Представления

ASP.NET MVC 4.0

2013

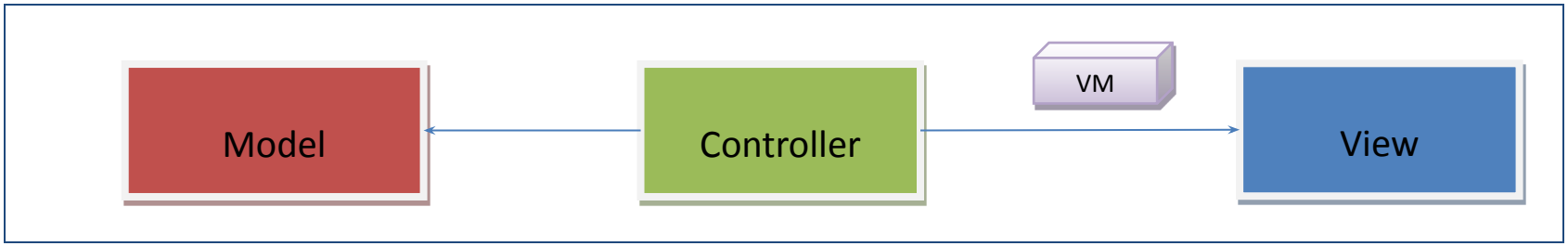
План

1. Модель представления
2. Макеты страниц
3. Частичные представления
4. Вспомогательные методы в шаблонах представлений
5. Пользовательские вспомогательные методы

Модель представления

Способы обмена данными между контроллером и представлением:

1) ViewData: ViewDataDictionary
2) ViewData.Model
3) ViwBag: dynamic



Модель представления – это данные, структура которых воспроизводит структуру представления.

Пример модели представления

```
public class LoginModel
{
    public string UserName { get; set; }

    public string Password { get; set; }

    public bool RememberMe { get; set; }
}
```

Модель представления для аутентификации пользователя.

```
 @using ArtMuseum.Models
 @model LoginModel
...

 @Model.UserName
```

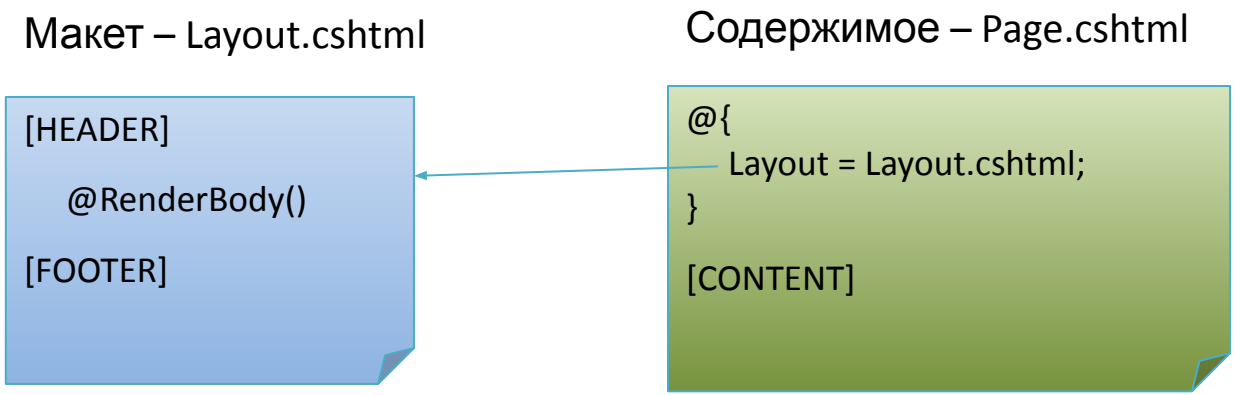
В типизированном представлении тип модели объявляется директивой @model. Сам объект модели доступен через свойство представления Model.

```
  <system.web.webPages.razor>
    <pages ...>
      <namespaces>
        <add namespace="ArtMuseum.Models" />
      </namespaces>
    </pages>
  </system.web.webPages.razor>
```

Пространство имен можно указать в директиве @using или в файле web.config в папке Views.

