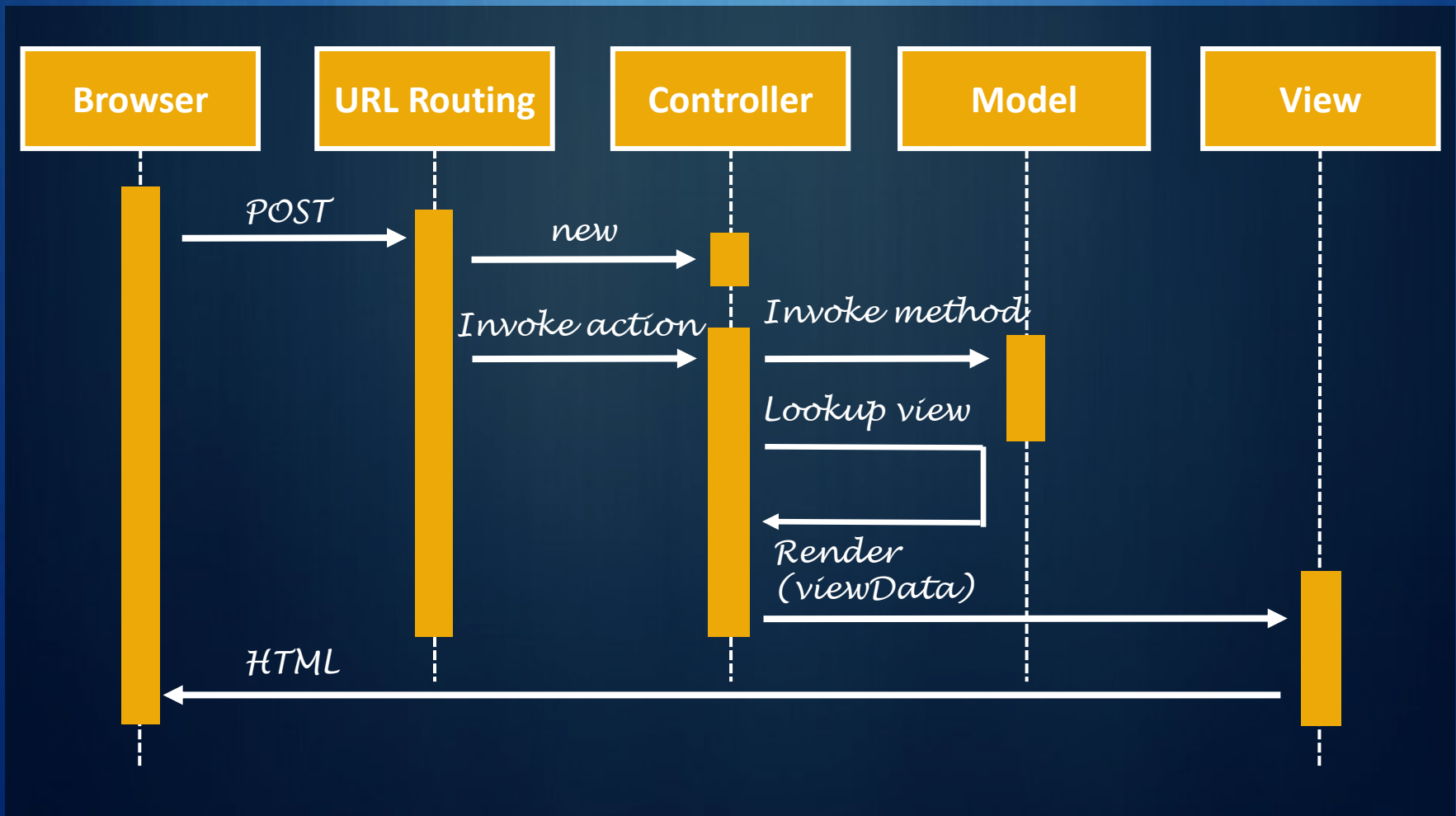


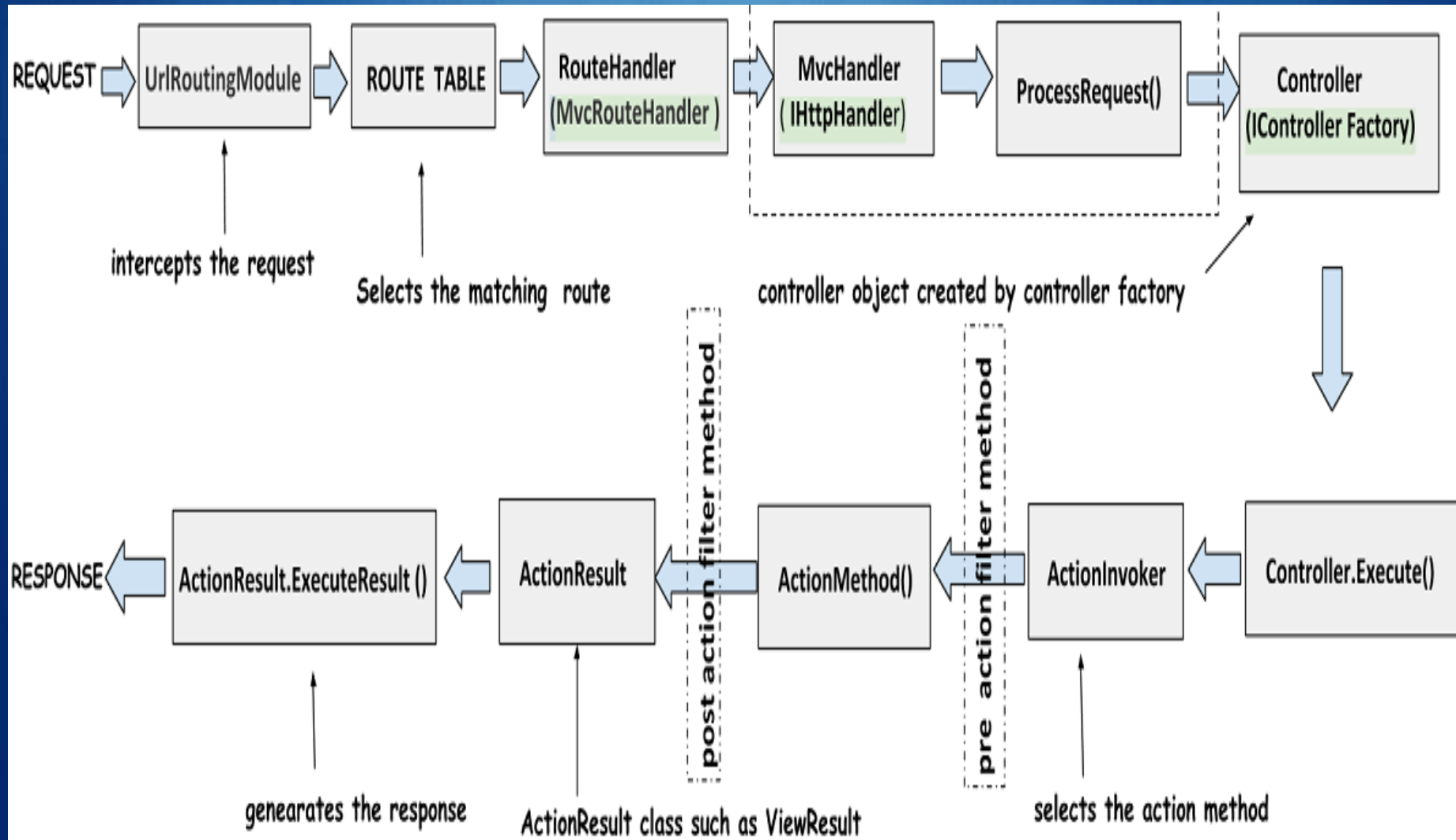
ASP MVC

v^*

Цикл запроса в MVC



Цикл запроса в MVC



Маршрутизация

Введение в маршрутизацию

RouteConfig.cs X Test.cshtml Index.cshtml HomeController.cs

MvcApplication2.RouteConfig RegisterRoutes(RouteCollection routes)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace MvcApplication2
{
    1 reference
    public class RouteConfig
    {
        1 reference
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
            );
        }
    }
}
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'MvcApplication2' (1 project)

