

ASP MVC

v^*

Представление

Частичные представления

Частичные представления - это отдельные файлы, которые содержат в себе фрагменты html разметки, которые могут быть включены в другие представления.

Частичные представления могут быть вызваны только полноценными представлениями, на свойстве Html вызывается метод Partial, которому в качестве аргумента передается имя частичного представления, и при визуализации страницы в точку вызова вспомогательного метода Partial будет вставлена разметка из данного частичного представления, имя которого передано в качестве аргумента в метод Partial.

Основная задача Partial View частичного представления – является избавления от дублирования кода.



Частичные представления

Частичные представления могут находиться

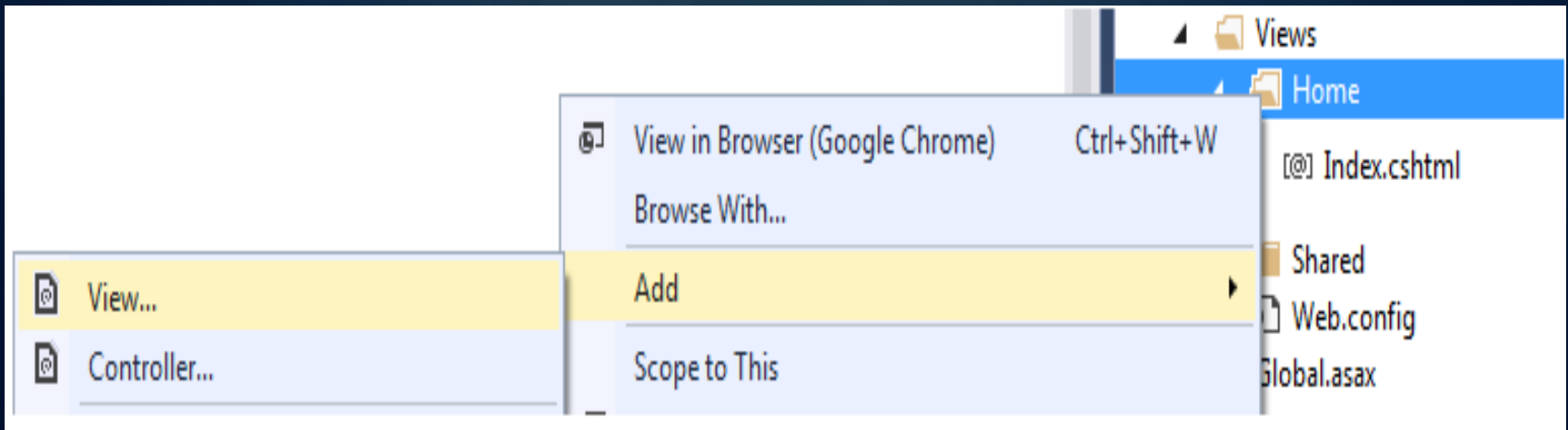
1. В папке Shared, которая расположена в папке представлений View, если ими могут воспользоваться все представления нашего приложения.
2. В папке соответствующего контроллера, если данным частичным представлением могут воспользоваться только представления данного контроллера.



Example

Частичные представления

Частичные представления, *создаются НЕ В МЕТОДЕ действия*, а в папке, например в папке Home, если мы хотим чтобы только представления контроллера Home могли воспользоваться данным частичным представлением.



Частичные представления

Add View

View name:
_SimpleList

View engine:
Razor (CSHTML)

☐ Create a strongly-typed view
Model class:

Scaffold template:
Empty ☒ Reference script libraries

☒ Create as a partial view

☒ Use a layout or master page:
(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID:
MainContent

Add Cancel

_SimpleList.cshtml

```
<ul>  
  <li>Item 01</li>  
  <li>Item 02</li>  
  <li>Item 03</li>  
  <li>Item 04</li>  
  <li>Item 05</li>  
</ul>
```

Частичные представления

Index.cshtml* [icon] [X]

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Использование частичных представлений</title>
</head>
<body>
    <div>
        @*Визуализация частичного представления (Views/Home/_SimpleList.cshtml)*@
        @Html.Partial("_SimpleList")
    </div>
</body>
</html>
```



Example

01_PartialViews

2015 © EPAM Systems

7

Частичные представления

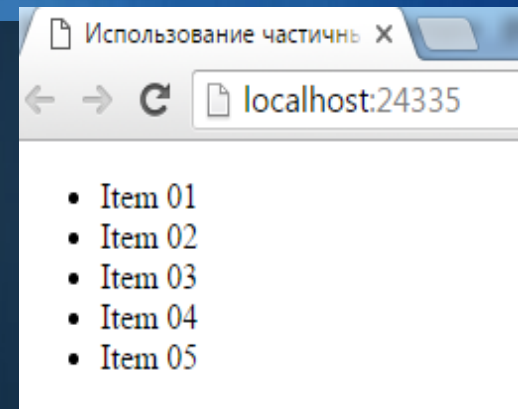
```
<!DOCTYPE html>

<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Использование частичных представлений</title>
</head>
<body>
  <div>

<ul>
  <li>Item 01</li>
  <li>Item 02</li>
  <li>Item 03</li>
  <li>Item 04</li>
  <li>Item 05</li>
</ul>

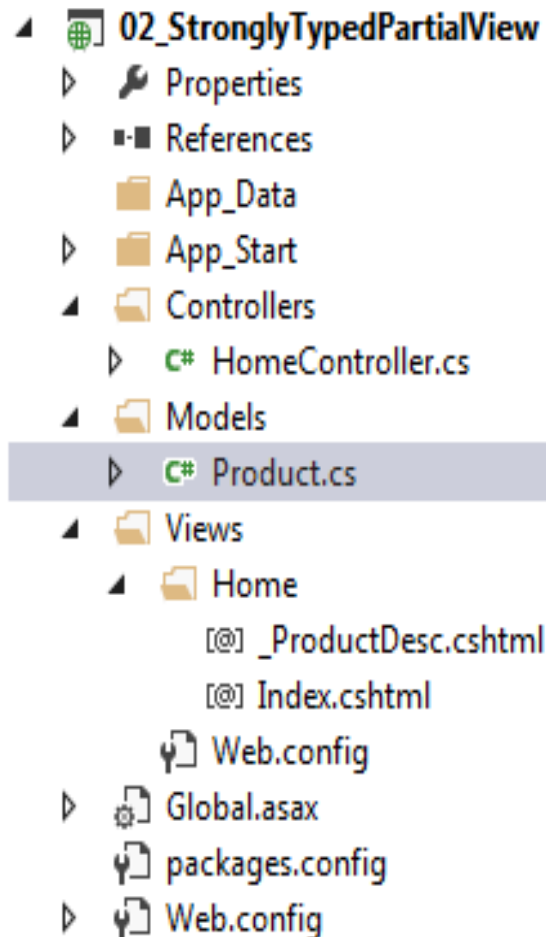
  </div>

</body>
</html>
```



Частичные представления

```
namespace _02_StronglyTypedPartialView.Models
{
    5 references
    public class Product
    {
        3 references
        public int Id { get; set; }
        3 references
        public string Name { get; set; }
        3 references
        public decimal Price { get; set; }
    }
}
```



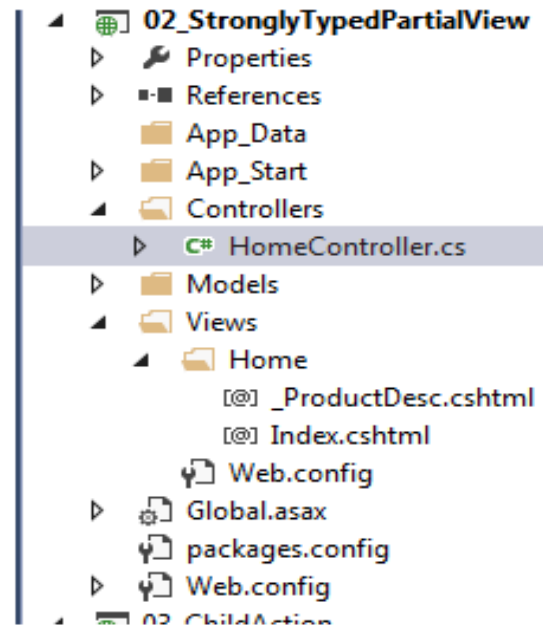
Example

Частичные представления

```
namespace _02_StronglyTypedPartialView.Controllers
{
    0 references
    public class HomeController : Controller
    {
        0 references
        public ActionResult Index()
        {
            List<Product> products = new List<Product>();

            products.Add(new Product()
            {
                Id = 1,
                Name = "Item 1",
                Price = 10
            });
            products.Add(new Product()
            {
                Id = 2,
                Name = "Item 2",
                Price = 5
            });
            products.Add(new Product()
            {
                Id = 3,
                Name = "Item 3",
                Price = 50
            });

            return View(products);
        }
    }
}
```



Example

02_StronglyTypedPartialView

2015 © EPAM Systems

10

Частичные представления

Add View

View name:
_ProductDesc

View engine:
Razor (CSHTML)

☒ Create a strongly-typed view

Model class:
Product (_02_StronglyTypedPartialView.Models)

Scaffold template:
Empty ☒ Reference script libraries

☒ Create as a partial view

☒ Use a layout or master page:

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

ContentPlaceHolder ID:
MainContent

Add Cancel

```
_ProductDesc.cshtml @model _02_StronglyTypedPartialView.Models.Product

<div>
    <ul>
        <li># @Model.Id</li>
        <li>Name: @Model.Name</li>
        <li>Price: @Model.Price $</li>
    </ul>
</div>
```



Example

02_StronglyTypedPartialView

2015 © EPAM Systems

11

Частичные представления

```
Index.cshtml*  ▢ ✕  
@model IEnumerable<_02_StronglyTypedPartialView.Models.Product>  
@{  
    Layout = null;  
}  
  
<!DOCTYPE html>  
  
<html>  
<head>  
    <meta name="viewport" content="width=device-width" />  
    <title>Strongly Typed Partial Views</title>  
</head>  
<body>  
    <div>  
        @foreach (var item in Model)  
        {  
            // Второй параметр метода Partial - модель,  
            //которая будет доступна в частичном представлении  
            @Html.Partial("_ProductDesc", item)  
        }  
    </div>  
  
</body>  
</html>
```



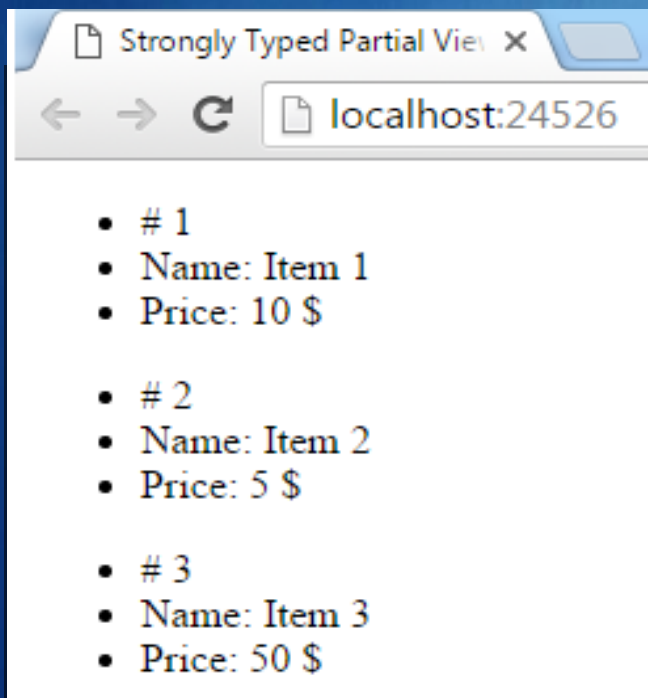
Example

02_StronglyTypedPartialView

2015 © EPAM Systems

12

Частичные представления



```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Strongly Typed Partial Views</title>
</head>
<body>
  <div>
    <div>
      <ul>
        <li># 1</li>
        <li>Name: Item 1</li>
        <li>Price: 10 $</li>
      </ul>
    </div>

    <div>
      <ul>
        <li># 2</li>
        <li>Name: Item 2</li>
        <li>Price: 5 $</li>
      </ul>
    </div>

