

ЛЕКЦИЯ

ASP .Net MVC. 2 часть

Лектор Крамар Ю.М.

Содержание

1. Передача данных между контроллером и представлением
2. Представления
3. Строго типизированные представления
4. Мастер-представления
5. Razor
6. Формы. Вспомогательные методы HTML (хелперы)

ПЕРЕДАЧА ДАННЫХ МЕЖДУ КОНТРОЛЛЕРОМ И ПРЕДСТАВЛЕНИЕМ

Передача данных из контроллеров

Передача данных из контроллеров в представления:

- ▶ ViewData
- ▶ ViewBag
- ▶ TempData

Передача данных из контроллеров

ViewData:

- ▶ ViewData — это словарный объект, производный от TempDataDictionary;
- ▶ Используется для передачи данных из контроллера в соответствующее представление;
- ▶ Жизненный цикл ограничен текущим запросом;
- ▶ Если происходит redirect, значение ViewData превращается в null;
- ▶ Необходимо осуществлять приведение типов и проверять на null, чтобы избежать исключений

Передача данных из контроллеров

TempData:

- ▶ TempData — словарь данных, производный от TempDataDictionary, для их хранения в коротких по времени жизни сессиях;
- ▶ TempData используется для передачи данных из текущего запроса в последующий запрос (редирект из одной страницы на другую);
- ▶ Жизненный цикл TempData очень короткий, длится до полной загрузки целевого представления;
- ▶ Обязательно приведение типов и проверка на null;
- ▶ Используется только для хранения одноразовых сообщений, результатов исполнения операций и т.д.

Передача данных из контроллеров

ViewBag:

- ▶ ViewBag — динамическое свойство, представляет собой обертку вокруг ViewData, и также используется для передачи данных из контроллера в соответствующее представление;
- ▶ Жизненный цикл также ограничивается текущим запросом;
- ▶ При редиректе значение также оборачивается в null;
- ▶ Нет необходимости в приведении типов для получения данных

Передача данных из контроллеров

Запись во ViewData в контроллере:

```
public ActionResult SomeMethod()
{
    ViewData["Head"] = "Привет мир!";
    return View("SomeView");
}
```

Чтение ViewData в представлении:

```
...
<div>
    <h2>@ViewData["Head"]</h2>
</div>
...
```

Синтаксис использования TempData в представлении аналогичен

Передача данных из контроллеров

Запись во ViewBag в контроллере:

```
public ActionResult SomeMethod()
{
    ViewBag.NickName = personInfo.NickName;
    return View("SomeView");
}
```

Чтение ViewBag в представлении:

```
...
<div>
    <h2> @ViewBag.NickName </h2>
</div>
...
```

ViewBag в отличие от TempData и ViewData строго типизирован

Куки

Запись куки:

```
HttpContext.Response.Cookies["name"].Value = name;  
HttpContext.Response.Cookies["time"].Value = "1";
```

Чтение куки:

```
If ( HttpContext.Request.Cookies["id"] != null)  
    string id = HttpContext.Request.Cookies["id"].Value;
```

Удаление куки:

```
HttpCookie myCookie = new HttpCookie("name");  
myCookie.Expires = DateTime.Now.AddDays(-1d);  
esponse.Cookies.Add(myCookie);  
myCookie = new HttpCookie("time");  
myCookie.Expires = DateTime.Now.AddDays(-1d);  
Response.Cookies.Add(myCookie);
```

Сеанс (сессия)

Запись в сессию:

```
HttpContext.Session["NickName"] = personInfo.NickName;
```

Чтение из сессии:

```
If ( HttpContext. Session["NickName"] != null)  
    string nickName = (string)(HttpContext.Session["NickName"]);
```

Контексты контроллера

ControllerContext и HttpContext

Выполнение метода действия контроллера завершается созданием объекта, производного от класса ActionResult.