MvcApplication2

Properties

References

App_Data

App_Start

BundleConfig.cs

FilterConfig.cs

RouteConfig.cs

WebApiConfig.cs

Content

Controllers

HomeController.cs

Models

Scripts

Views

Home

Index.cshtml

Test.cshtml

Shared

Введение в маршрутизацию

URL

`http://example.com/users/edit/5`

Route

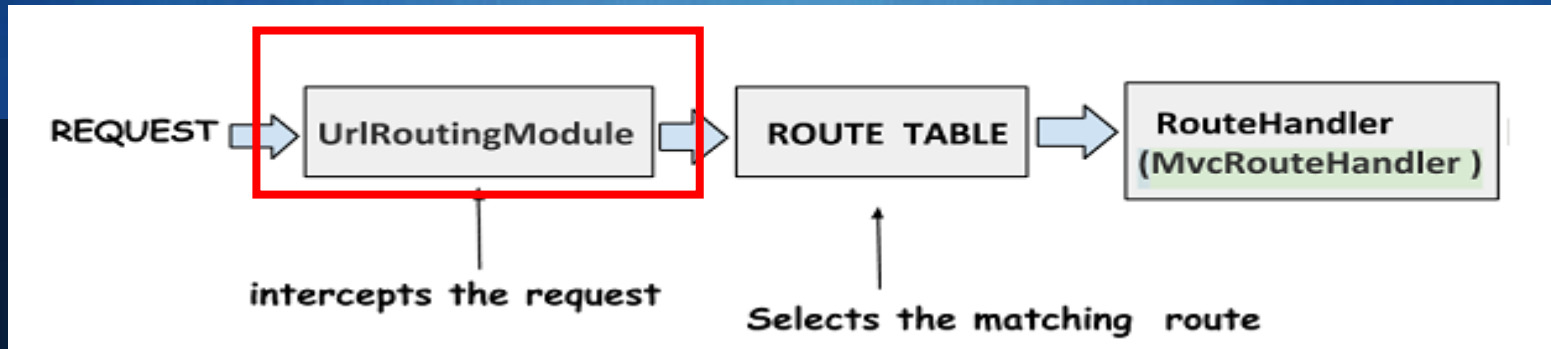
`{controller}/{action}/{id}`

Controller:
`UserController`

action::
`Edit`

id:
`5`

Введение в маршрутизацию



URL

`http://example.com/users/edit/5`

Route

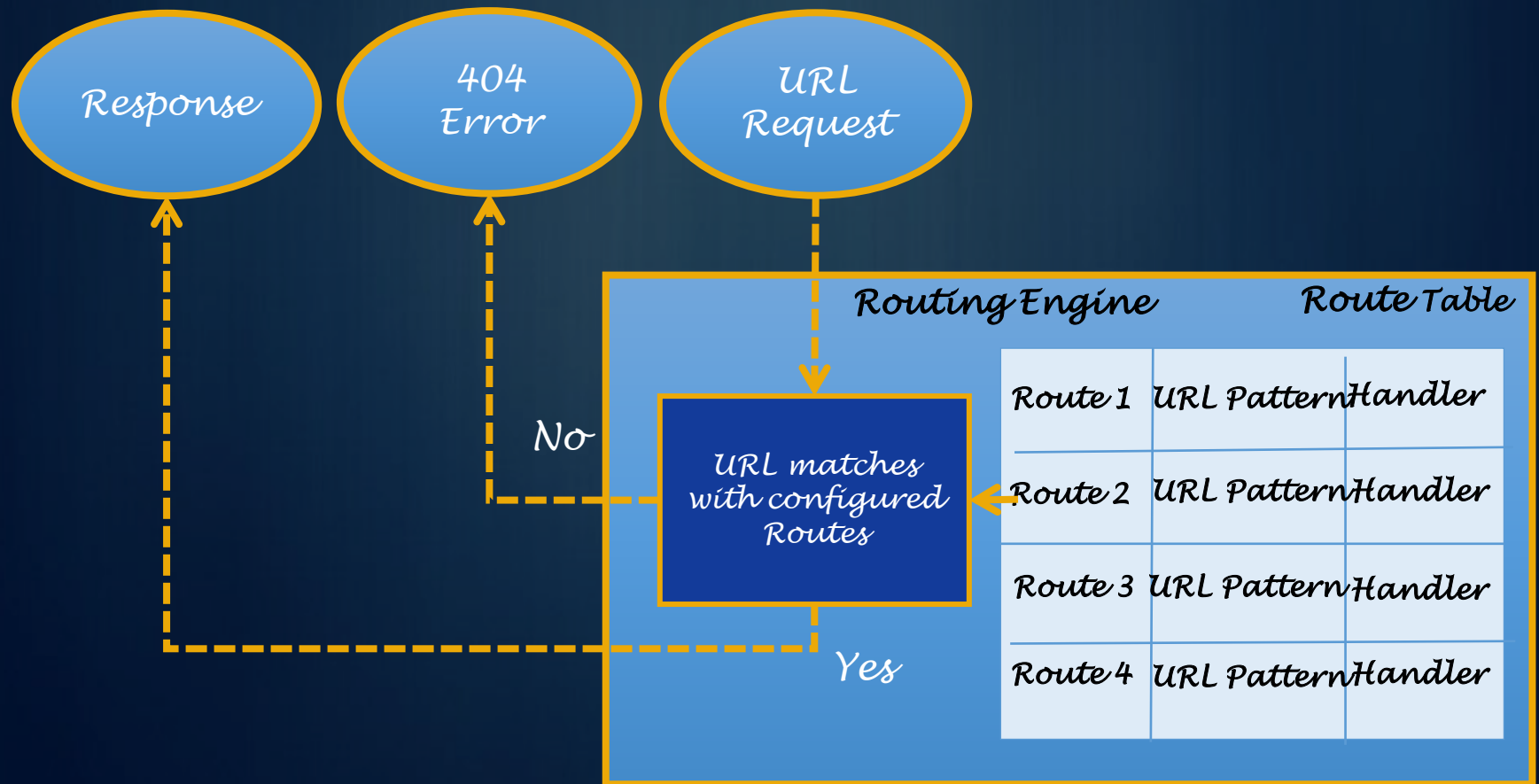
`{controller}/{action}/{id}`

Controller:
`UsersController`

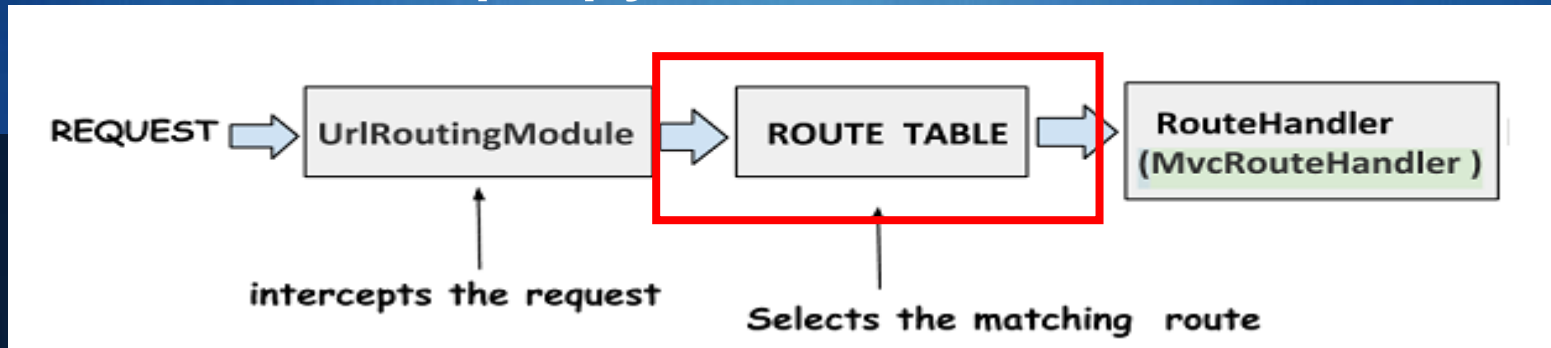
action::
`Edit`

id:
`5`

Введение в маршрутизацию



Введение в маршрутизацию



URL

`http://example.com/users/edit/5`

Route

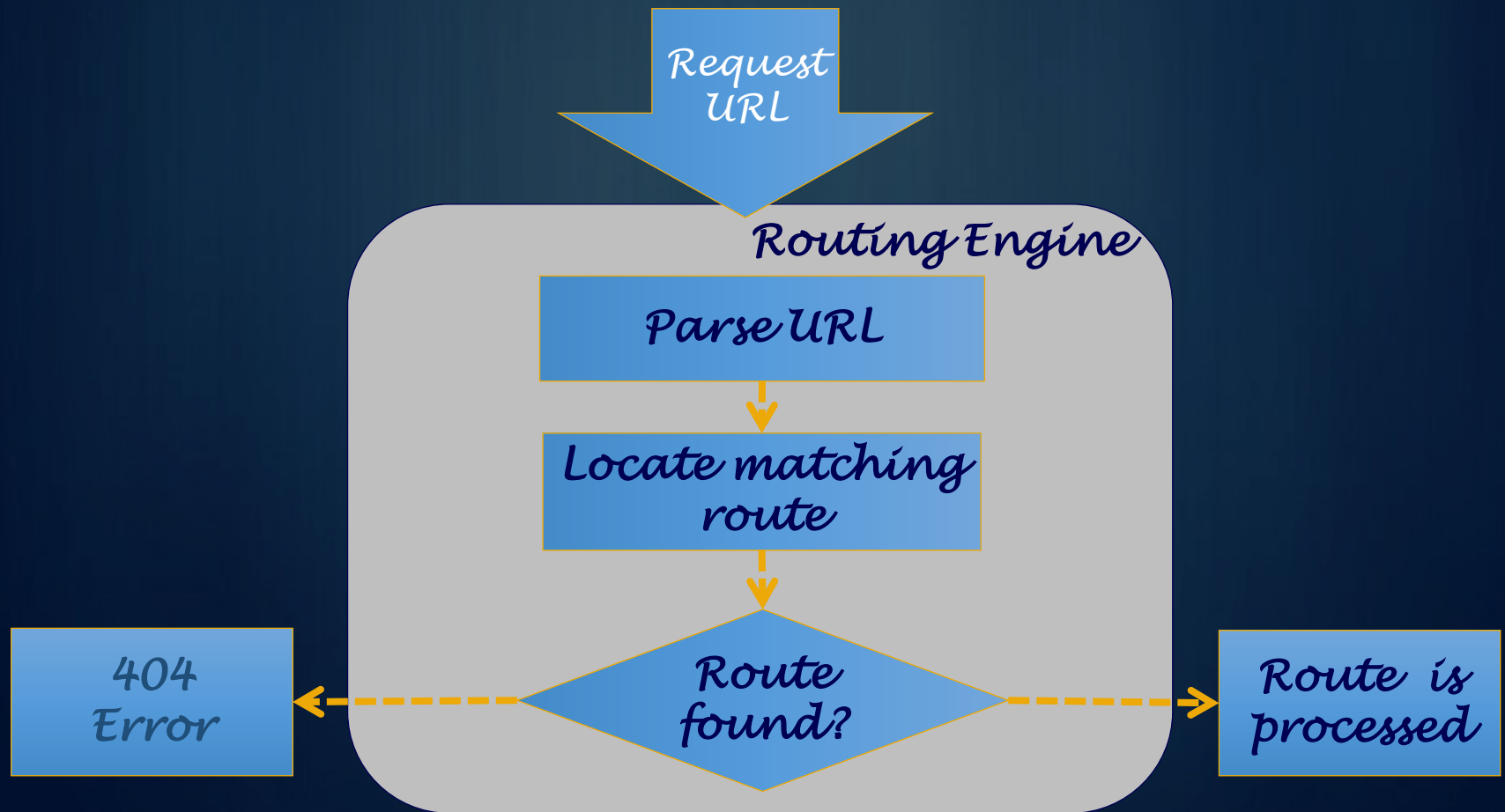
`{controller}/{action}/{id}`

Controller:
`UsersController`

action::
`Edit`

id:
`5`

Введение в маршрутизацию



Введение в маршрутизацию

```
routes.MapRoute(  
    name: "Guestbook",  