    <div>
      <ul>
        <li># 3</li>
        <li>Name: Item 3</li>
        <li>Price: 50 $</li>
      </ul>
    </div>
  </div>
</body>
</html>
```



Частичные представления

В приложениях часто используются одни и те же фрагменты тегов Razor и HTML-разметки в нескольких представлениях. Чтобы не дублировать контент, можно использовать **частичные представления**.

Частичные представления представляют собой отдельные файлы, которые содержат фрагменты кода с тегами и разметкой и могут быть включены в другие представления.

Можно также создать **строго типизированное частичное представление**, а затем передавать в него объекты моделей представлений, которые оно будет визуализировать.



Example

Дочерние действия

Дочерние действия – это методы действий, которые вызываются из представления. Они позволяют избежать дублирования логики контроллера, которую необходимо использовать в приложении несколько раз. Дочерние действия так же относятся к действиям, как частичные представления – к представлениям.

Дочерние действия чаще всего используются для отображения какого-либо управляемого данными виджета, который должен появляться на нескольких страницах и содержит данные, не относящиеся к основному действию (например, управляемое данными меню навигации, без необходимости поставлять данные о категориях навигации непосредственно от каждого действия метода)



Example

Дочерние действия

Дочерние действия:

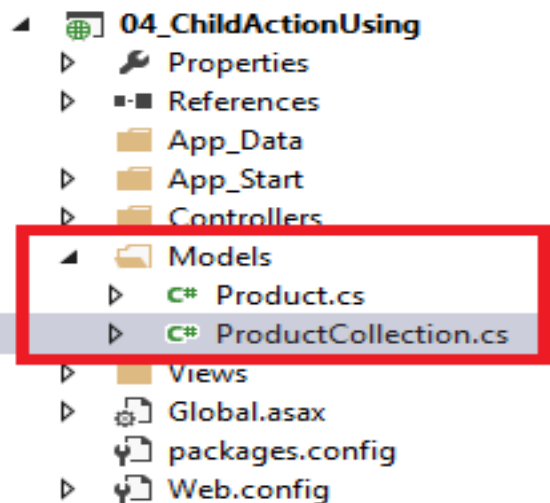
1. **должны быть помечены атрибутом** [ChildActionOnly] - это означает, что таблица системы маршрутизации не позволит запустить данный метод, при запросе на него, т.е. из вне его запустить нельзя, но мы можем его выполнять как дочернее действие представления.
2. **могут вызываться только из представлений**, чтобы перенести выполнение какой-то бизнес логики в контроллер. Т.к. задача именно выполнять бизнес логику, т.е. получить какие-то данные их обработать и вернуть в представление, которое и отправляет их в ответ клиенту в виде html разметки.
3. **могут возвращать либо частичные представления, либо контент**, т.е. строку или какой-то кусок html кода, т.к. данные методы действия вызываются в представлении, а в представление может быть вставлено только часть html разметки, что из себя и представляет частичное представление.



Дочерние действия

```
public class Product
{
    1reference
    public int Id { get; set; }
    1reference
    public string Name { get; set; }
    1reference
    public decimal Price { get; set; }
}
```

```
public class ProductCollection
{
    1reference
    public static List<Product> All
    {
        get
        {
            List<Product> products = new List<Product>();
            for (int i = 0; i < 20; i++)
            {
                products.Add(new Product()
                {
                    Id = i + 1,
                    Name = "Item Name " + i,
                    Price = (i + 1) * 2
                });
            }
            return products;
        }
    }
}
```



Example

Дочерние действия

```
public class HomeController : Controller
{
    References
    public ActionResult Index()
    {
        return View();
    }

    [ChildActionOnly]
    References
    public ActionResult ShowTable(int numberOfRows = 5)
    {
        IEnumerable<Product> products = ProductCollection.All.Take(numberOfRows);
        return PartialView("_Table", products);
    }
}
```



Дочерние действия

_Table.cshtml* ▢ ✕

```
@model IEnumerable<_04_ChildActionUsing.Models.Product>
```

```
<table border="1">
  <tr>
    <th> # </th>
    <th> Name </th>
    <th> Price </th>
  </tr>

  @foreach (var item in Model)
  {
    <tr>
      <td> @item.Id </td>
      <td> @item.Name </td>
      <td> $ @item.Price </td>
    </tr>
  }
</table>
```



Example

Дочерние действия

Index.cshtml* [icon] [X]

```
@{
    Layout = null;
}

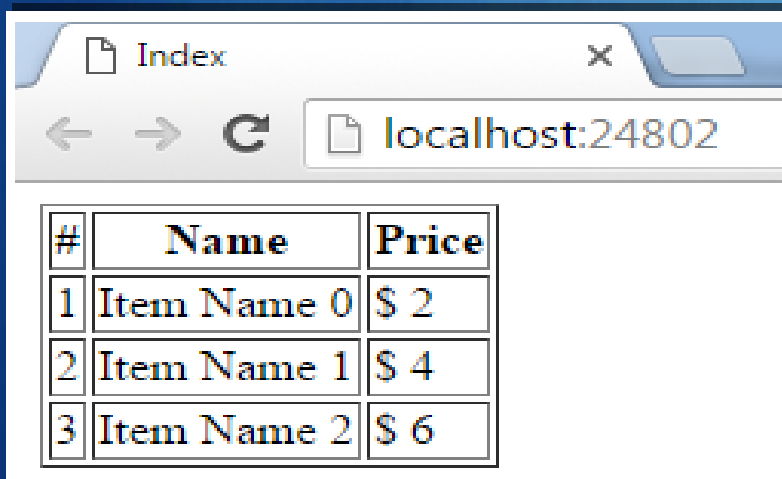
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
        @*В отличии от частичных представлений дочерние
           действия позволяют выполнять дополнительные
           действия в контроллере*@
        @Html.Action("ShowTable", new { numberOfRows = 3 })
    </div>
</body>
</html>
```



Example

Дочерние действия



#	Name	Price
1	Item Name 0	\$ 2
2	Item Name 1	\$ 4
3	Item Name 2	\$ 6

```
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Index</title>
</head>
<body>
  <div>

    <table border="1">
      <tr>
        <th> #      </th>
        <th> Name   </th>
        <th> Price  </th>
      </tr>

      <tr>
        <td> 1      </td>
        <td> Item Name 0    </td>
        <td> $ 2    </td>
      </tr>
      <tr>
        <td> 2      </td>
        <td> Item Name 1    </td>
        <td> $ 4    </td>
      </tr>
      <tr>
        <td> 3      </td>
        <td> Item Name 2    </td>
        <td> $ 6    </td>
      </tr>
    </table>
```



Layout-представления

Index.cshtml HomeController.cs _Layout.cshtml BundleCo

```
@{  
    Layout = null;  
}  
  
<!DOCTYPE html>  
  
<html>  
<head>  
    <meta name="viewport" content="width=device-width" />  
    <title>Index</title>  
</head>  
<body>  
    <div>  
  
    </div>  
</body>  
</html>
```

Layout-представления

The diagram illustrates the layout inheritance in ASP.NET MVC. It shows two code files: `_Layout.cshtml` and `Index.cshtml`.

_Layout.cshtml (Left):

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width" />
  <title>@ViewBag.Title</title>
  @Styles.Render("~/Content/css")
  @Scripts.Render("~/bundles/modernizr")
</head>
<body>
  @RenderBody()
  @Scripts.Render("~/bundles/jquery")
  @RenderSection("scripts", required: false)
</body>
</html>
```

Index.cshtml (Right):

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

<h2>Index</h2>
```

Diagram Annotations:

- An orange arrow points from the `<html>` tag in `_Layout.cshtml` to the `@{` block in `Index.cshtml`.
- A blue arrow points from the `@ViewBag.Title` in `_Layout.cshtml` to the `ViewBag.Title = "Index";` line in `Index.cshtml`.
- A blue arrow points from the `@RenderBody()` tag in `_Layout.cshtml` to the `<h2>Index</h2>` line in `Index.cshtml`.

Layout-представления

Add View

View name:
Index1

View engine:
Razor (CSHTML)

☐ Create a strongly-typed view

Model class:

Scaffold template:
Empty

☒ Reference script libraries

☐ Create as a partial view

☒ Use a layout or master page:

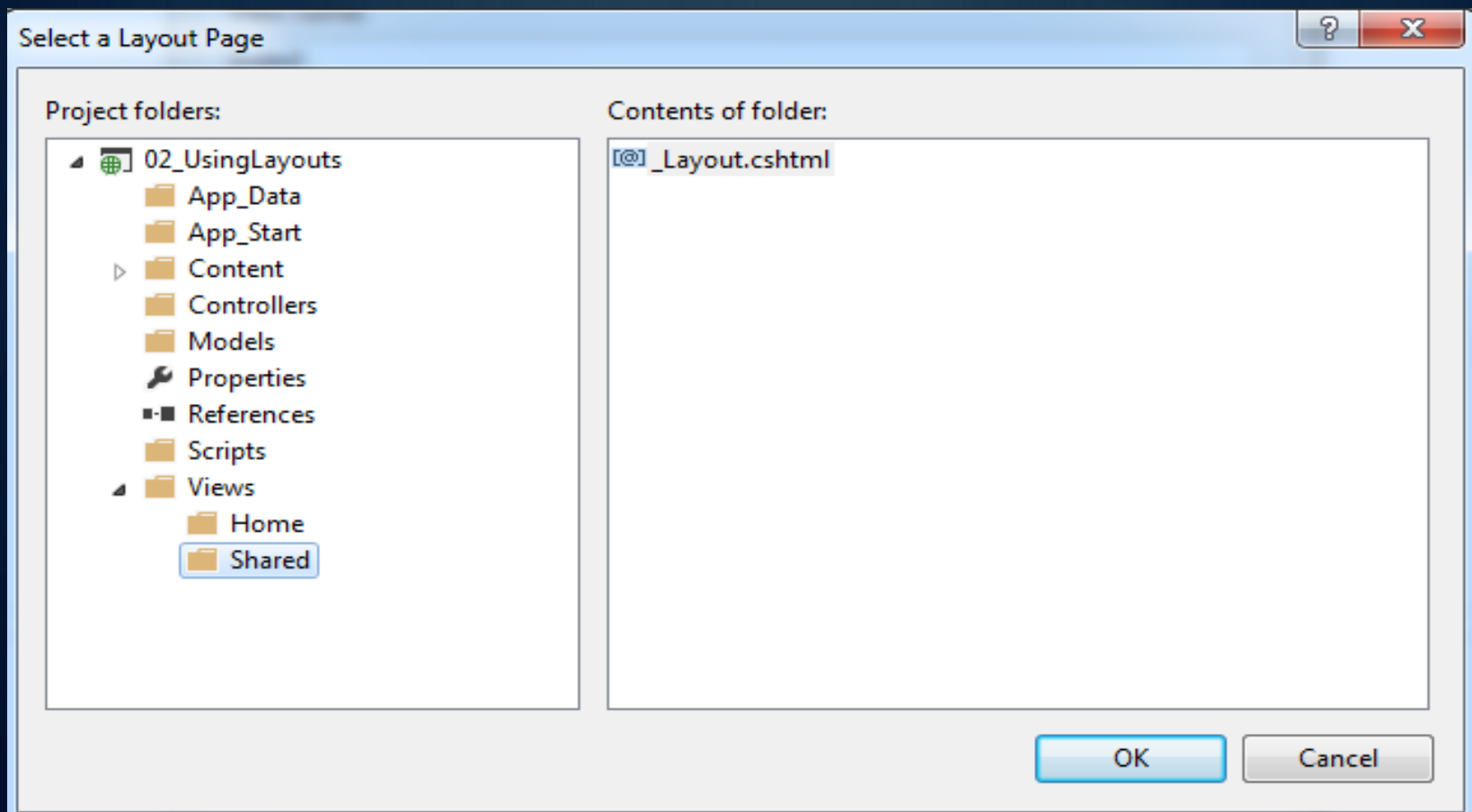
...

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

ContentPlaceHolder ID:
MainContent

Add Cancel

Layout-представления



Layout-представления

Diagram illustrating the structure of a layout view and its associated components in an ASP.NET MVC application.

_Layout.cshtml (Main Layout View):

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width" />
  <title>@ViewBag.Title</title>
  @Styles.Render("~/Content/css")
  @Scripts.Render("~/bundles/modernizr")
</head>
<body>
  @RenderBody()
  @Scripts.Render("~/bundles/jquery")
  @RenderSection("scripts", required: false)
</body>
</html>
```

Index.cshtml* (View):

```
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
```