В объекте ActionResult доступны два контекста MVC:

ControllerContext и HttpContext:

- ▶ HttpContext описывает данные конкретного http-запроса, который обрабатывается приложением,
- ▶ ControllerContext описывает данные http-запроса по отношению к данному контролеру

```
context.HttpContext.Response.Write("<div style='width:100%;text-align:center;'> <h2>" +  
context.Controller.ViewData["Capture"].ToString() +  
"</h2></div>");
```

ПРЕДСТАВЛЕНИЯ

Представления

View:

Хотя работа приложения MVC управляется главным образом контроллерами, но непосредственно пользователю приложение доступно в виде представления, которое и формирует внешний вид приложения.

В ASP.NET MVC 4 представления представляют файлы с расширением `cshtml/vbhtml/aspx/ascx`, которые содержат код с интерфейсом пользователя, как правило, на языке `html`

Представления

Простое представление:



Generated code

```
@{
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>SomeView</title>
</head>
<body>
    <div>
        <h2>@ViewBag.Message</h2>
    </div>
</body>
</html>
```

Представления

Передача данных модели в представление

Для передачи данных из модели приложения можно использовать механизм ViewBag, что не является очень удобным

```
public ActionResult Index0()  
{  
    IEnumerable<Book> books = db.Books;  
    ViewBag.Books = books;  
    return View();  
}
```

```
@foreach (var b in ViewBag.Books)  
{  
    <tr>  
        <td><p>@b.Name</p></td>  
        <td><p>@b.Author</p></td>  
        <td><p>@b.Price</p></td>  
        <td><p><a  
href="/Home/Buy0/@b.Id">Купить</a></p></td>  
    </tr>  
}
```


СТРОГО ТИПИЗИРОВАННЫЕ ПРЕДСТАВЛЕНИЯ

Представления

Строго типизированные представления

Лучший вариант для передачи данных в представление вместо ViewBag – создание на основе модели строго типизированного представления в ручную или с использованием мастера

Add View

View name:
View1

View engine:
Razor (CSHTML)

☒ Create a strongly-typed view

Model class:
Book (BookStore.Models)

Scaffold template:
List

☒ Reference script libraries

☐ Create as a partial view

☒ Use a layout or master page:
[Empty text field]

(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID:
MainContent

Add **Cancel**

Представления

Передача данных модели в представление

```
public ActionResult Index0()  
{  
    IEnumerable<Book> books = db.Books;  
    return View(books);  
}
```

```
@model IEnumerable<BookStore.Models.Book>  
...  
@foreach (BookStore.Models.Book b in Model)  
    {  
        <tr>  
            <td><p>@b.Name</p></td>  
            <td><p>@b.Author</p></td>  
            <td><p>@b.Price</p></td>  
            <td><p><a  
href="/Home/Buy/@b.Id">Купить</a></p></td>  
        </tr>  
    }
```