    url: "{controller}/{action}/{id}",  
    defaults: new {  
        controller = "Guestbook",  
        action = "Index",  
        id = UrlParameter.Optional  
    });
```

Route name (points to "Guestbook")

URL Pattern (points to "{controller}/{action}/{id}")

Defaults for Route (points to the defaults object)

URL	Controller	Action	Id
http://localhost	GuestbookController	Index	null
http://localhost/Guestbook	GuestbookController	Index	null
http://localhost/Guestbook/Index/123	GuestbookController	Index	123

Введение в маршрутизацию

```
routes.MapRoute(  
    name: "Guestbook",  
    url: "{controller}/{action}/{id}",  
    defaults: new {  
        controller = "Guestbook",  
        action = "Index",  
        id = UrlParameter.Optional  
    });
```

Route name (points to "Guestbook")

URL Pattern (points to "{controller}/{action}/{id}")

Defaults for Route (points to the defaults object)

URL	Controller	Action	Id
http://localhost	GuestbookController	Index	null
http://localhost/Guestbook	GuestbookController	Index	null
http://localhost/Guestbook/Index/123	GuestbookController	Index	123

Введение в маршрутизацию

```
routes.MapRoute(  
    name: "Guestbook",  
    url: "{controller}/{action}/{id}",  
    defaults: new {  
        controller = "Guestbook",  
        action = "Index",  
        id = UrlParameter.Optional  
    });
```

Route name (points to "Guestbook")

URL Pattern (points to "{controller}/{action}/{id}")

Defaults for Route (points to the defaults object)

URL	Controller	Action	Id
http://localhost	GuestbookController	Index	null
http://localhost/Guestbook	GuestbookController	Index	null
http://localhost/Guestbook/Index/123	GuestbookController	Index	123

Введение в маршрутизацию

```
routes.MapRoute(  
    name: "Guestbook",  
    url: " {controller}/{action }/{id}/{*catchall}",  
    defaults: new {
```

*Defaults for
Route*

```
        controller = "Guestbook",  
        action = "Index",  
        id = UrlParameter.Optional});
```