_ViewStart.cshtml* (View):

```
@*Код в этом файле выполнится перед вызовом
любого представления
*@

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

Annotations and Arrows:

- A blue arrow points from `@ViewBag.Title` in the `<title>` tag of `_Layout.cshtml` to `ViewBag.Title = "Index";` in `Index.cshtml*`.
- A blue arrow points from `@RenderBody()` in the `<body>` tag of `_Layout.cshtml` to `<h2>Index</h2>` in `Index.cshtml*`.

Layout-представления

_Layout.cshtml* X

```
<!DOCTYPE html>
<html>
<head>
  <title>@ViewBag.Title</title>
  <link href="@Url.Content("~/Content/Site.cs")" rel="stylesheet" type="text/css">
  <script src="@Url.Content("~/Scripts/jquery.js")" type="text/javascript">
</head>
<body>
  @*
  При создании компоновки, вместо метода
  можно использовать только методы Render
  поможет создать более строгую структуру
  представления состоящую только из секций
  *@
  @RenderSection("Header")
  @RenderSection("Body")
  @RenderSection("Footer")
</body>
</html>
```

Index.cshtml* X

```
@{
  ViewBag.Title = "Index RenderSections";
}

@section Header{
  <h1>
    <b>Header</b> представления Index
  </h1>
}

@section Body{
  <p>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit
    fusce vel sapien elit in malesuada semper mi, id
    sollicitudin urna fermentum ut fusce varius nisl
    ac ipsum gravida vel pretium tellus.
  </p>
}

@section Footer{
  <h3>
    <b>Footer</b> представления Index
  </h3>
}
```

Layout-представления

Если в Layout странице, присутствует вызов метода `RenderSection("имя секции")`, а в представлении, который связывается с данным шаблоном данная секция отсутствует, то при выполнении программы произойдет ошибка, во избежание ошибок во время выполнения можно воспользоваться следующими способами:

1. Воспользоваться перегруженной функцией

```
@*Визуализация необязательной секции Footer*@  
@RenderSection(name: "Footer", required: false)
```

которая визуализирует данную секцию из текущего представления, если оно присутствует, если же нет, то ничего не выполнится.

Layout-представления

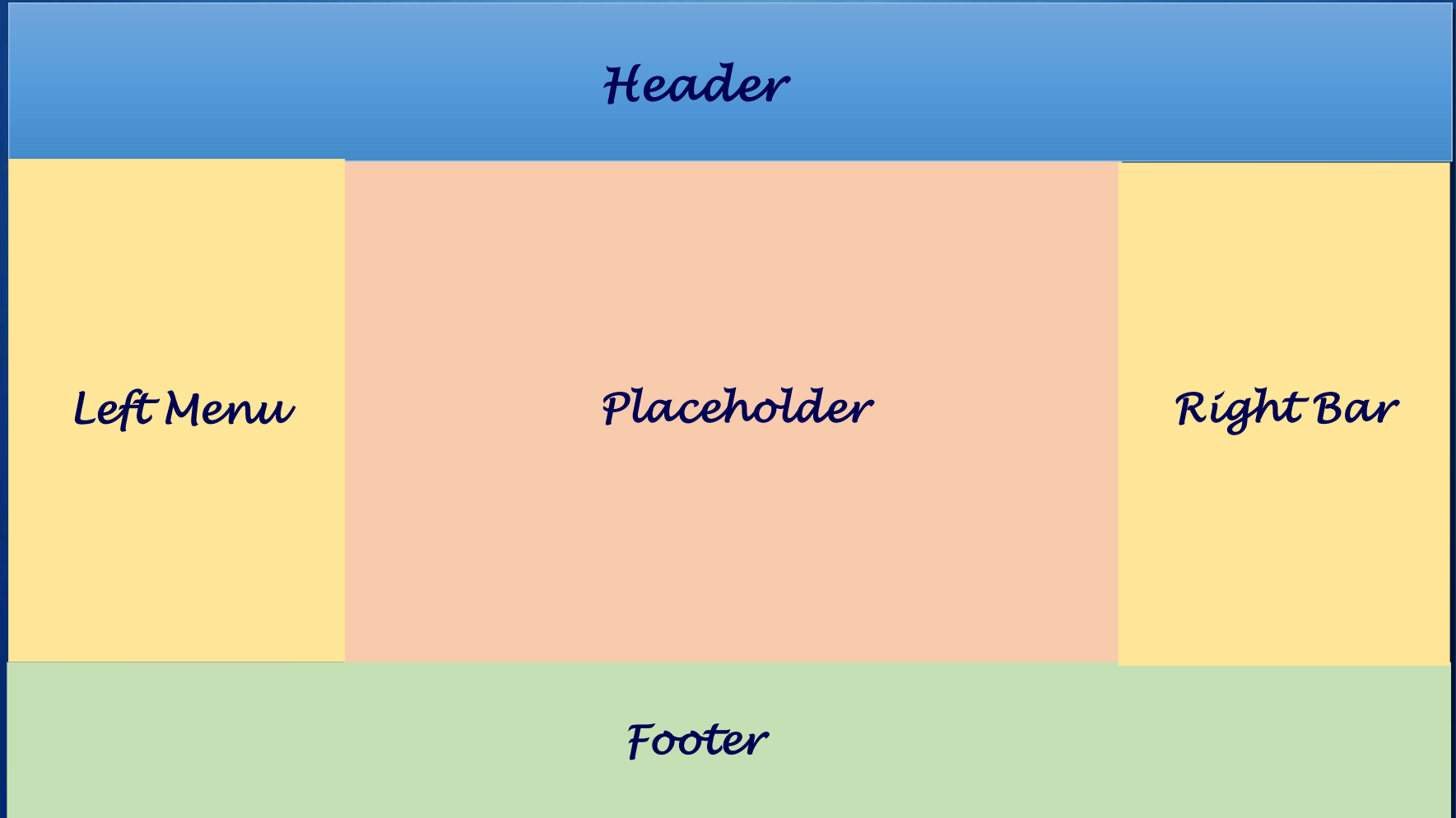
2. Воспользоваться функцией `IsSectionDefined("Footer")` до вызова метода `RenderSection("Footer")`.