Представления

Передача данных модели в представление

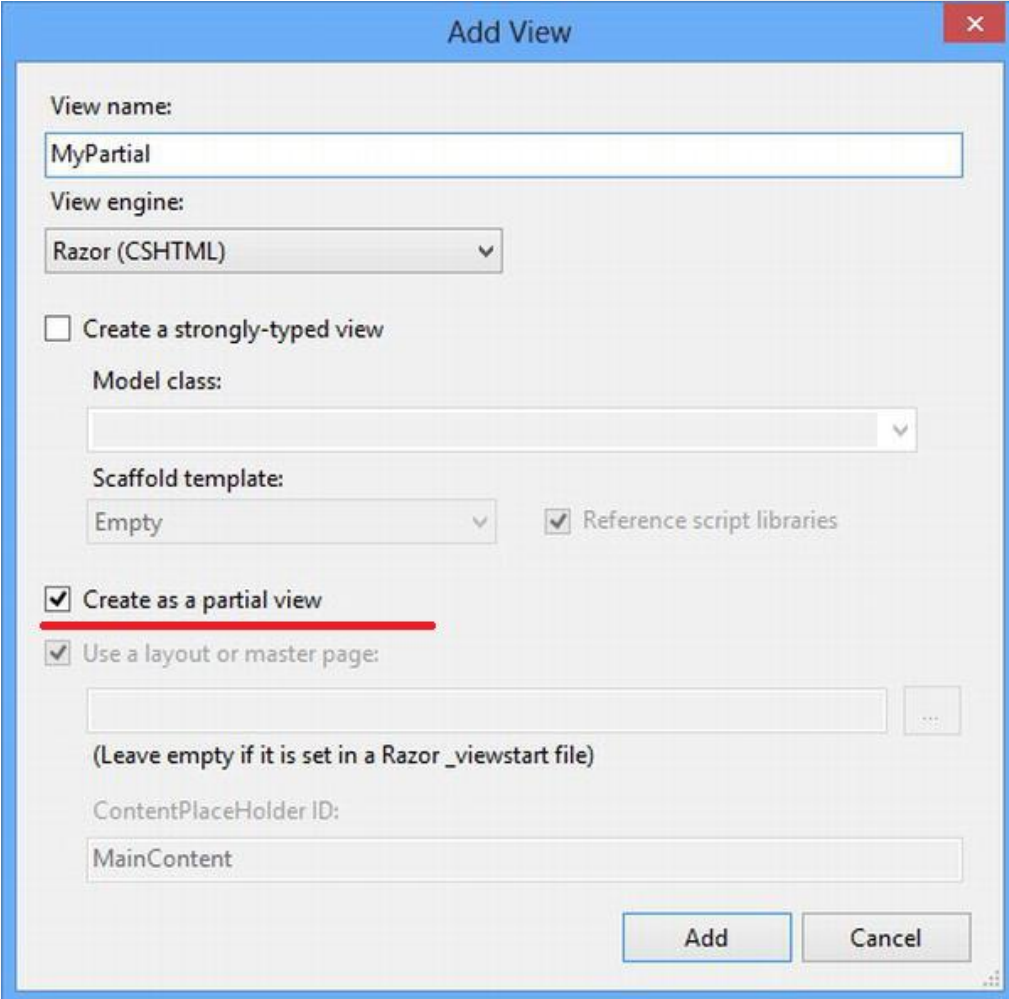
```
public ActionResult Index0()  
{  
    IEnumerable<Book> books = db.Books;  
    return View(books);  
}
```

```
@using BookStore.Models    // подключение namespace  
@model IEnumerable<Book>  
...  
@foreach (Book b in Model)  
    {  
        <tr>  
            <td><p>@b.Name</p></td>  
            <td><p>@b.Author</p></td>  
            <td><p>@b.Price</p></td>  
            <td><p><a  
href="/Home/Buy/@b.Id">Купить</a></p></td>  
        </tr>  
    }
```

Представления

Частичные представления

Используются для включения подсекции разметки в несколько представлений. Частичные представления могут содержать код, вспомогательные методы HTML и ссылки на другие частичные представления. Частичные представления не могут вызывать методы действий, поэтому их нельзя использовать для выполнения бизнес-логики.



The screenshot shows the 'Add View' dialog box with the following configuration:

- View name:** MyPartial
- View engine:** Razor (CSHTML)
- ☐ Create a strongly-typed view
- Model class:** (empty)
- Scaffold template:** Empty
- ☒ Reference script libraries
- ☒ Create as a partial view (highlighted with a red line)
- ☒ Use a layout or master page:
- ContentPlaceHolder ID:** MainContent

Buttons at the bottom: Add, Cancel.