Макеты страниц

Два представления могут находиться в отношении макет – содержимое (мастер-страница – содержимое).



Результат, который получит клиент:

[HEADER]

[CONTENT]

[FOOTER]

Пример

Макет

```
<!DOCTYPE html>
<html>
    <head>
        <title>Simple Site</title>
    </head>
    <body>

        <div id="header">
            <a href="/">Home</a>
            <a href="/About>About</a>
        </div>

        <div id="body">
            @RenderBody()
        </div>

    </body>
</html>
```

Страница

```
@{
    Layout = "SiteLayout.cshtml";
}

<h1>About This Site</h1>

<p>
    This is some content that will make up the "about"
    page of our web-site.  We'll use this in conjunction
    with a layout template.  The content you are seeing here
    comes from the Home.cshtml file.
</p>
<p>
    And obviously I can have code in here too. Here is the
    current date/time: @DateTime.Now
</p>
```

Секции

В макете можно предусмотреть не одно, а несколько мест для вставки содержимого. Эти места отмечаются кодом

@RenderSection(имя_секции, обязательность)

На странице вставляемое содержимое размещается в секциях.

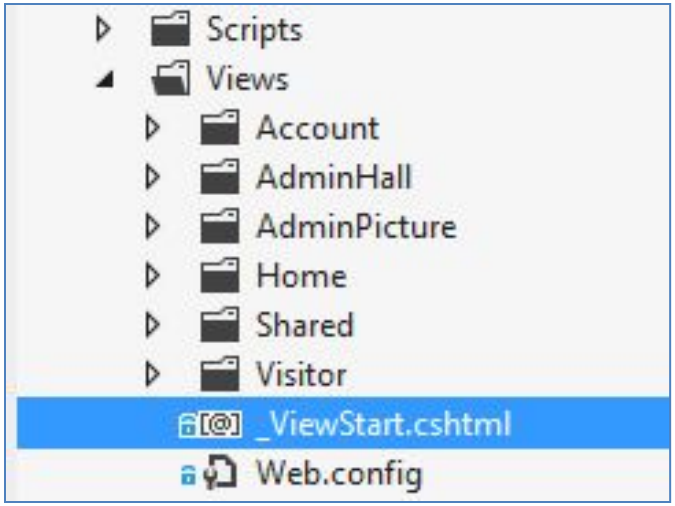
@section имя_секции{ ……. }

```
<!DOCTYPE html>
<html>
    <head>
        <title>Simple Site</title>
    </head>
    <body>

        <div id="header">
            <a href="/">Home</a>
            <a href="/About">About</a>
        </div>

        <div id="left-menu">
            @RenderSection("menu", optional:true)
        </div>

        <div id="body">
            @RenderBody()
        </div>

        <div id="footer">
            @RenderSection("footer", optional:true)
        </div>

    </body>
</html>
```

```
<h1>About This Site</h1>

<p>
    This is some content that will make up the "about"
    page of our web-site.  We'll use this in conjunction
    with a layout template.  The content you are seeing here
    comes from the Home.cshtml file.
</p>
<p>
    And obviously I can have code in here too. Here is the
    current date/time: @DateTime.Now
</p>

@section menu {
    <ul id="sub-menu">
        <li>About Item 1</li>
        <li>About Item 2</li>
    </ul>
}

@section footer {
    <p>This is my custom footer for Home</p>
}
```