```
@*Проверка наличия секции Footer в текущем представлении*@  
@if (IsSectionDefined("Footer"))  
{  
    @RenderSection("Footer");  
}  
else  
{  
    <h4>Default footer</h4>  
}
```

Layout-представления



Layout-представления

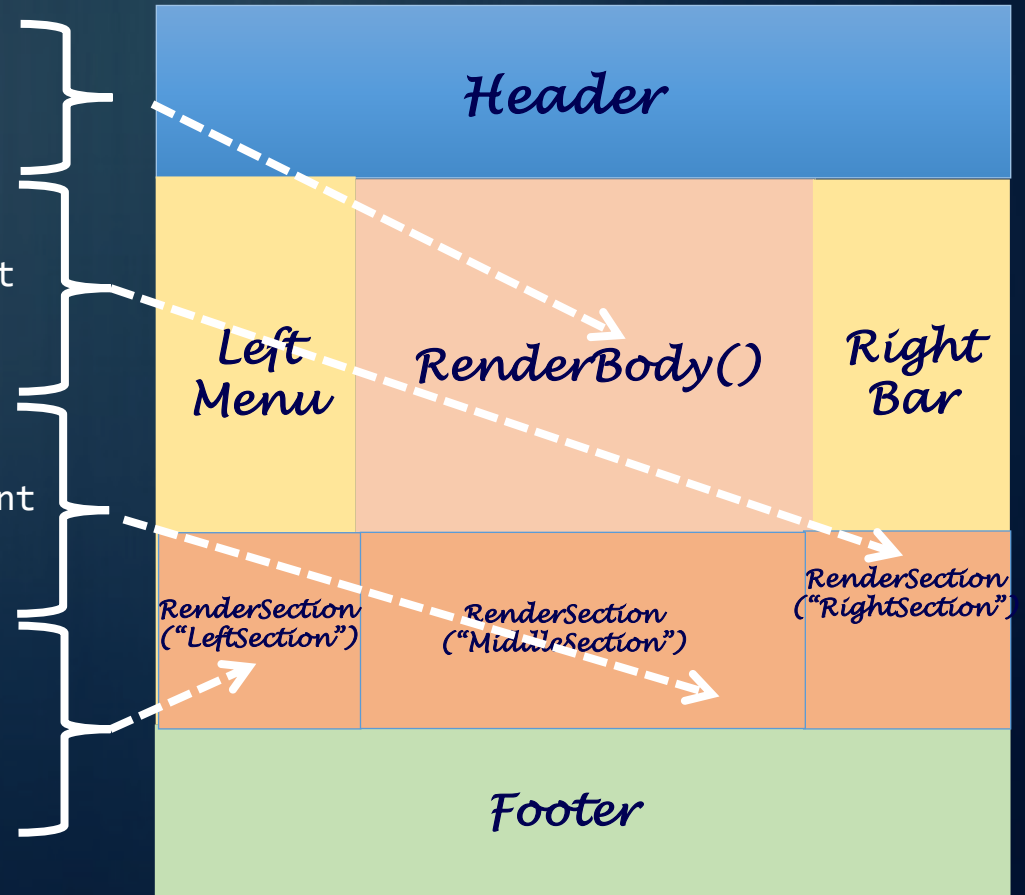


Layout-представления

Index.cshtml

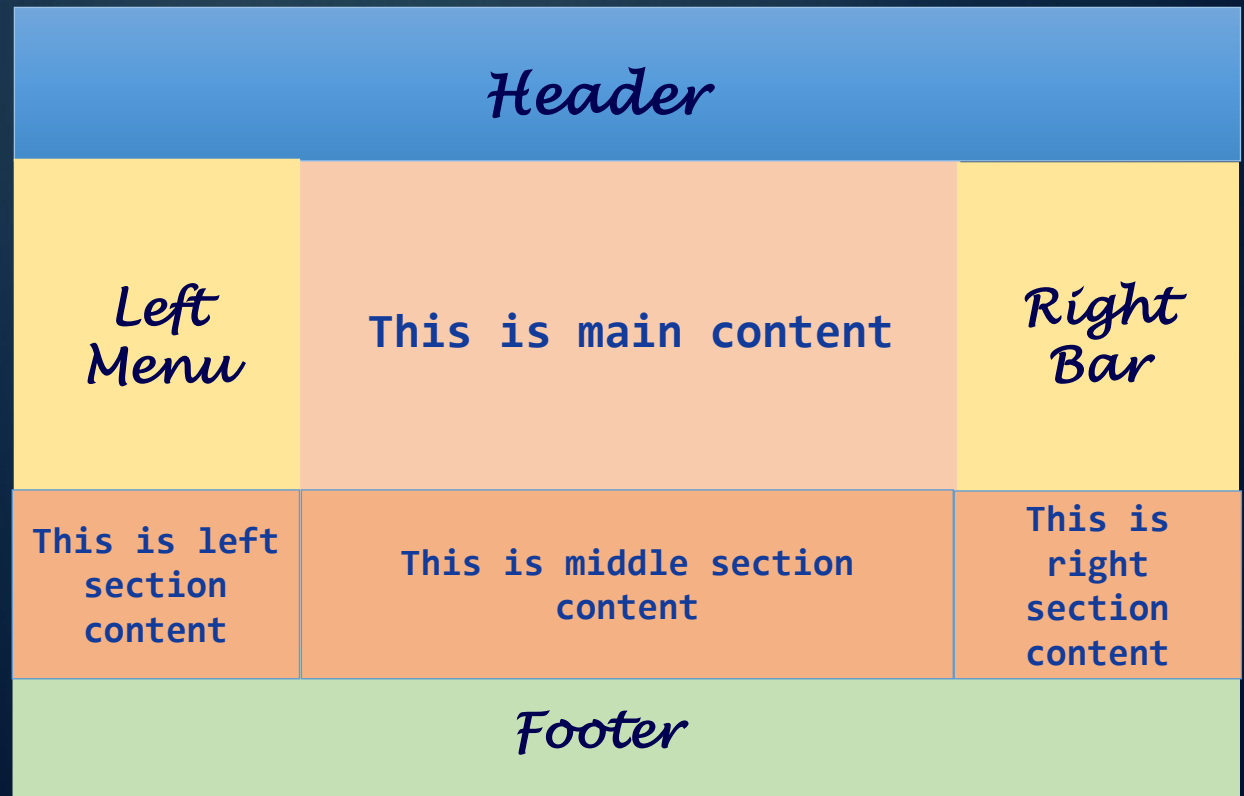
```
<div>
    This is main content
</div>
@section RightSection{
<text>
    This is right section content
</text>
}
@section MiddleSection{
<text>
    This is middle section content
</text>
}
@section LeftSection{
<text>
    This is left section content
</text>
}
```

_Layout.cshtml



Layout-представления

<http://localhost/Home/Index>



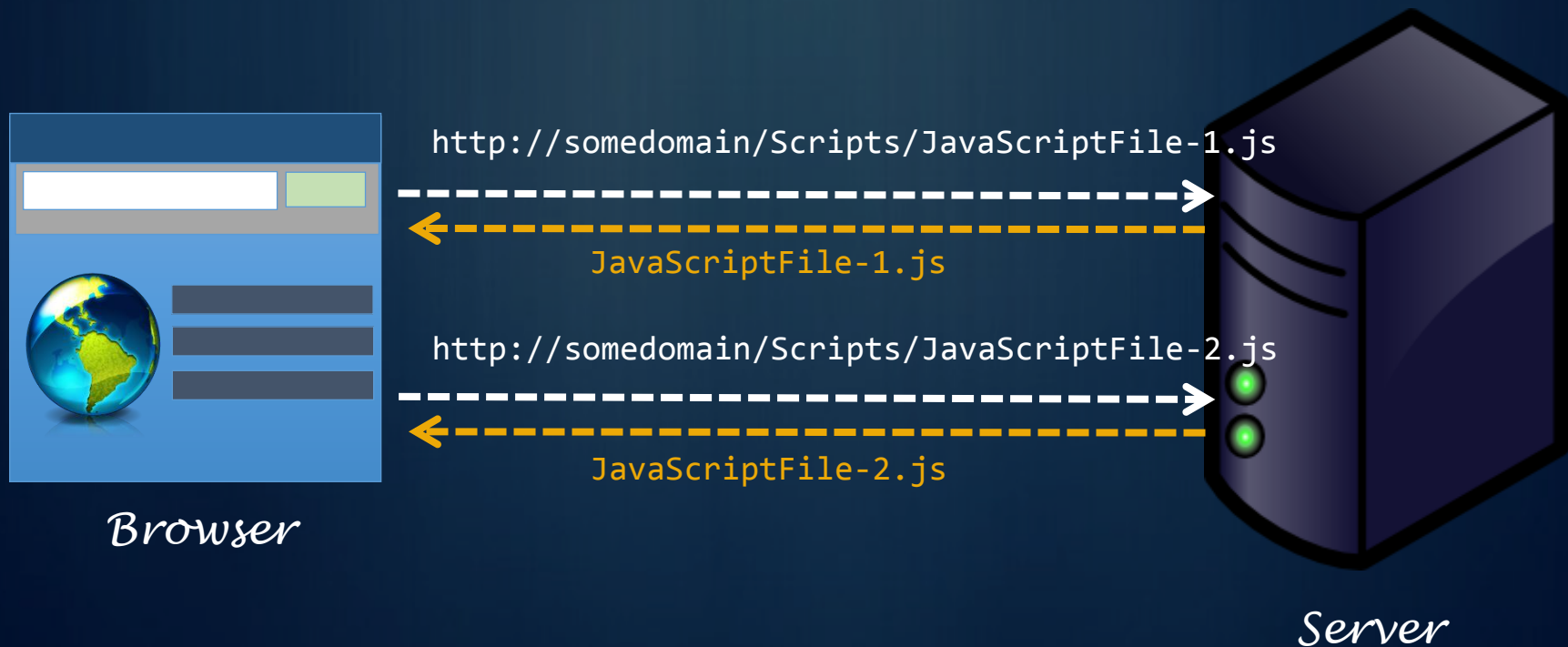
Layout-представления

_Layout.cshtml

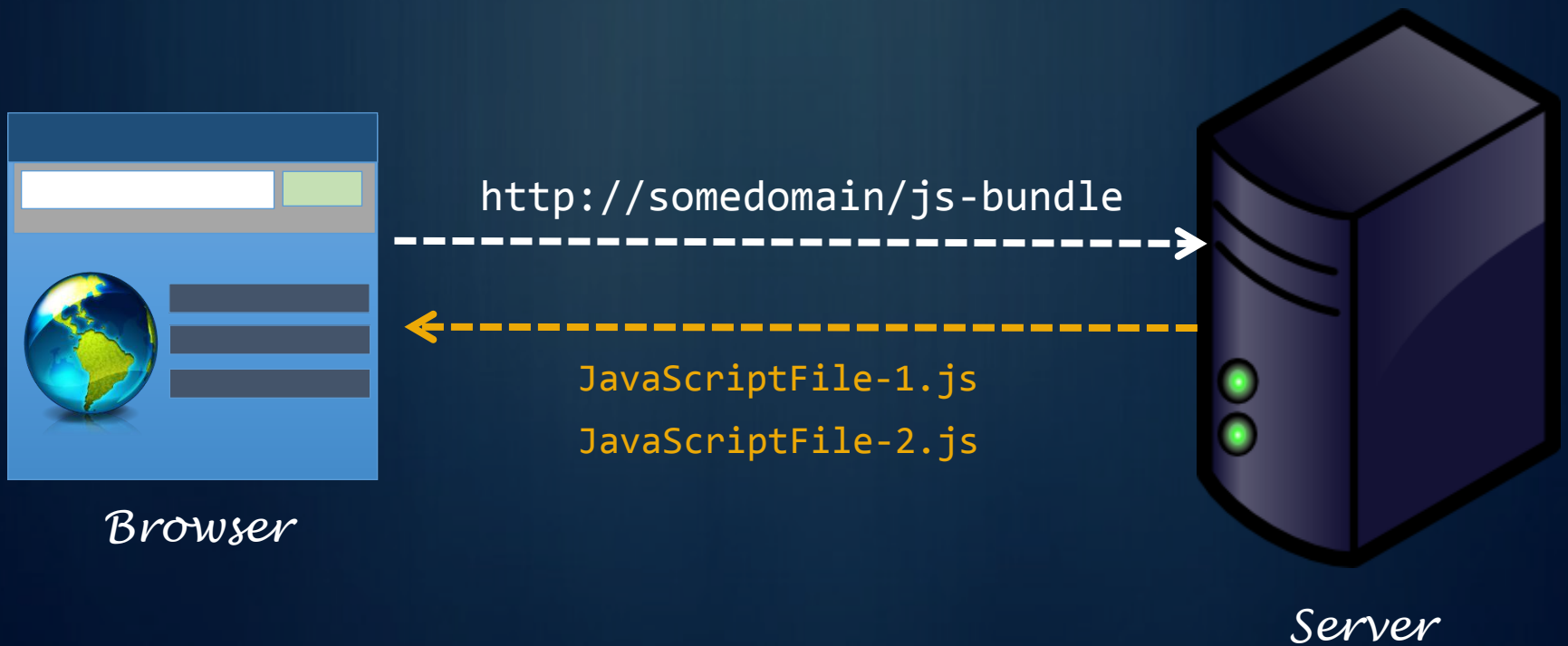
```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width" />
  <title>@ViewBag.Title</title>
  @Styles.Render("~/Content/css")
  @Scripts.Render("~/bundles/modernizr")
</head>
<body>
  @RenderBody()

  @Scripts.Render("~/bundles/jquery")
  @RenderSection("scripts", required: false)
</body>
</html>
```

Бандлы и минификация



Бандлы и минификация



Example

Бандлы и минификация

```
public class BundleConfig
{
    // For more information on Bundling, visit http://go.microsoft.com/fwlink/?LinkId=254725
    1 reference
    public static void RegisterBundles(BundleCollection bundles)
    {
        // Создание бандла ~/bundles/jquery в который войдут файлы jquery-*
        bundles.Add(new ScriptBundle("~/bundles/jquery")
            .Include("~/Scripts/jquery-{version}.js")
            .Include("~/Scripts/jquery.validate.js"));

        // Создание бандла ~/bundles/modernizr для библиотеки modernizr
        bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(
            "~/Scripts/modernizr-*"));

        // Создание бандла для стилей
        bundles.Add(new StyleBundle("~/Content/css").Include("~/Content/site.css"));
    }
}
```

- ▲ App_Start
 - ▶ **C# BundleConfig.cs**
 - ▶ C# FilterConfig.cs
 - ▶ C# RouteConfig.cs
 - ▶ C# WebApiConfig.cs



Example

Бандлы и минификация

```
namespace _08_Bundling
```

```
{
```

```
1 reference
```

```
public class BundleConfig
```

```
{
```

```
// For more information on Bundling, visit http://go.microsoft.com/fwlink/?LinkId
```

```
1 reference
```

```
public static void RegisterBundles(BundleCollection bundles)
```

```
{
```

```
// Создание бандла ~/bundles/jquery в который войдут файлы jquery-*
```

```
bundles.Add(new ScriptBundle("~/bundles/jquery")
```

```
.Include("~/Scripts/jquery-{version}.js")
```

```
.Include("~/Scripts/jquery.validate.js"));
```

```
// Создание бандла ~/bundles/modernizr для библиотеки modernizr
```

```
bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(
```

```
"~/Scripts/modernizr-*"));
```

```
// Создание бандла для стилей
```

```
bundles.Add(new StyleBundle("~/Content/css").Include("~/Content/site.css"));
```

```
}
```

```
}
```

```
}
```

Scripts

_references.js

jquery-1.8.2.intellisense.js

jquery-1.8.2.js

jquery-1.8.2.min.js

jquery-ui-1.8.24.js

jquery-ui-1.8.24.min.js

jquery.unobtrusive-ajax.js

jquery.unobtrusive-ajax.min.js

jquery.validate-vsdoc.js

jquery.validate.js

jquery.validate.min.js

jquery.validate.unobtrusive.js

jquery.validate.unobtrusive.min.js

knockout-2.2.0.debug.js

knockout-2.2.0.js

modernizr-2.6.2.js

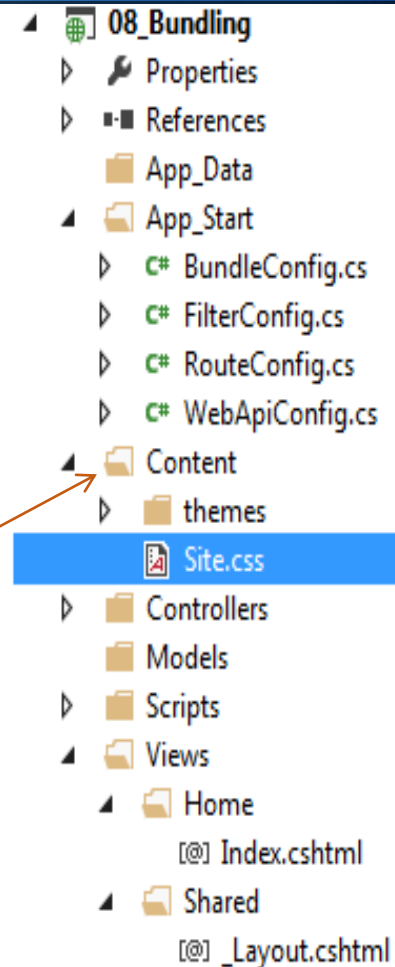


Example

Бандлы и минификация

namespace _08_Bundling

```
{  
    1 reference  
    public class BundleConfig  
    {  
        // For more information on Bundling, visit http://go.microsoft.com/fwlink/?LinkId=  
        1 reference  
        public static void RegisterBundles(BundleCollection bundles)  
        {  
            // Создание бандла ~/bundles/jquery в который войдут файлы jquery-*  
            bundles.Add(new ScriptBundle("~/bundles/jquery")  
                .Include("~/Scripts/jquery-{version}.js")  
                .Include("~/Scripts/jquery.validate.js"));  
  
            // Создание бандла ~/bundles/modernizr для библиотеки modernizr  
            bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(  
                "~/Scripts/modernizr-*"));  
  
            // Создание бандла для стилей  
            bundles.Add(new StyleBundle("~/Content/css").Include("~/Content/site.css"));  
        }  
    }  
}
```



Example

Бандлы и минификация

```
<meta name="viewport" content="width=device-width" />
<title>@ViewBag.Title</title>
```

@*

Бандл - набор CSS или JavaScript файлов, которые на этапе выполнения будут минимизированы и объединены в один файл. Минификация и бандлинг будут работать только если запустить веб приложение с настройкой <compilation debug="false" targetFramework="4.5" /> в файле web.config

Настройка бандлов происходит в файле App_Start/BundleConfig.cs

*@

@*

Бандл, который содержит все стили.

На этапе выполнения преобразовывается в <link href="...." />

*@