Представления

Частичные представления:

```
// частичное представление
<div>
This is the message from the partial view.
@Html.ActionLink("This is a link to the Index action", "Index")
</div>
```

```
// использование частичного представления
@{ ViewBag.Title = "List";
Layout = null;
}

<h3>This is the /Views/Common/List.cshtml View
</h3>
@Html.Partial("MyPartial")
```

Представления

Строго типизированные частичные представления

```
//Частичное представление MyStronglyTypedPartial
@model IEnumerable<string>
<div>
This is the message from the partial view.
<ul>
@foreach (string str in Model)
{
<li>@str</li>
}
</ul>
</div>
```

```
//использование частичного представления
@{
ViewBag.Title = "List";
Layout = null;}
<h3>This is the /Views/Common/List.cshtml View</h3>
@Html.Partial("MyStronglyTypedPartial", new[] { "Apple",
"Orange", "Pear" })
```

МАСТЕР–ПРЕДСТАВЛЕНИЯ

Мастер–страницы

Мастер–страницы:

Для создания единообразного вида сайта применяются мастер–страницы. Мастер–страницы – это тоже представления.

Особенности использования мастер–страниц:

- ▶ `@RenderBody()` – плейсхолдер на мастер–странице, на месте вызова которого размещается контент представлений содержимого.
- ▶ Секция `Layout` – секция в представлении содержимого для подключения мастер–страницы

Мастер–страницы

Мастер–страница:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")"
rel="stylesheet" type="text/css" />
</head>

<body>
    <nav>
        <ul class="menu">
            <li>@Html.ActionLink("Главная", "Index", "Home")</li>
        </ul>
    </nav>
    @RenderBody()
    <footer>@RenderSection("Footer")</footer>
</body>
</html>
```

Мастер-страницы

Мастер-страница по умолчанию:

```
@{
    Layout = null;
}
или
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<!-- здесь остальное содержание -->
@section Footer {
    Все права защищены. Syte Corp. 2012.
}
```

Можно сохранить мастер-страницу как `_ViewStart.cshtml` в папке `Views`, удалив в представлениях секцию `Layout`. Она будет использоваться по умолчанию во всем проекте

RAZOR

Razor

View engine (Движок представлений):

- ▶ Движок представления обрабатывает ASP.NET-контент и ищет инструкции, как правило, для вставки динамического контента в выходные данные, отправленные браузеру. Microsoft поддерживает два вида движков:
- ▶ движок ASPX работает с тегами `<% и %>`, которые являлись основой развития ASP.NET в течение многих лет
- ▶ движок Razor, который работает с отдельными областями контента, обозначается символом `@`.

```
<% foreach(BookStore.Models.Book b in ViewBag.Books) { %>  
    <li>Книга: <%= b.Name %></li> <% } %>
```

```
@foreach(BookStore.Models.Book b in ViewBag.Books) {<li>Книга:  
b.Name </li>}
```

Razor

Движок представлений Razor :

Razor – это не новый язык, а лишь способ рендеринга представлений, который имеет определенный синтаксис для перехода от разметки html к коду C#.

Введение в MVC 3 движка Razor позволило уменьшить синтаксис при вызове кода C#, сделать код более "чистым".

Razor тесно связан с MVC, но с появлением ASP.NET 4.5 движок представления Razor также поддерживает ASP.NET Web Pages.

Razor

Главная цель представлений – визуализировать компоненты доменной модели как компоненты пользовательского интерфейса. Для этого нужно уметь добавлять в представления *динамический контент*.

Динамический контент генерируется во время выполнения и может быть разным для разных запросов.

Razor

Основы синтаксиса Razor :

Типы динамического контента:

- ▶ Код
- ▶ Вспомогательные методы HTML (HTML helpers)
- ▶ Секции
- ▶ Дочерние действия

Razor

Способы добавления динамического контента в представления

Способ	Применение
Код	Используется для создания небольших, независимых частей логики представления, например, операторы if или foreach. Это основной способ создания динамического контента в представлении, и на нем основаны некоторые другие подходы.
Секции	Используются для разбиения контента на блоки, которые будут вставлены в макет в определенных местах.

Razor

Способ	Применение
Дочерние действия	Используются для создания повторно используемых элементов управления UI и виджетов, в которых необходима бизнес-логика. Дочернее действие вызывает метод действия, визуализирует представление и внедряет результат в поток ответа.
Вспомогательные методы HTML	Используются для создания одного элемента HTML или небольшой коллекции элементов, обычно на основании данных модели представления или объекта ViewData. Можно использовать встроенные вспомогательные методы MVC Framework, можно также создавать свои собственные. (Вспомогательные методы HTML рассматриваются позже)

Razor

Код:

Использование синтаксиса Razor характеризуется тем, что перед выражением кода стоит знак @, после которого осуществляется переход к коду C#. Существуют два типа переходов к простому динамическому контенту:

- ▶ к выражениям кода;
- ▶ к блоку кода.

Например:

