
    i++;
}

```

Чтобы вывести не теговый текст, нужно использовать псевдотэги:

```

@foreach (var role in Model)
{
    @role.Name<text>, </text>
}

```

Для вывода html-текста – или должна возвращаться MvcHtmlString, или использовать конструкцию [Html.Raw\(html-string-value\)](#), иначе текст будет выведен с экранированием тегов.

## PageableData

Рассмотрим постраничный вывод таблицы из БД. Проанализируем:

1. Контроллер должен получить в параметрах значение страницы, которую мы будем выводить
2. По умолчанию это будет первая страница
3. При выводе, мы должны знать:
  1. Список элементов БД, которые выводим
  2. Количество страниц
  3. Текущую страницу

Создадим Generic-класс PageableData (/Models/Info/PageableData.cs):

```
public class PageableData<T> where T : class
{
    protected static int ItemPerPageDefault = 20;

    public IEnumerable<T> List { get; set; }

    public int PageNo { get; set; }

    public int CountPage { get; set; }

    public int ItemPerPage { get; set; }

    public PageableData(IQueryable<T> queryableSet, int page, int itemPerPage = 0)
    {
        if (itemPerPage == 0)
        {
            itemPerPage = ItemPerPageDefault;
        }
        ItemPerPage = itemPerPage;

        PageNo = page;
        var count = queryableSet.Count();

        CountPage = (int)decimal.Remainder(count, itemPerPage) == 0 ? count / itemP
erPage : count / itemPerPage + 1;
        List = queryableSet.Skip((PageNo - 1) * itemPerPage).Take(itemPerPage);
    }
}
```

По умолчанию количество выводимых значений на странице – 20, но мы можем изменить этот параметр в конструкторе. Передаем IQueryable и вычисляем кол-во страниц CountPage. Используя PageNo, выбираем страницу:

```
List = queryableSet.Skip((PageNo - 1) * itemPerPage).Take(itemPerPage);
```

В контроллере используем:

```
public class UserController : DefaultController
{
    public ActionResult Index(int page = 1)
    {
        var data = new PageableData<User>(Repository.Users, page, 30);
        return View(data);
    }
}
...
```

Во View используем данный класс:

```
@model LessonProject.Models.Info.PageableData<LessonProject.Model.User>

@{
    ViewBag.Title = "Users";
    Layout = "~/Areas/Default/Views/Shared/_Layout.cshtml";
}

<h2>Users</h2>

<p>
    @foreach (var user in Model.List)
    {
        <div class="item">
            <span class="id">
                @user.ID
            </span>
            <span class="email">
                @user.Email
            </span>
            <span class="activateDate">
                @user.AddedDate
            </span>
        </div>
    }
</p>
```

Запускаем, проверяем (<http://localhost/User>)



Для продолжения, сгенерируем больше данных (просто ctrl-c, ctrl-v в таблице в Server Explorer)



Перейдем к созданию Helper'а пагинатора, который даст нам возможность пролистывать этот список.

## Helper (PagerHelper)

Так как мы используем bootstrap, то и на базе него будем делать пагинатор. В коде он выглядит так:

```
<div class="pagination">
  <ul>
    <li><a href="#">Prev</a></li>
    <li><a href="#">1</a></li>
    <li><a href="#">2</a></li>
    <li><a href="#">3</a></li>
    <li><a href="#">4</a></li>
    <li><a href="#">5</a></li>
    <li><a href="#">Next</a></li>
  </ul>
</div>
```

Нас интересует только внутренняя часть .

Helper создается как Extension для класса System.Web.Mvc.HtmlHelper. План таков:

- Вывести Prev (сделать активным если надо)
- Вывести ссылки на первые три страницы 1, 2, 3
- Вывести троеточие, если необходимо
- Вывести активной ссылкой текущей страницы
- Вывести троеточие, если необходимо
- Вывести последние три страницы
- Вывести Next (сделать активной если надо)
- Заключить всё в ul и вывести как MvcHtmlString

Код будет выглядеть так:

```

public static MvcHtmlString PageLinks(this HtmlHelper html, int currentPage, int totalPages, Func<int, string> pageUrl)
{
    StringBuilder builder = new StringBuilder();