```
@Styles.Render("~/Content/css")
```

@*

Бандл ссылается на JavaScript библиотеку modernizr для определения поддержки HTML5 и CSS3 возможностей в браузере

На этапе выполнения преобразовывается в <script src="...."></script>

*@

```
@Scripts.Render("~/bundles/modernizr")
```

```
</head>
```

```
<body>
```

```
@RenderBody()
```

@*Бандл с JavaScript библиотекой jQuery*@

```
@Scripts.Render("~/bundles/jquery")
```

@*Не обязательная секция для сценариев отдельных представлений*@

```
@RenderSection("scripts", required: false)
```

```
</body>
```

Example

2015 © EPAM Systems

Бандлы и минификация

JavaScript

```
sayHello = function(name){  
  var msg = "Hello, " + name;  
  alert(msg);  
}
```

*JavaScript function
will be optimized into*

Minified JavaScript

```
sayHello = function(n){var t="Hello, "+n;alert(t);}
```



Example

Бандлы и минификация

Бандл - набор CSS или JavaScript файлов, которые на этапе выполнения будут минимизированы и объединены в один файл.

Минификация и бандлинг будут работать только если запустить веб приложение с настройкой

Web.config* [icon] X

```
<system.web>
  <httpRuntime targetFramework="4.5" />
  <compilation debug="false" targetFramework="4.5" />
  <authentication mode="Forms">
    <forms loginUrl="~/Account/Login" timeout="2880" />
  </authentication>
```



Example

Бандлы и минификация

```
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width" />
  <title>Index</title>

  <link href="/Content/css?v=ji3n01pdg6VLv3CVUWntxgZNf1zRciWDbm4YfW-y0RI1" rel="stylesheet"/>

  <script src="/bundles/modernizr?v=qVODBytEBVVePTNtSFXgRX0NCEjh9U_Oj8ePaSiRcGg1"></script>
</head>
<body>

<h2>Index (Bundling & Minification)</h2>

<p>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit fusce vel sapien elit in malesuada semper mi,
  pretium tellus.
  Tincidunt integer eu augue augue nunc elit dolor, luctus placerat scelerisque euismod, iaculis eu lacu
  metus vel.
  Placerat suscipit, orci nisl iaculis eros, a tincidunt nisi odio eget lorem nulla condimentum tempor m
  ante.
</p>

  <script src="/bundles/jquery?v=cdZYbm6OGaxcGF2pLxluQRey dqJGM6riAnxMGX1fyLk1"></script>
</body>
</html>
```



Бандлы и минификация

← → ↻  localhost:24003/Content/css?v=j3nO1pdg6VLv3CVUWntxgZNf1zRciWDbm4YfW-y0RI1

```
body{font-size:.85em;font-family:"Segoe UI",Verdana,Helvetica,Sans-Serif;color:#232323;background-color:#fff}header,
#ddd;padding:0 1.4em 1.4em 1.4em;margin:0 0 1.5em 0}legend{font-size:1.2em;font-weight:bold}textarea{min-height:75px
validation-error{color:red}.field-validation-valid{display:none}.input-validation-error{border:1px solid red;background
weight:bold;color:red}.validation-summary-valid{display:none}
```



Example

Введение в AJAX

Что такое AJAX

AJAX (аббревиатура от «**Asynchronous Javascript And Xml**») — технология обращения гибкого взаимодействия между клиентом и сервером без перезагрузки страницы.

За счет этого уменьшается время отклика и веб-приложение по интерактивности больше напоминает десктоп.

Несмотря на то, что в названии технологии присутствует буква X (от слова XML), использовать XML вовсе не обязательно. Под AJAX подразумевают любое общение с сервером без перезагрузки страницы, организованное при помощи JavaScript

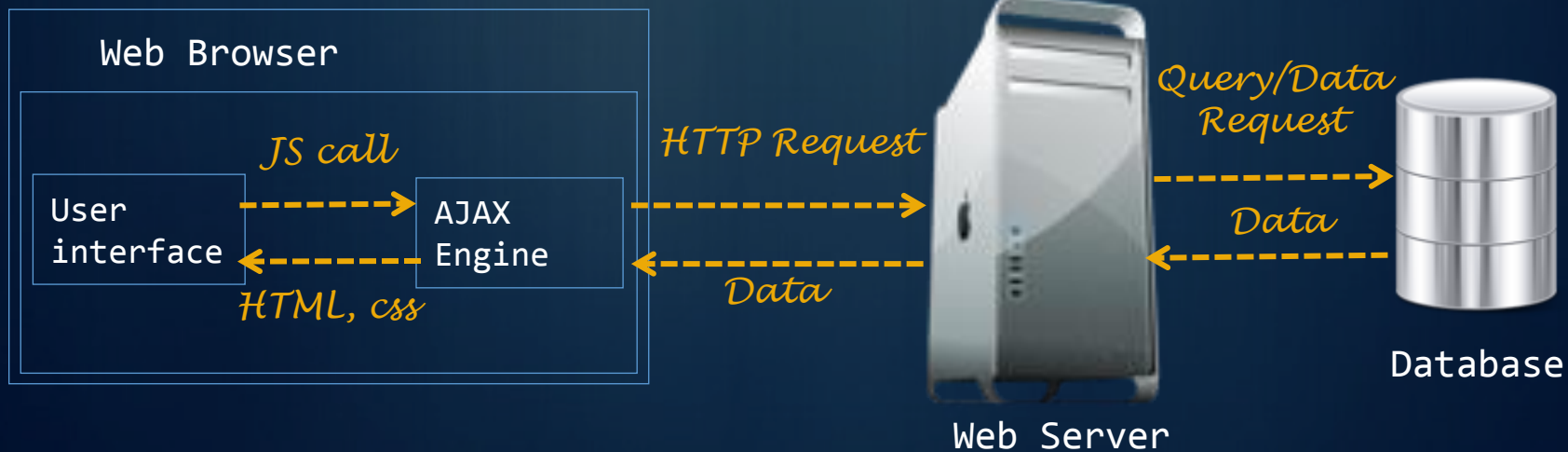
Как работает AJAX

Традиционное веб-приложение



Как работает AJAX

Веб-приложение с применением AJAX



Какие технологии включает AJAX

AJAX - это набор технологий, которые поддерживаются веб-браузерами. AJAX использует:

- **HTML** в качестве "каркаса"
- **CSS** для оформления
- **DOM** для извлечения или изменения информации на странице
- Объект **XMLHttpRequest** для асинхронного обмена данными с сервером
- **JavaScript** для связи перечисленных выше технологий между собой

Основы XMLHttpRequest

Объект **XMLHttpRequest** (кратко его называют, «XHR») дает возможность из JavaScript делать HTTP-запросы к серверу без перезагрузки страницы. Несмотря на слово «XML» в названии, XMLHttpRequest может работать с любыми данными, а не только с XML.

<http://www.w3.org/TR/XMLHttpRequest/>



Example

Основы XMLHttpRequest

```
class OrdersDatabase
{
    1 reference
    public static IEnumerable<Order> Orders
    {
        get
        {
            Thread.Sleep(2000);

            List<Order> list = new List<Order>();
            list.Add(new Order() { Id = 1, Product = "Product 1", Customer = "Ivanov", Quantity = 1 });
            list.Add(new Order() { Id = 2, Product = "Product 2", Customer = "Petrov", Quantity = 10 });
            list.Add(new Order() { Id = 3, Product = "Product 2", Customer = "Fedorov", Quantity = 12 });
            list.Add(new Order() { Id = 4, Product = "Product 3", Customer = "Fedorov", Quantity = 6 });
            list.Add(new Order() { Id = 5, Product = "Product 1", Customer = "Petrov", Quantity = 7 });
            list.Add(new Order() { Id = 6, Product = "Product 4", Customer = "Ivanov", Quantity = 11 });
            list.Add(new Order() { Id = 7, Product = "Product 2", Customer = "Petrov", Quantity = 10 });
            list.Add(new Order() { Id = 8, Product = "Product 5", Customer = "Petrov", Quantity = 2 });
            list.Add(new Order() { Id = 9, Product = "Product 1", Customer = "Ivanov", Quantity = 11 });
            list.Add(new Order() { Id = 10, Product = "Product 5", Customer = "Fedorov", Quantity = 3 });
            list.Add(new Order() { Id = 11, Product = "Product 2", Customer = "Petrov", Quantity = 3 });
            list.Add(new Order() { Id = 12, Product = "Product 1", Customer = "Ivanov", Quantity = 3 });
            list.Add(new Order() { Id = 13, Product = "Product 4", Customer = "Petrov", Quantity = 7 });
            return list;
        }
    }
}
```

```
public class Order
{
    13 references
    public int Id { get; set; }
    13 references
    public string Product { get; set; }
    14 references
    public string Customer { get; set; }
    13 references
    public int Quantity { get; set; }
}
```



Основы XMLHttpRequest

Index.cshtml*

```
@{
    ViewBag.Title = "Orders";
}

<h2>Orders</h2>

@using (Html.BeginForm())
{
    <table cellpadding="4">
        <thead>
            <tr>
                <th>Id</th>
                <th>Product</th>
                <th>Quantity</th>
                <th>Customer</th>
            </tr>
        </thead>

        <tbody id="tabledata">
            @Html.Action("OrdersData", new { id = Model })
        </tbody>

    </table>
    <p>
        @*При отправке обратного запроса на сервер, в форме будет содержаться поле id которое
        методом действия будет использоваться для фильтрации данных*@
        @Html.DropDownList("id", new SelectList(new[] { "All", "Ivanov", "Petrov", "Fedorov" },
            (Model ?? "All")))

        <input type="submit" value="Submit" />
    </p>
}
```



Example

001_SampleApplication

2015 © EPAM Systems

52

Основы XMLHttpRequest

```
HomeController.cs* X
SampleApplication.Controllers.HomeController

0 references
public class HomeController : Controller
{
    0 references
    public ActionResult Index()...

    [HttpPost]
    0 references
    public ActionResult Index(string id)...

    0 references
    public ActionResult OrdersData(string id)
    {
        var data = OrdersDatabase.Orders;
        if (!string.IsNullOrEmpty(id) && id != "All")
        {
            // выполняем выборку по свойству Customer если
            // значение id не пустое и не равно "All"
            data = data.Where(e => e.Customer == id);
        }
        return PartialView(data);
    }
}
```

```
OrdersData.cshtml* X
@model IEnumerable<SampleApplication.Models.Order>

@foreach (SampleApplication.Models.Order order in Model)
{
    <tr>
        <td>@Html.DisplayFor(m => order.Id)</td>
        <td>@Html.DisplayFor(m => order.Product)</td>
        <td>@Html.DisplayFor(m => order.Quantity)</td>
        <td>@Html.DisplayFor(m => order.Customer)</td>
    </tr>
}
```



Основы XMLHttpRequest

```
Index.cshtml*  X
@{
    ViewBag.Title = "Orders";
}

<h2>Orders</h2>

@using (Html.BeginForm())
{
    <table cellpadding="4">
        <thead>
            <tr>
                <th>Id</th>
                <th>Product</th>
                <th>Quantity</th>
                <th>Customer</th>
            </tr>
        </thead>

        <tbody id="tabledata">
            @Html.Action("OrdersData", new { id = Model })
        </tbody>

    </table>

    <p>
        @*При отправке обратного запроса на сервер, в форме будет содержаться поле id которое
        методом действия будет использоваться для фильтрации данных*@
        @Html.DropDownList("id", new SelectList(new[] { "All", "Ivanov", "Petrov", "Fedorov" },
            (Model ?? "All")))