```
// использование объекта
<p>@b.Name</p>

// использование класса и метода
<h3>@DateTime.Now.ToShortTimeString()</h3>
```

Razor

Код:

синтаксис Razor поддерживает как использование отдельных операторов C#, так и написание целых блоков кода

Например:

```
// использование оператора
@foreach (BookStore.Models.Book b in Model)
{
    <p>@b.Name</p>
}
// создание блока кода
@{
    string head = "Привет мир!!!";
    head = head + " Добро пожаловать на сайт!";
}
<h3>@head</h3>
```

Razor

Секции:

Движок Razor поддерживает концепцию секций, которые позволяют выделять различные области в макете. Секции Razor позволяют разбивать представление на части и контролировать то, где они внедряются в макет:

```
@section Header
{ <div class="view">
@foreach (string str in new[] { "Home", "List", "Edit" })
@Html.ActionLink(str, str, null, new { style = "margin: 5px" })
    </div>
} ...
@section Footer
{ <div class="view">
This is the footer
</div>
}
```

Razor

Секции:

```
<body>

@RenderSection("Header")
<div class="layout">
This is part of the layout
</div>

@RenderBody()
  <div class="layout">
This is part of the layout
</div>

@RenderSection("Footer")
<div class="layout">
This is part of the layout
</div>
</body>
```

Razor

Секции:

```
// тестирование наличия секций
<body>

@if (IsSectionDefined("Footer"))
{   @RenderSection("Footer")}
Else
{   <h4>This is the default footer</h4>
}
```

```
// или задание секции как необязательной
@RenderSection("scripts", false)
```

Razor

Дочерние действия:

```
// дочернее действие
public class HomeController : Controller {
    [ChildActionOnly]
    public ActionResult Time()
    {
        return PartialView(DateTime.Now);
    }
}
```

```
// частичное представление дочернего действия
@model DateTime
<p>The time is: @Model.ToShortTimeString()</p>
```

```
// вызов дочернего действия из представления
@{
    Layout = null;
}
...
@Html.Action("Time")
```


Razor

Дочерние действия:

```
// дочернее действие с параметром
public class HomeController : Controller {
    [ChildActionOnly]
    public ActionResult Time(DateTime time)
    {
        return PartialView(time);
    }
}
```

```
// частичное представление дочернего действия
@model DateTime
<p>The time is: @Model.ToShortTimeString()</p>
```

```
// вызов дочернего действия из представления
@{
    Layout = null;
}
...
@Html.Action("Time", new { time = DateTime.Now })
```

ФОРМЫ. ВСПОМОГАТЕЛЬНЫЕ МЕТОДЫ HTML (ХЕЛПЕРЫ)

Вспомогательные методы HTML (хелперы)

Представления используют разметку html для визуализации содержимого. Для упрощения кода визуализации ASP.NET MVC предоставляет вспомогательные методы (HTML-хелперы), позволяющие генерировать html-код.