Route name

URL Pattern

URL

id

catchall

http://localhost/Guestbook/Index/1/2/hello/56

1

2/hello/56

http://localhost/Guestbook/Index

null

null

http://localhost/Guestbook/Index/123

123

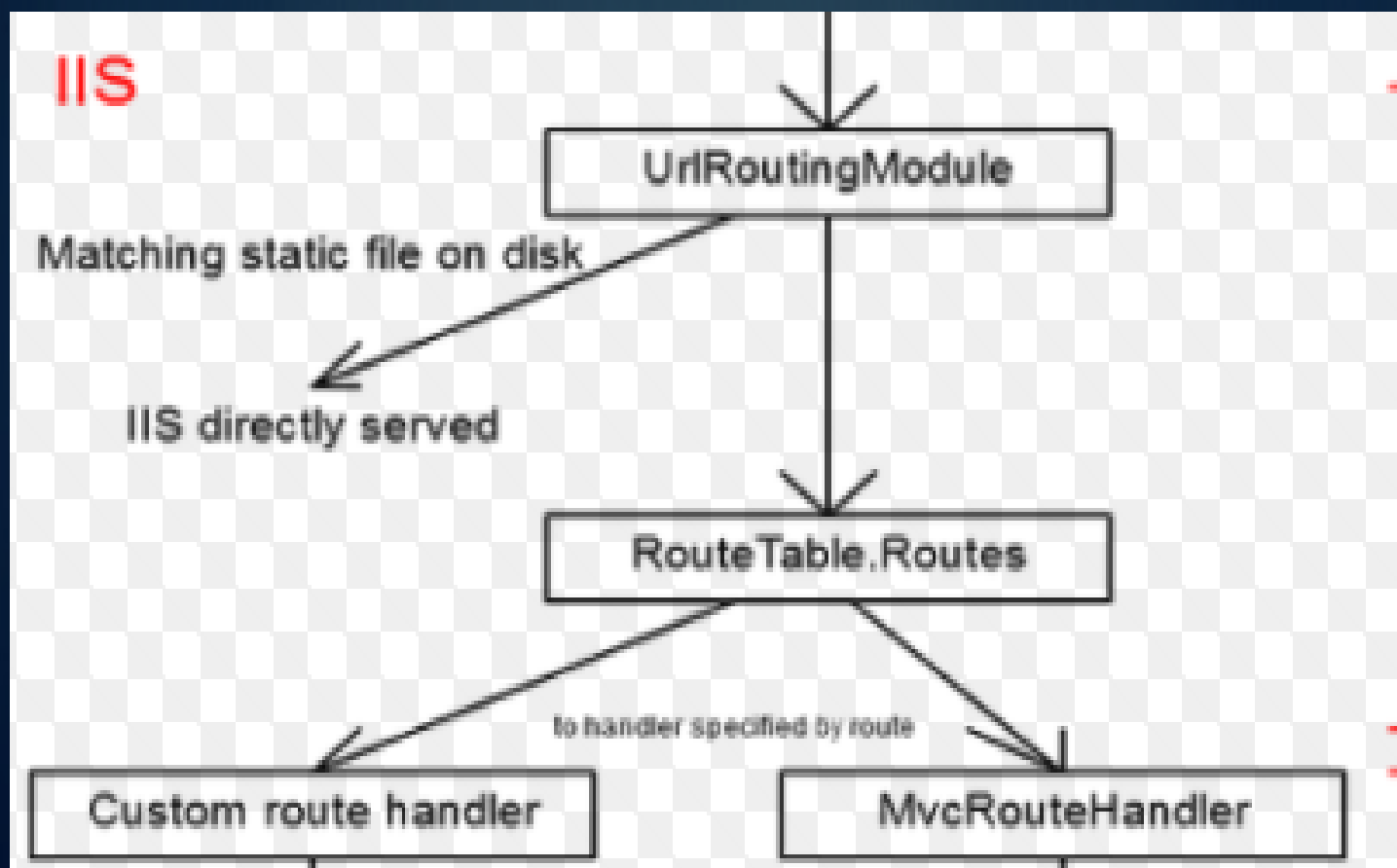
null

Введение в маршрутизацию

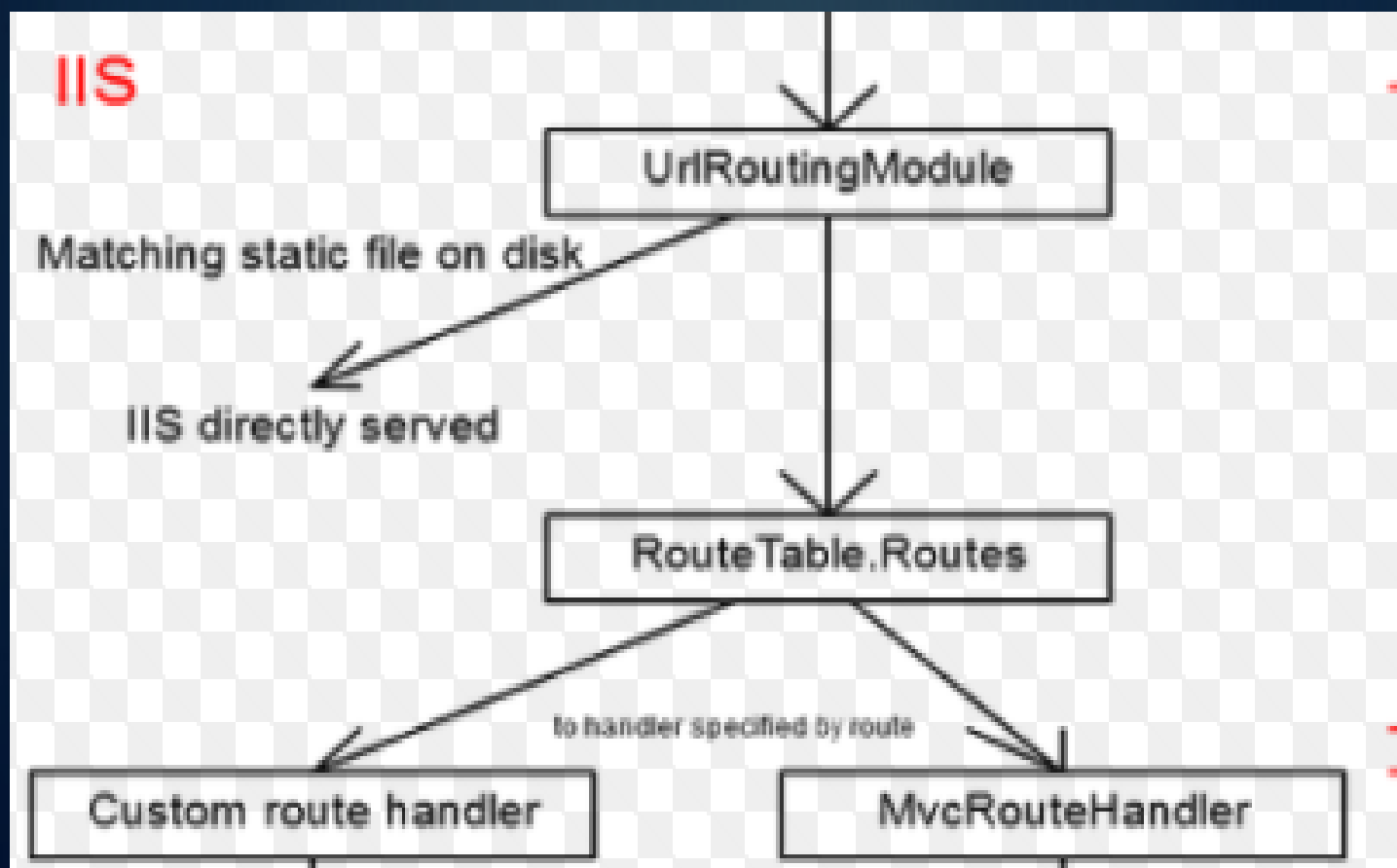
```
routes.MapRoute(
    name: "Guestbook",
    url: " {controller}/{action }/{id}/{*catchall}",
    defaults: new {
        controller = "Home",
        action = "Index",
        id = UrlParameter.Optional
    },
    constraints: new {
        controller = "^H.*",
        action = "^Index$|^About$",
        id = "^\\d*$" });
```