        <input type="submit" value="Submit" />
    </p>
}
```



Example

001_SampleApplication

2015 © EPAM Systems

54

Основы XMLHttpRequest

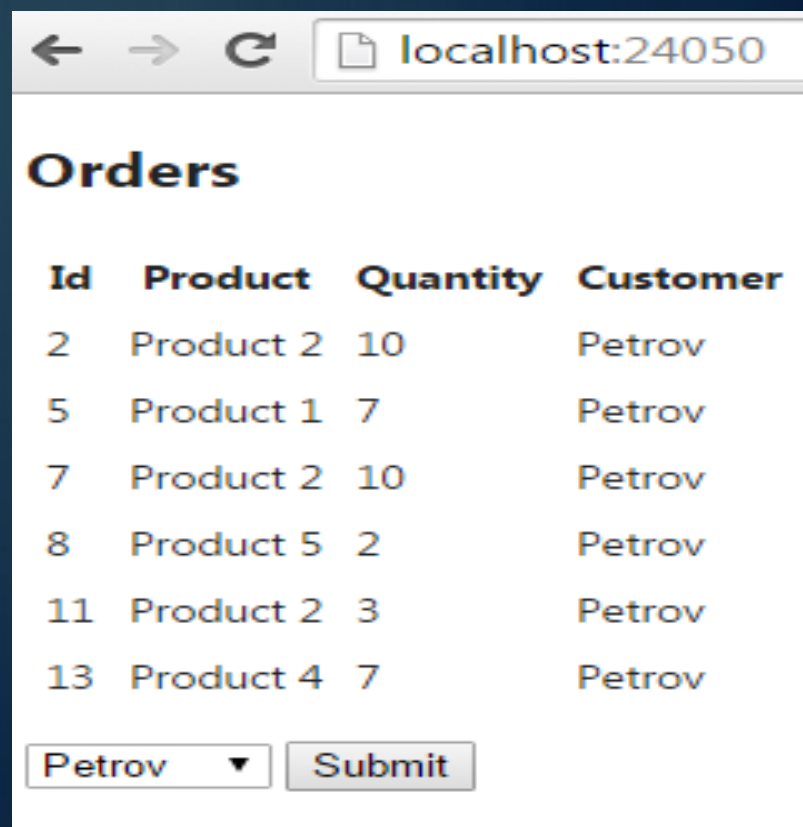


← → ↻ localhost:24050

Orders

Id	Product	Quantity	Customer
1	Product 1	1	Ivanov
2	Product 2	10	Petrov
3	Product 2	12	Fedorov
4	Product 3	6	Fedorov
5	Product 1	7	Petrov
6	Product 4	11	Ivanov
7	Product 2	10	Petrov
8	Product 5	2	Petrov
9	Product 1	11	Ivanov
10	Product 5	3	Fedorov
11	Product 2	3	Petrov
12	Product 1	3	Ivanov
13	Product 4	7	Petrov

All ▼ Submit



← → ↻ localhost:24050

Orders

Id	Product	Quantity	Customer
2	Product 2	10	Petrov
5	Product 1	7	Petrov
7	Product 2	10	Petrov
8	Product 5	2	Petrov
11	Product 2	3	Petrov
13	Product 4	7	Petrov

Petrov ▼ Submit



Example

001_SampleApplication

2015 © EPAM Systems

55






Основы XMLHttpRequest

Id	Product	Quantity	Customer
3	Product 2	12	Fedorov
4	Product 3	6	Fedorov
10	Product 5	3	Fedorov

Fedorov ▼

Submit

Name
Path

 localhost
 site.css /Content
 modernizr-2.6.2.js /Scripts
 jquery-1.8.2.js /Scripts
 browserLink /4079c54eb5944cc2bb330e...

× Headers Preview Response Timing

▼ General

Request URL: http://localhost:24050/
Request Method: POST
Status Code: ● 200 OK
Remote Address: [::1]:24050

► Response Headers (11)

► Request Headers (13)

▼ Form Data

view source

view URL encoded

id: Fedorov



Example

001_SampleApplication

2015 © EPAM Systems

56

Основы XMLHttpRequest

The screenshot displays the 'Response' tab of a web browser's developer tools. The left sidebar shows the file tree for 'localhost', including 'site.css', 'modernizr-2.6.2.js', 'jquery-1.8.2.js', and 'browserLink'. The main pane shows the HTML response of an XMLHttpRequest, with line numbers 1 through 23. The HTML code is as follows:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8" />
5   <meta name="viewport" content="width=device-width" />
6   <title>Orders</title>
7   <link href="/Content/site.css" rel="stylesheet"/>
8
9   <script src="/Scripts/modernizr-2.6.2.js"></script>
10 </head>
11 <body>
12
13   <h2>Orders</h2>
14
15   <form action="/" method="post">   <table cellpadding="4">
16     <thead>
17       <tr>
18         <th>Id</th>
19         <th>Product</th>
20         <th>Quantity</th>
21         <th>Customer</th>
22       </tr>
23     </thead>
```



Example

001_SampleApplication

2015 © EPAM Systems

57

Основы XMLHttpRequest

Index.cshtml* [icon] X

```
@{  
    ViewBag.Title = "Orders";  
  
    // UpdateTargetId - элемент на странице, который будет  
    //обновляется после получения новых данных со стороны сервера.  
    // Url - адрес ресурса на сервере, который возвращает  
    //данные для обновления страницы  
    var options = new AjaxOptions()  
    {  
        UpdateTargetId = "tabledata",  
        Url = Url.Action("OrdersData")  
    };  
}
```



Example

002_SimpleAjaxApplication

2015 © EPAM Systems

58

Основы XMLHttpRequest

```
Index.cshtml* X
@using (Ajax.BeginForm(options))
{
    <table cellpadding="4">
    <thead>
        <tr>
            <th>Id</th>
            <th>Product</th>
            <th>Quantity</th>
            <th>Customer</th>
        </tr>
    </thead>

    <tbody id="tabledata">
        @Html.Action("OrdersData", new { id = Model })
    </tbody>

    </table>
    <p>
        @*При отправке обратного запроса на сервер, в форме будет содержаться поле id которое
        методом действия будет использоваться для фильтрации данных*@
        @Html.DropDownList("id", new SelectList(new[] { "All", "Ivanov", "Petrov", "Fedorov" },
            (Model ?? "All")))
        <input type="submit" value="Submit" />
    </p>
}
```

```
var options = new AjaxOptions()
{
    UpdateTargetId = "tabledata",
    Url = Url.Action("OrdersData")
};
```

Основы XMLHttpRequest

Name Path: OrdersData /Home

Headers Preview Response Timing

General

Request URL: http://localhost:24701/Home/OrdersData
Request Method: POST
Status Code: 200 OK
Remote Address: [::1]:24701

Response Headers (11)

Request Headers (13)

Form Data view source view URL encoded

id: Fedorov
X-Requested-With: XMLHttpRequest

Name Path: OrdersData /Home

Headers Preview Response Timing

```
1 <tr>
2   <td>3</td>
3   <td>Product 2</td>
4   <td>12</td>
5   <td>Fedorov</td>
6 </tr>
7 <tr>
8   <td>4</td>
9   <td>Product 3</td>
10  <td>6</td>
11  <td>Fedorov</td>
12 </tr>
13 <tr>
14   <td>10</td>
15   <td>Product 5</td>
16   <td>3</td>
17   <td>Fedorov</td>
18 </tr>
```



Example

Основы XMLHttpRequest

```
<form action="/" data-ajax="true" data-ajax-mode="replace"  
      data-ajax-update="#tabledata"  
      data-ajax-url="/Home/OrdersData" id="form0" method="post">
```

При этом у формы появились несколько новых атрибутов – это

1. **data-ajax-url="/Home/OrdersData"** это при нажатии на submit, запросы должны ссылаться на дочернее действие OrdersData из контроллера Home указанном именно в этом атрибуте.
2. **data-ajax-update="#tabledata"** когда от сервера придет ответ при данном запросе, то используя id- шник **tabledata** и найдет на странице tabledata и именно в этот tabledata поместит результат полученный со стороны сервера
3. **data-ajax-mode="replace"** говорит о том что необходимо заранее удалить весь контент который находился на этом месте и разместить новый полученный.



Основы XMLHttpRequest

```
Index.cshtml* X
@{
    ViewBag.Title = "Orders";

    // LoadingElementId - элемент, который должен отображаться
    // в процессе отправки и получения данных с сервера.
    // Confirm - при наличии значения у данного свойства, перед
    // отправкой запроса на сервер будет отображаться диалоговое окно с запросом.
    var options = new AjaxOptions()
    {
        UpdateTargetId = "tabledata",
        Url = Url.Action("OrdersData"),
        LoadingElementId = "loadingIndicator",
        Confirm = "Вы хотите запросить новые данные?"
    };
}

<h2>Orders</h2>

@using (Ajax.BeginForm(options))
{
    <table>...</table>
    <p>
        @*При отправке обратного запроса на сервер, в форме будет
        одержаться поле id которое методом действия будет использоваться для фильтрации данных*@
        @Html.DropDownList("id", new SelectList(new[] { "All", "Ivanov", "Petrov", "Fedorov" },
            (Model ?? "All")))

        <input type="submit" value="Submit" /> 
    </p>
}
```



Основы XMLHttpRequest

Orders

Id	Product	Quantity	Customer
3	Product 2	12	Fedorov
4	Product 3	6	Fedorov
10	Product 5	3	Fedorov

Ivanov ▼

Submit

Подтвердите действие на localhost:25000

Вы хотите запросить новые данные?

☐ Предотвратить создание дополнительных диалоговых окон на этой странице.

OK

Отмена

Orders

Id	Product	Quantity	Customer
3	Product 2	12	Fedorov
4	Product 3	6	Fedorov
10	Product 5	3	Fedorov

Ivanov ▼

Submit



Example 3

Основы XMLHttpRequest

Index.cshtml* X

```
@{
    ViewBag.Title = "Orders";
    var options = new AjaxOptions()
    {
        UpdateTargetId = "tabledata",
        Url = Url.Action("OrdersData"),
        LoadingElementId = "loadingIndicator"
    };
}

@using (Ajax.BeginForm(options))
{
    <div id="loadingIndicator" style="display:none;">Loading...</div>
    <table>...</table>
    <p>...</p>
}

@*Ajax.ActionLink - создание ссылки, клик по которой,
будет обрабатываться кодом библиотеки jquery.unobtrusive-ajax*@

@foreach (string str in new[] { "All", "Ivanov", "Petrov", "Fedorov" })
{
    <div style="margin-right: 5px; float: left;">
        @Ajax.ActionLink(str, "OrdersData", new { id = str },
            new AjaxOptions
            {
                UpdateTargetId = "tabledata",
                LoadingElementId = "loadingIndicator"
            })
    </div>
}
}
```



Example 4

Основы XMLHttpRequest

Orders

Loading...

Id	Product	Quantity	Customer
1	Product 1	1	Ivanov
6	Product 4	11	Ivanov
9	Product 1	11	Ivanov
12	Product 1	3	Ivanov

All



Submit

[All](#) [Ivanov](#) [Petrov](#) [Fedorov](#)

Orders

Id	Product	Quantity	Customer
3	Product 2	12	Fedorov
4	Product 3	6	Fedorov
10	Product 5	3	Fedorov

All



Submit

[All](#) [Ivanov](#) [Petrov](#) [Fedorov](#)



Example 4

Основы XMLHttpRequest

```
<div style="margin-right: 5px; float: left;">
  <a data-ajax="true" data-ajax-loading="#loadingIndicator"
    data-ajax-mode="replace" data-ajax-update="#tabledata"
    href="/Home/OrdersData/All">All</a>
</div>

<div style="margin-right: 5px; float: left;">
  <a data-ajax="true" data-ajax-loading="#loadingIndicator"
    data-ajax-mode="replace" data-ajax-update="#tabledata"
    href="/Home/OrdersData/Ivanov">Ivanov</a>
</div>

<div style="margin-right: 5px; float: left;">
  <a data-ajax="true" data-ajax-loading="#loadingIndicator"
    data-ajax-mode="replace" data-ajax-update="#tabledata"
    href="/Home/OrdersData/Petrov">Petrov</a>
</div>

<div style="margin-right: 5px; float: left;">
  <a data-ajax="true" data-ajax-loading="#loadingIndicator"
    data-ajax-mode="replace" data-ajax-update="#tabledata"
    href="/Home/OrdersData/Fedorov">Fedorov</a>
</div>
```



Основы XMLHttpRequest

Index.cshtml ↗ ✕

```
@{  
    ViewBag.Title = "Orders";  
  
    // С помощью следующих свойств можно указать JavaScript функции,  
    //которые будут запускаться по мере выполнения запросов.  
    // OnBegin - начало асинхронного запроса на сервер.  
    // onSuccess - успешное завершение обработки запроса на сервере.  
    // onFailure - функция запустится в случае возникновения ошибки.  
    // onComplete - функция запускается по завершению асинхронного  
    //запроса не зависимо от результата.  
    var options = new AjaxOptions()  
    {  
        UpdateTargetId = "tabledata",  
        Url = Url.Action("OrdersData"),  
        LoadingElementId = "loadingIndicator",  
  
        OnBegin = "OnBeginJs",  
        onSuccess = "OnSuccessJs",  
        onFailure = "OnFailureJs",  
        onComplete = "OnCompleteJs",  
    };  
}
```



AJAX- свойства для call back function

Свойства	Описание
OnBegin	Задаёт обратный вызов перед отправкой запроса. Соотносится с событием beforeSend библиотеки jQuery. Т.е. содержит имя JavaScript функции запускаемой перед отправкой запроса на сервер. Используется если необходимо показать какой-то индикатор, т.е. не пользуясь свойством LoadingElementId, которое навешивает обработчик события на элемент, а просто самостоятельно вывести какое-то сообщение
OnSuccess	задаёт обратный вызов, который вызывается после выполнения запроса (как удачного, так и неудачного). Соотносится с событием success библиотеки jQuery. Определяется действие, которое будет происходить когда сервер возвращает нам ответ. И именно здесь мы можем обработать данные которые пришли с сервера и что-то с ними сделать, например вывести или произвести ещё какие-то операции.