```
@helper BookList(IEnumerable<BookStore.Models.Book> books)
{
    <ul>
        @foreach (BookStore.Models.Book b in books)
        {
            <li>@b.Name</li>
        }
    </ul>
}
```

```
<h3>Список книг</h3>
@BookList(ViewBag.Books)
<!-- если используется строго типизированное представление -->
@BookList(Model)
```

Вспомогательные методы HTML (хелперы)

Можно создать новый класс с HTML-хелпером как методом расширения, расширив набор классов стандартной библиотеки `HtmlHelper`.

```
public static class ListHelper{
    public static MvcHtmlString CreateList(this HtmlHelper
html, string[] items)
    {
        TagBuilder ul = new TagBuilder("ul");
        foreach (string item in items)
        {
            TagBuilder li = new TagBuilder("li");
            li.SetInnerText(item);
            ul.InnerHtml += li.ToString();
        }
        return new MvcHtmlString(ul.ToString());
    }
}
```

```
@Html.CreateList(cities)
<!-- или можно вызвать так -->
@ListHelper.CreateList(Html, countries)
```

Формы

Для передачи данных на сервер используется HTML-форма. Ее можно создать средствами классического набора HTML-тегов

```
<form method="post" action="/Home/Buy">
  <input type="hidden" value="@ViewBag.BookId" name="BookId" />
  <table>
    <tr><td><p>Введите свое имя </p></td>
      <td><input type="text" name="Person" /> </td></tr>
    <tr><td><p>Введите адрес :</p></td>
      <td><input type="text" name="Address" /> </td></tr>
    <tr><td><input type="submit" value="Отправить" /> </td>
      <td></td></tr>
  </table>
</form>
```

Формы

[Главная](#)

Форма оформления покупки

Введите свое имя

Введите адрес :

Все права защищены. Syte Corp. 2014.

Формы и HTML-хелперы

Встроенными HTML-хелперами `BeginForm/EndForm` можно создать ту же самую форму. На странице может быть несколько форм.

```
<!-- или @using(Html.BeginForm()) если у метода есть обе версии
HttpGet и HttpPost -->

@using(Html.BeginForm("Buy", "Home", FormMethod.Post))
{
    <input type="hidden" value="@ViewBag.BookId" name="BookId" />
    <table>
        <tr><td><p>Введите свое имя </p></td>
            <td><input type="text" name="Person" /> </td></tr>
        <tr><td><p>Введите адрес :</p></td>
            <td><input type="text" name="Address" /> </td></tr>
        <tr><td><input type="submit" value="Отправить" /> </td>
            <td></td></tr>
    </table>
}
```

Метод `BeginForm` принимает в качестве параметров имя метода действия и имя контроллера, а также тип запроса.

Формы и HTML-хелперы

В MVC определен широкий набор хелперов ввода практически для каждого html-элемента. Вне зависимости от типа все базовые html-хелперы используют как минимум два параметра:

- ▶ первый параметр применяется для установки значений для атрибутов id и name,
- ▶ второй параметр – для установки значения атрибута value

```
<p>Введите адрес :</p>  
@Html.TextBox("Address", "Введите адрес")
```

Вместо

```
<p>Введите адрес :</p>  
<input type="text" name="Address" />
```


Формы и HTML-хелперы

Еще html-хелперы

```
<p> Label</p>
@Html.Label("Name")
<p> TextArea</p>
@Html.TextArea("text", "TextArea <br/> Text", 5, 50, null)
<p> Hidden</p>
@Html.Hidden("BookId", "2")
<p> Password</p>
@Html.Password("UserPassword", "val")
<p> RadioButton</p>
@Html.RadioButton("color", "red")
<span>красный</span> <br />
@Html.RadioButton("color", "blue")
<span>синий</span> <br />
@Html.RadioButton("color", "green", true)
<span>зеленый</span>
<p> CheckBox</p>
@Html.CheckBox("Enable", false)
```

Строго типизированные хелперы

Кроме базовых хелперов в ASP.NET MVC имеются их двойники – строго типизированные хелперы.

Этот вид хелперов принимает в качестве параметра лямбда-выражение, в котором указывается то свойство модели, к которому должен быть привязан данный хелпер.

Строго типизированные хелперы могут использоваться только в строго типизированных представлениях, а тип модели, которая передается в хелпер, должен быть тем же самым, что указан для всего представления с помощью директивы `@model`.

Строго типизированные хелперы

Модель и строго типизированное представление