    //Prev
    var prevBuilder = new TagBuilder("a");
    prevBuilder.InnerHtml = "<";
    if (currentPage == 1)
    {
        prevBuilder.MergeAttribute("href", "#");
        builder.AppendLine("<li class=\"active\">" + prevBuilder.ToString() + "
</li>");
    }
    else
    {
        prevBuilder.MergeAttribute("href", pageUrl.Invoke(currentPage - 1));
        builder.AppendLine("<li>" + prevBuilder.ToString() + "</li>");
    }

    //По порядку
    for (int i = 1; i <= totalPages; i++)
    {
        //Условие что выводим только необходимые номера
        if (((i <= 3) || (i > (totalPages - 3))) || ((i > (currentPage - 2)) &&
(i < (currentPage + 2))))
        {
            var subBuilder = new TagBuilder("a");
            subBuilder.InnerHtml = i.ToString(CultureInfo.InvariantCulture);
            if (i == currentPage)
            {
                subBuilder.MergeAttribute("href", "#");
                builder.AppendLine("<li class=\"active\">" + subBuilder.ToStrin
g() + "</li>");
            }
            else
            {
                subBuilder.MergeAttribute("href", pageUrl.Invoke(i));
                builder.AppendLine("<li>" + subBuilder.ToString() + "</li>");
            }
        }
    }
}

```

```

        else if ((i == 4) && (currentPage > 5))
        {
            //Третьеочие первое
            builder.AppendLine("<li class=\"disabled\"> <a href=\"#\">...</a>
</li>");
        }
        else if ((i == (totalPages - 3)) && (currentPage < (totalPages - 4)))
        {
            //Третьеочие второе
            builder.AppendLine("<li class=\"disabled\"> <a href=\"#\">...</a>
</li>");
        }
    }
    //Next
    var nextBuilder = new TagBuilder("a");
    nextBuilder.InnerHtml = "»";
    if (currentPage == totalPages)
    {
        nextBuilder.MergeAttribute("href", "#");
        builder.AppendLine("<li class=\"active\">" + nextBuilder.ToString() + "
</li>");
    }
    else
    {
        nextBuilder.MergeAttribute("href", pageUrl.Invoke(currentPage + 1));
        builder.AppendLine("<li>" + nextBuilder.ToString() + "</li>");
    }
    return new MvcHtmlString("<ul>" + builder.ToString() + "</ul>");
}

```

Добавим namespace LessonProject.Helper в объявления во View. Это можно сделать двумя способами:

- В самом View  
@using LessonProject.Helper;
- В Web.config (рекомендуется)

```

<configSections>
...
<sectionGroup name="system.web.webPages.razor" type="System.Web.WebPages.Razor.Confi
guration.RazorWebSectionGroup, System.Web.WebPages.Razor, Version=2.0.0.0, Culture=n
eutral, PublicKeyToken=31BF3856AD364E35">
    <section name="host" type="System.Web.WebPages.Razor.Configuration.HostSectio
n, System.Web.WebPages.Razor, Version=2.0.0.0, Culture=neutral, PublicKeyToken=31BF3
856AD364E35" requirePermission="false" />
    <section name="pages" type="System.Web.WebPages.Razor.Configuration.RazorPages
Section, System.Web.WebPages.Razor, Version=2.0.0.0, Culture=neutral, PublicKeyToken
=31BF3856AD364E35" requirePermission="false" />
</sectionGroup>
</configSections>
+
<system.web.webPages.razor>
    <pages pageBaseType="System.Web.Mvc.WebViewPage">
        <namespaces>
            <add namespace="LessonProject.Helper" />
        </namespaces>
    </pages>
</system.web.webPages.razor>

```



Добавляем пагинатор во View:

```
<div class="pagination">
    @Html.PageLinks (Model.PageNo, Model.CountPage, x => Url.Action("Index", new {page = x}))
</div>
```

Обратите внимание на конструкцию

`x => Url.Action("Index", new {page = x})`

Это делегат, который возвращает ссылку на страницу. А `Url.Action()` – формирует ссылку на страницу `/User/Index` с параметром `page = x`.

Вот что получилось (уменьшил количество вывода на странице до 5, чтобы образовалось больше страниц):