AJAX- свойства для call back function

Свойства	Описание
OnFailure	задает обратный вызов, который вызывается после неудачного выполнения запроса. Соотносится с событием error библиотеки jQuery. Данная функция будет запускаться если на сервере произошли какие-то ошибки. Если сервер вернул статус код 500 или 404 мы можем обработать такие ситуации.
OnComplete	задает обратный вызов, который вызывается после удачного выполнения запроса. Соотносится с событием complete библиотеки jQuery. Функция, которая говорит о том, что асинхронная операция завершена и абсолютно не важно успешно или нет. И в этой функции например спрятать тот текст который отображали в Call Back функции OnBegin

Основы XMLHttpRequest

```
Index.cshtml*  X
@section scripts
{
    @*Ненавязчивый JavaScript для выполнения запроса на сервер без полного обновления страницы*@
    <script src="~/Scripts/jquery.unobtrusive-ajax.min.js"></script>

    <script type="text/javascript">
        @*На OnBeginJs мы просто отображаем сообщения *@
        function OnBeginJs() {
            alert("OnBegin Callback");
        }
        @*На OnSuccessJs(data) мы может получить доступ
        к данным которые пришли с сервера *@
        function OnSuccessJs(data) {
            alert("OnSuccess Callback: " + data);
        }
        @*На OnFailureJs(request, error) мы может получить
        доступ к объекту который отправлял запрос на сервер
        и получить информацию об ошибке произошедшем на сервере
        или на клиенте *@
        function OnFailureJs(request, error) {
            alert("OnFailure Callback: " + error);
        }
        @*На OnCompleteJs мы можем получить доступ
        к объекту который делал запрос на сервер
        и увидеть статус, т.е. успешно или не успешно закончилась
        асинхронная операция и в переменной status будет находиться
        значение Success либо Failure. *@
        function OnCompleteJs(request, status) {
            alert("OnComplete Callback: " + status);
        }
    </script>
}
```



Основы XMLHttpRequest

Orders

Id	Product	Quantity	Customer
----	---------	----------	----------

1	Product 1	1	Ivanov
2	Product 2	10	Petrov
3	Product 2	12	Fedorov
4	Product 3	6	Fedorov
5	Product 1	7	Petrov
6	Product 4	11	Ivanov
7	Product 2	10	Petrov
8	Product 5	2	Petrov
9	Product 1	11	Ivanov
10	Product 5	3	Fedorov
11	Product 2	3	Petrov
12	Product 1	3	Ivanov
13	Product 4	7	Petrov

Fedorov ▼

Submit

[All](#) [Ivanov](#) [Petrov](#) [Fedorov](#)
[ErrorLink](#)

Подтвердите действие на localhost:25576

OnBegin Callback

OK



Example 5

2015 © EPAM Systems

71

Основы XMLHttpRequest

Orders

Loading...

Id	Product	Quantity	Customer
3	Product 2	12	Fedorov
4	Product 3	6	Fedorov
10	Product 5	3	Fedorov

Fedorov ▼

Submit

[All](#) [Ivanov](#) [Petrov](#) [Fedorov](#)
[ErrorLink](#)

Подтвердите действие на localhost:25576 ✕

OnSuccess Callback: <tr>
<td>3</td>
<td>Product 2</td>
<td>12</td>
<td>Fedorov</td>
</tr>
<tr>
<td>4</td>
<td>Product 3</td>
<td>6</td>
<td>Fedorov</td>
</tr>

Основы XMLHttpRequest

Orders

Id	Product	Quantity	Customer
3	Product 2	12	Fedorov
4	Product 3	6	Fedorov
10	Product 5	3	Fedorov

Fedorov ▼

Submit

[All](#) [Ivanov](#) [Petrov](#) [Fedorov](#)
[ErrorLink](#)

Подтвердите действие на localhost:25576

OnComplete Callback: success

☐ Предотвратить создание дополнительных диалоговых окон на этой странице.

OK



Основы XMLHttpRequest

```
public class HomeController : Controller
{
    O references
    public ActionResult Index()...

    [HttpPost]
    O references
    public ActionResult Index(string id)...

    O references
    public ActionResult OrdersData(string id)
    {
        var data = OrdersDatabase.Orders;
        if (!string.IsNullOrEmpty(id) && id != "All")
        {
            // выполняем выборку по свойству Customer
            //если значение id не пустое и не равное "All"
            data = data.Where(e => e.Customer == id);
        }
        return PartialView(data);
    }

    O references
    public ActionResult JsonOrdersData(string id)
    {
        var data = OrdersDatabase.Orders;
        if (!string.IsNullOrEmpty(id) && id != "All")
        {
            data = data.Where(e => e.Customer == id);
        }

        return Json(data, JsonRequestBehavior.AllowGet);
    }
}
```



Example 6

Основы XMLHttpRequest

```
Index.cshtml  X
@{
    ViewBag.Title = "Orders";

    var options = new AjaxOptions()
    {
        UpdateTargetId = "tabledata",
        Url = Url.Action("JsonOrdersData"),
        LoadingElementId = "loadingIndicator",
        OnSuccess = "ParseResponse",
    };
}
```

Инициализация свойств Ajax объекта AjaxOptions

1. Обновлять необходимо tabledata
2. Запрос на сервер адресован дочернему методу JsonOrdersData
3. Указываем индикатор загрузки
4. Когда с сервера придет ответ мы запустим функцию ParseResponse



Основы XMLHttpRequest

Index.cshtml

@section scripts

{

@*Ненавязчивый JavaScript для выполнения запроса на сервер без полного обновления страницы*@

<script src="~/Scripts/jquery.unobtrusive-ajax.min.js"></script>

<script type="text/javascript">

function ParseResponse(data) {

var target = \$("#tabledata");

target.empty();

for (var i = 0; i < data.length; i++) {

target.append("<tr><td>" + data[i].Id + "</td><td>"

+ data[i].Product + "</td><td>"

+ data[i].Quantity + "</td><td>"

+ data[i].Customer + "</td></tr>");

}

}

</script>

}



Example 6

ОСНОВЫ XMLHttpRequest

Orders

Loading...

Id	Product	Quantity	Customer
1	Product 1	1	Ivanov
2	Product 2	10	Petrov
3	Product 2	12	Fedorov
4	Product 3	6	Fedorov
5	Product 1	7	Petrov
6	Product 4	11	Ivanov
7	Product 2	10	Petrov
8	Product 5	2	Petrov
9	Product 1	11	Ivanov
10	Product 5	3	Fedorov
11	Product 2	3	Petrov
12	Product 1	3	Ivanov
13	Product 4	7	Petrov

Fedorov ▼ Submit

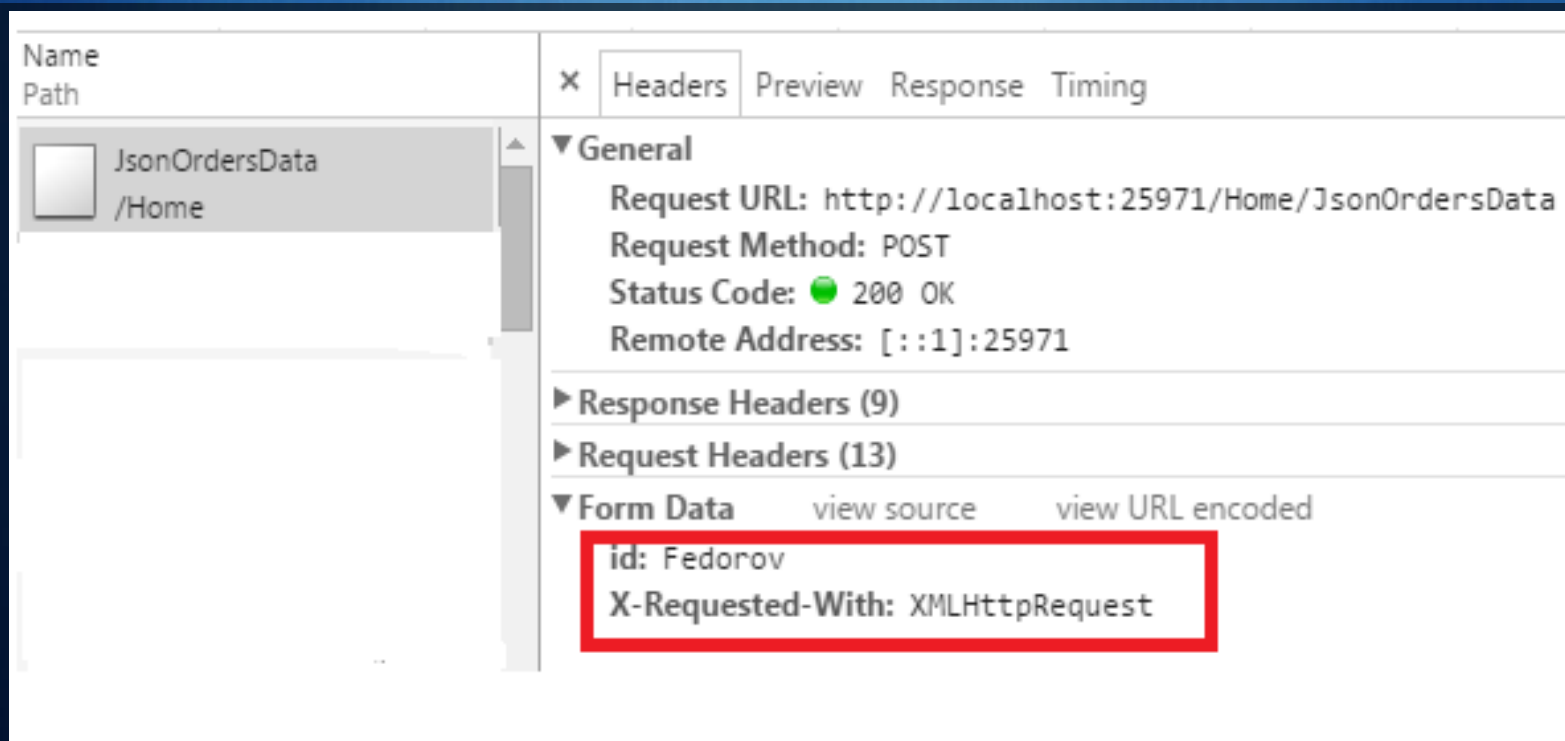
Orders

Id	Product	Quantity	Customer
3	Product 2	12	Fedorov
4	Product 3	6	Fedorov
10	Product 5	3	Fedorov

Fedorov ▼ Submit



Основы XMLHttpRequest



Example 6

Основы XMLHttpRequest

Name	Path	×	Headers	Preview	Response	Timing
	JsonOrdersData /Home	1			<pre>[{"Id":3,"Product":"Product 2","Customer":"Fedorov","Quantity":12}, {"Id":4,"Product":"Product 3","Customer":"Fedorov","Quantity":6}, {"Id":10,"Product":"Product 5","Customer":"Fedorov","Quantity":3}]</pre>	



Example 6

Фильтры

Использование фильтров

Фильтры – это атрибуты .NET, которые добавляют дополнительные этапы в конвейер обработки запроса

```
public class AdminController : Controller
{
    // ... instance variables and constructor
    [Authorize]
    public ActionResult Index()
    {
        // ...rest of action method
    }
    [Authorize]
    public ActionResult Create()
    {
        // ...rest of action method
    }
    // ... other action methods
}
```