```
public class Purchase
{
    public int PurchaseId { get; set; }
    public string Person { get; set; }
    public string Address { get; set; }
    public int BookId { get; set; }
    public DateTime Date { get; set; }
}
```

```
<form method="post" action="">
    <input type="hidden" value="@ViewBag.BookId" name="BookId" />
    <table>
        <tr><td><p>Введите свое имя </p></td>
            <td><input type="text" name="Person" />
        </td></tr>
        <tr><td><p>Введите адрес :</p></td><td>
            <input type="text" name="Address" />
        </td></tr>
        <tr><td><input type="submit" value="Отправить" />
            <td></td></tr>
    </table>
</form>
```

Строго типизированные хелперы

Модель и строго типизированные хелперы

```
@model BookStore.Models.Purchase

<div>
    <h3>Форма оформления покупки</h3>
    @using(Html.BeginForm("Buy", "Home", FormMethod.Post))
    {
        @Html.HiddenFor(m=>m.BookId)
        @Html.LabelFor(m => m.Person, "Введите свое имя")
        <br />
        @Html.TextBoxFor(m=>m.Person)
        <br /><br />
        @Html.LabelFor(m => m.Address, "Введите адрес")
        <br />
        @Html.TextBoxFor(m=>m.Address)
        <p><input type="submit" value="Отправить" /></p>
    }
</div>
```

Шаблонные хелперы

Шаблонные хелперы, исходя из свойства модели, генерируют тот элемент html, который наиболее подходит данному свойству, исходя из его типа и метаданных.

- ▶ **Display** – создает элемент разметки для отображения значения указанного свойства модели: `Html.Display("Name")`
- ▶ **DisplayFor** – строго типизированный аналог хелпера `Display`: `Html.DisplayFor(m => m.Name)`
- ▶ **Editor** – создает элемент разметки для редактирования указанного свойства модели: `Html.Editor("Name")`
- ▶ **EditorFor** – строго типизированный аналог хелпера `Editor`: `Html.EditorFor(m => m.Name)`
- ▶ **DisplayText** – создает выражение для указанного свойства модели в виде простой строки: `Html.DisplayText("Name")`
- ▶ **DisplayTextFor** – строго типизированный аналог хелпера `DisplayText`: `Html.DisplayTextFor(m => m.Name)`
- ▶ **DisplayForModel** – создает поля для чтения для всех свойств модели: `Html.DisplayForModel()`
- ▶ **EditorForModel** – создает поля для редактирования для всех свойств модели: `Html.EditorForModel()`

Шаблонные хелперы

Шаблонный хелпер `DisplayForModel`, исходя из свойств модели, генерирует набор элементов html, для всех свойств модели представления.

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

@model BookStore.Models.Book

<h2>Книга № @Model.Id</h2>
@Html.DisplayForModel()
```

Исходный код проектов

BookStore

- Проект MVC_empty

Создается по шаблону «Пустой mvc-проект». Добавлен контроллер Home с методом Index. Демонстрируется структура mvc-проекта

- Проект BookStore

Добавлены классы модели (классы сущностей, контекста и инициализатора контекста), контроллер Home, представления к нему. Для представлений создана мастер-страница. Для сущности Book добавлена валидация модели. Стили валидации определены в Content/Site.css



MVC_demos.rar

ASP .Net MVC. 2 часть

Ресурсы:

<http://smarly.net/pro-asp-net-mvc-4>

<http://www.metanit.com/sharp/mvc5>