Неявное указание макета

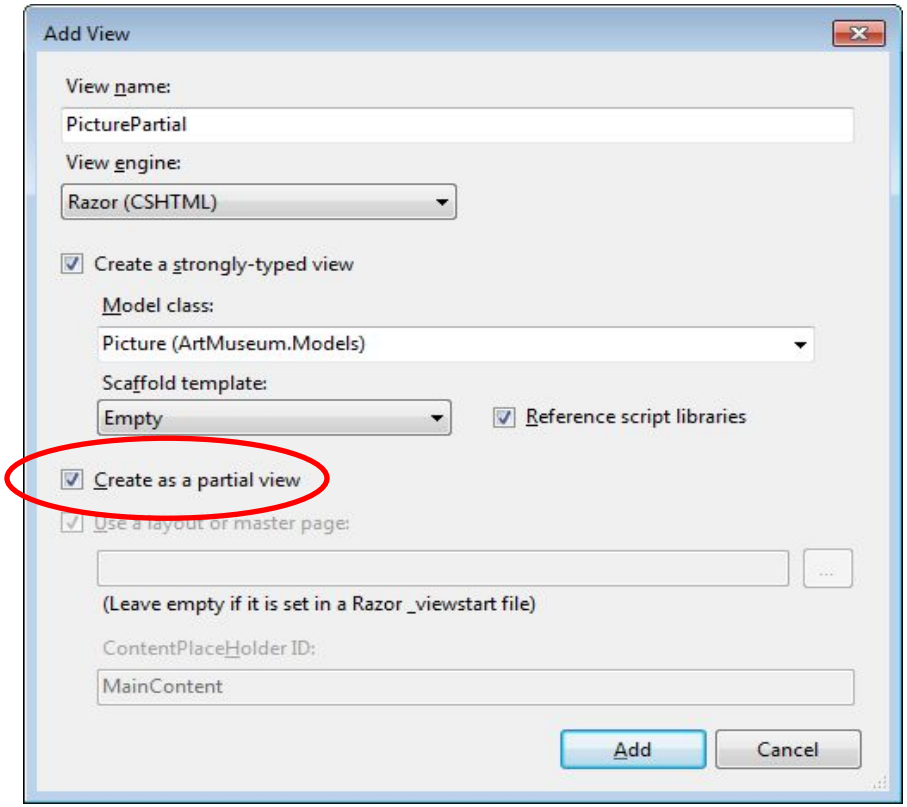
Обычно несколько представлений одного контроллера пользуются общим макетом. Его можно указать опосредованно в файле _ViewStart.cshtml, который находится в корневом каталоге представлений.



Содержание _ViewStart.cshtml:

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

Частичные представления



Функционально частичное представление аналогично User Control.

По умолчанию ч.п. создается в папке ~/Views/Shared.

Макет в ч.п. не указывается.

Вызов на странице:

```
@Html.Partial("PicturePartial", new ArtMuseum.Models.Picture { Name = «Богатырская застава" })
```

Имя представления

Объект модели

Вспомогательные методы

Вспомогательные методы (helpers – помощники) вызываются из шаблонов страниц и возвращают html-код.

Вспомогательные методы расширяют тип HtmlHelper. Представления имеют свойство Html типа HtmlHelper,через которое можно получить доступ к любым помощникам.

Код помощников содержится в классе System.Web.Mvc.Html в форме статических методов.

```
1:  Html.TextBox("Name")            // MVC 2.0

2:  Html.TextBoxFor(m => m.Name)    // MVC 3.0
```

Параметром помощника может быть строка (как в MVC 2.0) или делегат (как в MVC 3.0). Делегат позволяет выполнить ранний контроль типа.

Оба помощника возвращают одну и ту же строку:

```
<input id="Name" name="Name" type="textbox" />
```

Значения элементов управления

При синтезе страницы значения элементов управления черпаются из двух источников:

1) коллекция ViewData.ModelState;
2) параметра метода-помощника.