Использование фильтров

```
[Authorize(Roles = "admin")]
public class ExampleController : Controller
{
    [ShowMessage]
    [OutputCache(Duration = 60)]
    public ActionResult Index()
    {
        // ... action method body
    }
}
```

Использование фильтров

Тип фильтра	Интерфейсы	Реализация по умолчанию	Описание
Фильтр аутентификации	IAuthenticationFilter	нет	Запускается вначале, перед любым другим фильтром или методом действия
Фильтр авторизации	IAuthorizationFilter	AuthorizeAttribute	Фильтр, определяющий, имеет ли пользователь доступ к данному ресурсу. Данный фильтр запускается после фильтра аутентификации, но до любого другого фильтра или метода действия

Использование фильтров

Тип фильтра	Интерфейсы	Реализация по умолчанию	Описание
Фильтр действия	<code>IActionFilter</code>	<code>ActionFilterAttribute</code>	Запускается до и после метода действия
Фильтр результата	<code>IResultFilter</code>	<code>ActionFilterAttribute</code>	Запускается до и после выполнения результата действия
Фильтр исключений	<code>IExceptionHandler</code>	<code>HandleErrorAttribute</code>	Запускается только в том случае, если другой фильтр, метод действия или результат действия генерирует исключение

Фильтры аутентификации

Фильтры аутентификации— срабатывают до любого другого фильтра и выполнения метода, а также тогда, когда метод уже завершил выполнение, но его результат, объект `ActionResult`, не обработан. Эти фильтры предназначены для управления аутентификацией пользователей. Фильтры аутентификации реализуют интерфейс **`IAuthenticationFilter`**

```
public interface IAuthenticationFilter
{
    void OnAuthentication(AuthenticationContext filterContext);
    void OnAuthenticationChallenge(AuthenticationChallengeContext
filterContext);
}
```

Реализация метода **`OnAuthentication`** призвана обеспечить проверку пользователя: аутентифицирован ли он в системе. Метод **`OnAuthenticationChallenge`** используется для ограничения доступа для аутентифицированного пользователя.

Фильтры авторизации

Фильтры авторизации - это фильтры, которые запускаются после фильтров аутентификации, перед любым другим фильтром или методом действия. Эти фильтры осуществляют политику авторизации, гарантируя, что методы действий могут быть вызваны только пользователями, имеющими право доступа. Фильтры авторизации реализуют интерфейс `IAuthorizationFilter`

```
public interface IAuthorizationFilter
{
    void OnAuthorization(AuthorizationContext filterContext);
}
```

Более безопасный подход - создать подкласс класса `AuthorizeAttribute`, который будет содержать самый сложный код и облегчит написание пользовательского кода авторизации

Фильтры авторизации

```
public class CustomAuthAttribute : AuthorizeAttribute
{
    private bool localAllowed;
    public CustomAuthAttribute(bool allowedParam)
    {
        localAllowed = allowedParam;
    }
    protected override bool AuthorizeCore(HttpContextBase httpContext)
    {
        if (httpContext.Request.IsLocal)
        {
            return localAllowed;
        }
        else
        {
            return true;
        }
    }
}
```

Фильтры авторизации

Класс `AuthorizeAttribute` используется как основа для пользовательского фильтра, у которого есть своя собственная реализация метода `AuthorizeCore`, который используется для выполнения общих задач авторизации.

Используя `AuthorizeAttribute` напрямую, можно определить правила авторизации с помощью двух доступных свойств этого класса

Название	Тип	Описание
Users	string	Разделенный запятыми список имен пользователей, которым разрешен доступ к методу действия.
Roles	string	Разделенный запятыми список названий ролей. Чтобы получить доступ к методу действия, пользователь должен обладать по крайней мере одной из этих ролей.

Фильтры исключений

Если при вызове метода действия было выброшено необработанное исключение, будет запущен фильтр исключений. Исключения могут поступать из:

- другого фильтра (фильтра авторизации, действия или результата)
- самого метода действия
- при выполнении результата действия

Фильтры исключений

Фильтры исключений должны реализовывать интерфейс `IExceptionHandler`

```
public interface IExceptionHandler
{
    void OnException(ExceptionContext filterContext);
}
```

Когда появится необработанное исключение, будет вызван метод `OnException`, параметром которого является объект `ExceptionContext`, наследующий `ControllerContext` и имеющий ряд полезных свойств, с помощью которых можно получить информацию о запросе

Фильтры исключений. Свойства ControllerContext

Название	Тип	Описание
Controller	ControllerBase	Возвращает объект контроллера для данного запроса
HttpContext	HttpContextBase	Обеспечивает доступ к информации о запросе и доступ к ответу
IsChildAction	bool	Возвращает true, если это дочернее действие
RequestContext	RequestContext	Предоставляет доступ к объекту HttpContext и данным маршрутизации, хотя и то, и то доступно через другие свойства
RouteData	RouteData	Возвращает данные маршрутизации для данного запроса

Фильтры исключений. Свойства ExceptionContext

В дополнение к свойствам, унаследованным от класса **ControllerContext**, класс **ExceptionContext** определяет некоторые дополнительные свойства, которые также полезны при работе с исключениями

Название	Тип	Описание
ActionDescriptor	ActionDescriptor	Предоставляет подробную информацию о методе действия
Result	ActionResult	Результат для метода действия; фильтр может отменить запрос, установив для этого свойства иное значение, кроме null
Exception	Exception	Необработанное исключение
ExceptionHandled	bool	Возвращает true, если другой фильтр отметил это исключение как обработанное

Фильтры исключений

```
public class RangeExceptionAttribute : FilterAttribute, IExceptionHandler
{
    public void OnException(ExceptionContext filterContext)
    {
        if (!filterContext.ExceptionHandled &&
            filterContext.Exception is ArgumentOutOfRangeException)
        {
            filterContext.Result = new RedirectResult("~/Content/
                                                    RangeErrorPage.html");
            filterContext.ExceptionHandled = true;
        }
    }
}

[RangeException]
public ActionResult RangeTest(int id)
{ ...}
```



Фильтры исключений

```
public class RangeExceptionAttribute: FilterAttribute, IExceptionHandler
{
    public void OnException(ExceptionContext filterContext)
    {
        if (!filterContext.ExceptionHandled &&
            filterContext.Exception is ArgumentOutOfRangeException)
        {
            var value = (int)((ArgumentOutOfRangeException)
                             filterContext.Exception).ActualValue;
            filterContext.Result = new ViewResult()
            {
                ViewName = "RangeError",
                ViewData = new ViewDataDictionary<int>(value)
            };
            filterContext.ExceptionHandled = true;
        }
    }
}
```



Встроенная обработка исключений.

Свойства HandleErrorAttribute

Название	Тип	Описание
ExceptionType	Type	Тип исключения, который обрабатываться данным фильтром. Это свойство также будет обрабатывать типы исключений, которые наследуют от указанного, но будет игнорировать все другие. По умолчанию для свойства ExceptionType указано значение System.Exception , что означает, что оно будет обрабатывать все стандартные исключения.
View	string	Имя представления, которое рендерится данным фильтром. Если значение не задано, то по умолчанию используются следующие пути: /Views/Имя_контроллера/Error.cshtml или /Views/Shared/Error.cshtml
Master	string	Имя используемой мастер-страницы

Встроенная обработка исключений

```
[HandleError(ExceptionType = typeof(ArgumentOutOfRangeException),
View = "RangeError")]
public ActionResult RangeTest(int id)
{
    if (id < 100)
    {
        throw new ArgumentOutOfRangeException("id", id, "");
    }
    else
    {
        ViewBag.Message = string.Format("value of id : {0}", id);
    }
    return View("Index");
}

<system.web>
    <customErrors mode="On"/>
    ...
</system.web>
```



Фильтры действий

Фильтры действий позволяют проконтролировать входной контекст запроса при доступе к действию, а также выполнить определенные действия по завершению работы метода действий. Фильтр действий должен реализовать интерфейс `IActionFilter`:

```
public interface IActionFilter
{
    void OnActionExecuting(ActionExecutingContext filterContext);
    void OnActionExecuted(ActionExecutedContext filterContext);
}
```

*called before
calling the action*



*called after
calling the action*



Фильтры действий. Свойства ActionExecutingContext

Свойство	Тип	Описание
ActionDescriptor	ActionDescriptor	Предоставляет информацию о вызываемом методе действия
Result	ActionResult	Результат метода действий

Фильтры действий. Свойства ActionExecutedContext

Свойство	Тип	Описание
ActionDescriptor	ActionDescriptor	Предоставляет информацию о вызываемом методе действия
Result	ActionResult	Результат метода действий
Canceled	bool	Хранит значение, показывающее, отменен ли вызов действия. Если имеет значение true, если вызов действия был отменен другим фильтром
Exception	Exception	Возвращает исключение, выбрасываемое данным методом действий или другим фильтром
ExceptionHandled	bool	Хранит значение, показывающее, обработано ли исключение

Фильтры результатов

Фильтры результатов во многом похожи на фильтры действий, поскольку так же могут срабатывать как до возвращения результата действия, так и после. Фильтры результатов реализуют интерфейс `IResultFilter`:

```
public interface IResultFilter
{
    void OnResultExecuting(ResultExecutingContext filterContext);
    void OnResultExecuted(ResultExecutedContext filterContext);
}
```

before

after

Фильтры действий и результатов

Фильтры действий и результатов объединены в одну реализацию - абстрактный класс **ActionFilterAttribute**, который объединяет черты обоих фильтров:

```
public abstract class ActionFilterAttribute : FilterAttribute,
                                           IActionFilter,
                                           IResultFilter
{
    public virtual void OnActionExecuting(ActionExecutingContext
                                           filterContext) {}
    public virtual void OnActionExecuted(ActionExecutedContext
                                           filterContext) {}
    public virtual void OnResultExecuting(ResultExecutingContext
                                           filterContext) {}
    public virtual void OnResultExecuted(ResultExecutedContext
                                           filterContext) {}
}
```



Регистрация фильтров

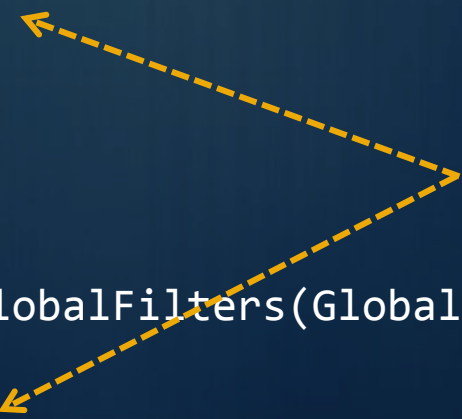
Существует три уровня подключения фильтров:

- Глобальный уровень (**Global Level**)
- Уровень контроллера (**Controller level**)
- Уровень метода действия (**Action method level**)

Регистрация фильтров

```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
    }
}

public class FilterConfig
{
    public static void RegisterGlobalFilters(GlobalFilterCollection filters)
    {
        filters.Add(new HandleErrorAttribute());
    }
}
```



Global Level

Регистрация фильтров

```
[HandleError] ←----- Controller level
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }
}
```

```
public class HomeController : Controller
{
    [HandleError] ←----- Action method level
    public ActionResult Index()
    {
        return View();
    }
}
```


Использование встроенных фильтров

Фильтр **RequireHttps** - заставляет использовать протокол HTTPS, а браузер перенаправит пользователя на то же действие, только с префиксом https, применяется только к GET-запросам.

Фильтр **OutputCache** - сообщает MVC-фреймворку кэшировать вывод метода действия, чтобы полученный контент можно было в дальнейшем использовать повторно, что может увеличить производительность, особенно когда идет речь о выборке из базы данных, которая может занимать значительное время. С помощью параметра `Duration` мы можем настроить время (в секундах):

```
[OutputCache (Duration=360)]
public ActionResult Index()
{
    //.....
}
```

Использование встроенных фильтров

Фильтр `ValidateAntiForgeryToken` предназначен для противодействия подделке межсайтовых запросов, производя верификацию токенов при обращении к методу действия.

`[ValidateAntiForgeryToken]`

```
public ActionResult Login(LoginModel model, string returnUrl)
{
    if (ModelState.IsValid && WebSecurity.Login(model.Email,
        model.Password, persistCookie: model.RememberMe))
    {
        return RedirectToLocal(returnUrl);
    }

    ModelState.AddModelError("", "Invalid login or password");
    return View(model);
}
```