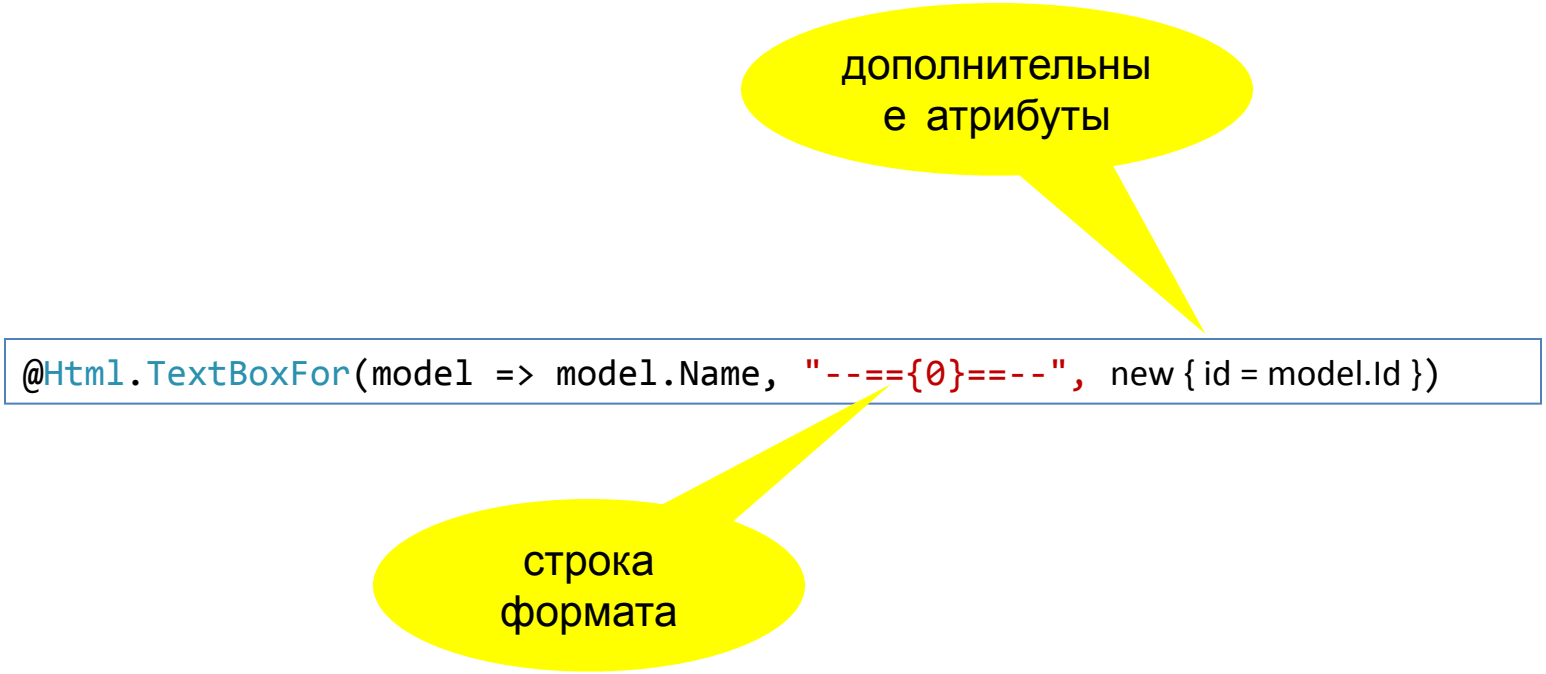
В коллекцию ModelState собираются значения, введенные пользователем в процессе работы с формой.

Параметр помощника задает первоначальное значение элемента управления.

```
@Html.TextBoxFor(model => model.Name)
```

Атрибуты элемента управления

Любые дополнительные атрибуты элемента управления можно задать в виде параметра помощника типа object.



Формы

Есть два способа сгенерировать код формы. Во втором способе закрывающий тэг </form> создается методом Dispose().

```
@{Html.BeginForm("Search", "Home", FormMethod.Get, new { target = "_blank",
                                    class="editForm", data_validatable=true } }
    <input type="text" name="Title" />
    <input type="submit" value="Search" />
@{Html.EndForm();}
```

```
@using (Html.BeginForm("Search", "Home", FormMethod.Get, new { target = "_blank",
                                        class="editForm", data_validatable=true }))
{
    <input type="text" name="Title" />
    <input type="submit" value="Search" />
}
```

```
<form action="/Home/Search" method="get"  target="_blank"
      class="editForm", data_validatable=true >
   …
</form>
```

Перегруженные помощники

```
public static class LabelExtensions
{
1   public static MvcHtmlString Label(this HtmlHelper html, string expression);

2   public static MvcHtmlString Label(this HtmlHelper html, string expression,
       object htmlAttributes);

3   public static MvcHtmlString Label(this HtmlHelper html, string expression,
        string labelText, object htmlAttributes);

4   public static MvcHtmlString LabelFor<TModel, TValue>(this HtmlHelper<TModel> html,
        Expression<Func<TModel, TValue>> expression);
}
```

```
1: Html.Label("Author")           // MVC 2.0

4: Html.LabelFor(m => m.Author)   // MVC 3.0
```

Популярные помощники

HTML helper	Description
DisplayFor	Возвращают соотвующую html-разметку для каждого свойства объекта, представленного лямбда-выражением. Как правило, задается объект простого, а не составного типа, поэтому порождается лишь один html элемент. Необходимые данные могут быть получены из атрибутов или метаданных модели.
EditorFor	
CheckBoxFor	
DropDownListFor	
HiddenFor	
LabelFor	
ListBoxFor	
PasswordFor	
RadioButtonFor	
TextAreaFor	
TextBoxFor	
ValidateFor	
ValidationMessageFor	

Интегральные помощники

В связи с автоматической генерацией шаблонов в MVC появились крупнопанельные конструкции вроде EditorForModel() или DisplayForModel().

Они годны лишь для временного употребления, когда надо быстро получить прототип, например.

```
<h2>Edit</h2>

@using (Html.BeginForm("Edit", "Home"))
{
     @Html.EditorForModel()

     <button type="submit">Submit</button>
}
```

Объявление собственных помощников

В качестве примера объявим помощник для генерации ссылки с подтверждением.

Ссылка с подтверждением – это такая ссылка, что если по ней кликнуть, в браузере появится диалоговое окно с двумя кнопками и сообщением message. Переход по url произойдет, если нажать кнопку OK. Параметр text - текст ссылки.

```
namespace HelpersMVC.Views.Shared
{
    public static class MyHelpers
    {
        public static MvcHtmlString ConfirmingLink<TModel>(
            this HtmlHelper<TModel> helper,
            string url, string text, string prompt = "Really?")
        {
            text = HttpUtility.HtmlEncode(text);
            string html = string.Format(
                "<a href='{0}' onclick='return confirm({1})'>{2}</a>",
                url, prompt, text);
            return MvcHtmlString.Create(html);
        }
    }
}
```

Вызов помощника

```
@using HelpersMVC.Views.Shared

@Html.ConfirmingLink("https://www.google.com.ua/", "Go to Google?")
```

Пространство имен самодельного помощника импортируем при помощи директивы @using

Импорт пространства имен для всех шаблонов некоторой папки можно сделать в локальном файле web.config.

```
<system.web.webPages.razor>
  <pages pageBaseType="System.Web.Mvc.WebViewPage">
    <namespaces>
      <add namespace="System.Web.Mvc" />
      <add namespace="System.Web.Mvc.Ajax" />
      .
      .
      <!-- etc -->
    </namespaces>
  </pages>
</system.web.webPages.razor>
```

Помощники Razor

Генерировать html-код естественней и проще при помощи шаблонов Razor.

Для этого каталоге App_Code разместим файл MyHelpers.cshtml

```
@using System.Web.Mvc.Html
@using System.Web.Mvc

@helper ConfirmingLink(string url, string text, string message="Really?")
{
    <a href='@url' onclick='return confirm("@message")'>@text</a>
}
```

Вызываются помощники Razor без указания пространства имен.

```
@MyHelpers.ConfirmingLink("https://www.google.com.ua/", "Go to Google?")
```

Самостоятельно

База данных содержит записи календаря и записи личной информации.

Запись календаря содержит информацию о событии:
дата, время начала, время окончания, название, тип события, список участников

Личная запись содержит имя, фамилию и дату рождения.

Создать страницы для поддержки CRUD-операций с календарем и личными записями.