URL	controller	action	id
http://localhost/Home/About/100	HomeController	About	100
http://localhost/Hello/Foo		Error	
http://localhost/Hello/About	HelloController	About	null

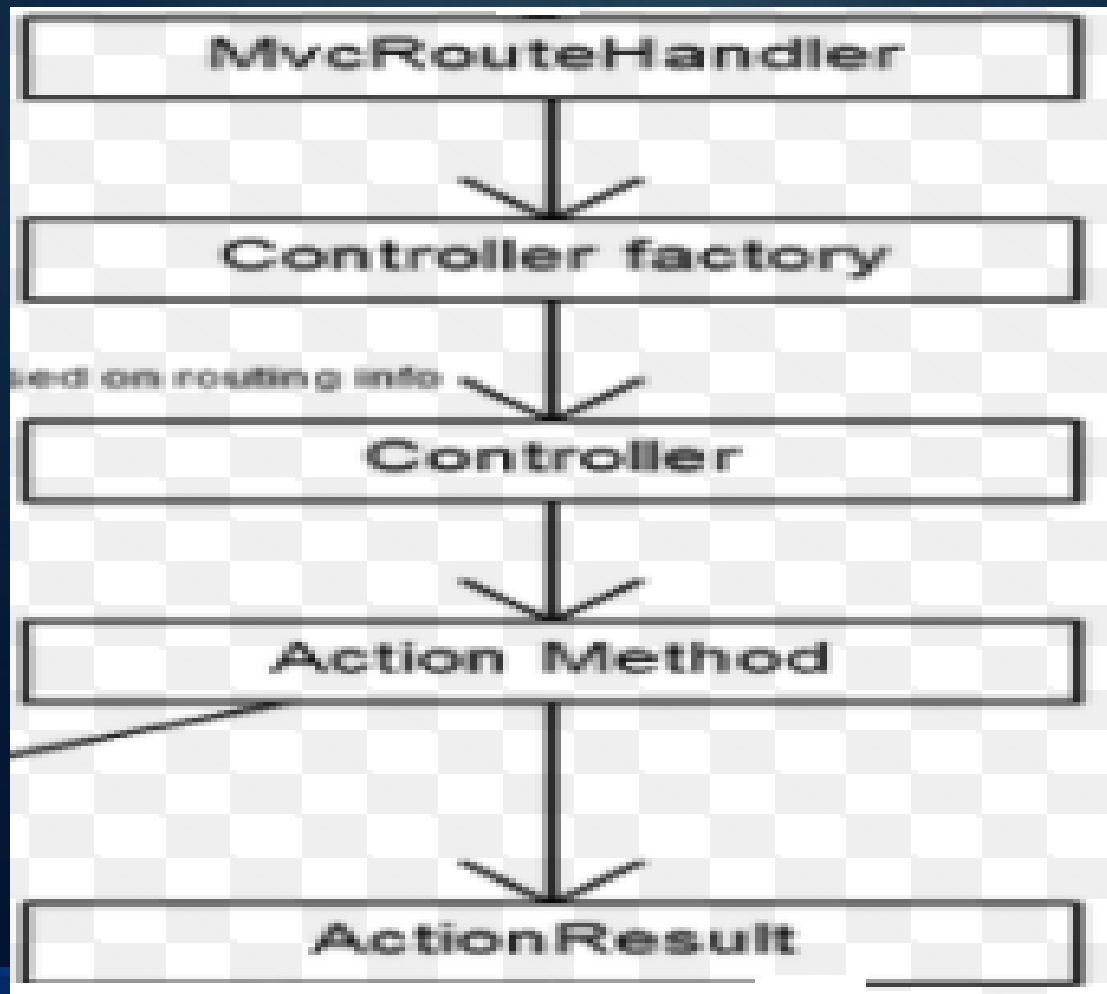
Входящая и исходящая маршрутизация



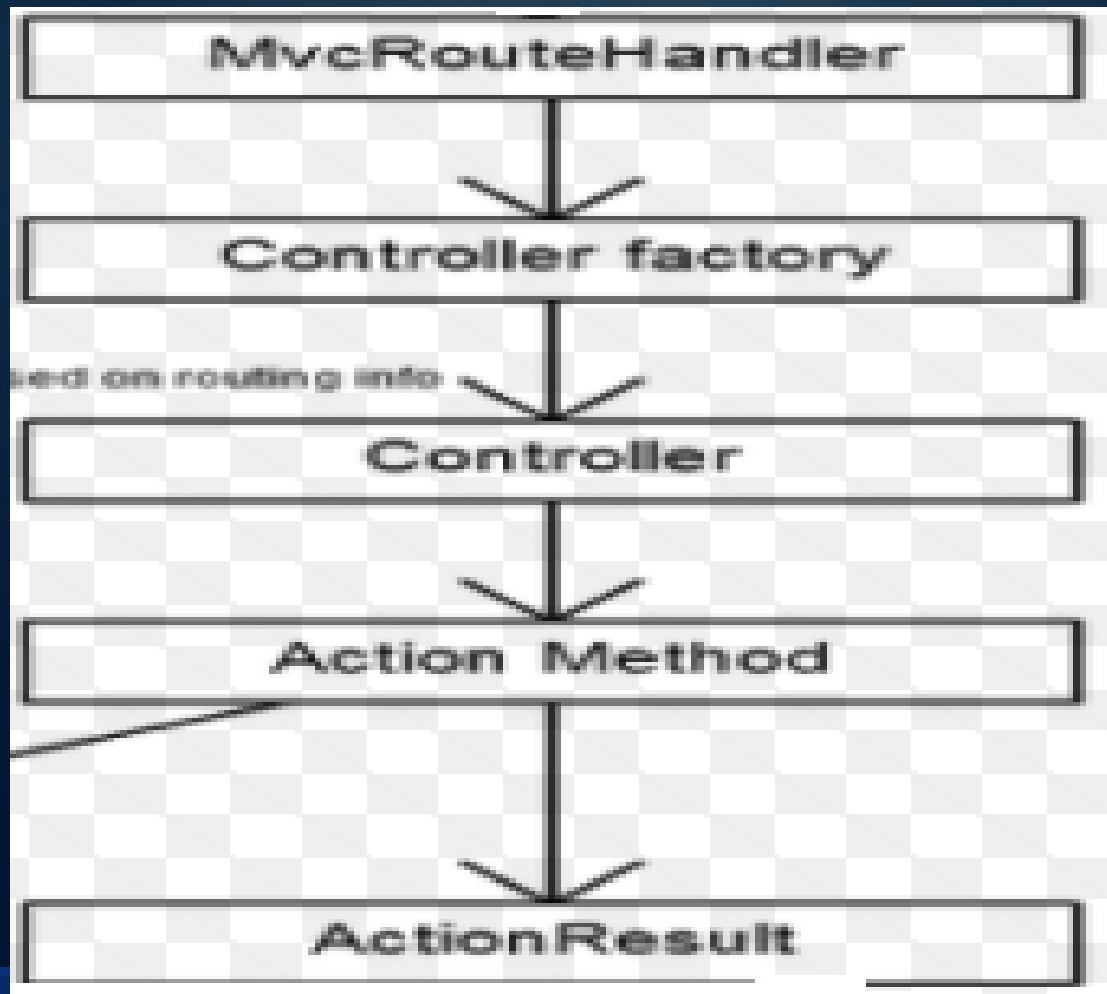
Входящая и исходящая маршрутизация



Входящая и исходящая маршрутизация



Входящая и исходящая маршрутизация



Контроллеры

Основы контроллеров

DefaultControllerFactory – это встроенная фабрика контроллеров, которой вполне достаточно, для большинства приложений.

Когда фабрика контроллеров получает запрос от системы маршрутизации, она смотрит на данные маршрутизации, чтобы определить соответствующий контроллер, и пытается найти класс в веб-приложении, удовлетворяющий следующим критериям:

- Класс должен быть открытым
- Класс не должен быть абстрактным
- Класс не должен принимать обобщенные параметры
- Имя класса должно завершаться словом **Controller**
- Класс должен реализовывать интерфейс **Controller**

Основы контроллеров

Контроллер - это объект, который обрабатывает поступившие к приложению запросы.

В MVC мы можем создать контроллер двумя способами:

1. Создать класс и реализовать интерфейс *System.Web.Mvc.IController*
2. Создать класс и наследоваться от класса *System.Web.Mvc.Controller*

Создание контроллера на основе интерфейса IController

```
using System;
using System.Web.Routing;

namespace System.Web.Mvc
{
    public interface IController
    {
        void Execute(RequestContext requestContext);
    }
}
```

Интерфейс IController

1. находится в пространстве System.Web.Mvc, который
2. используется всеми контроллерами Mvc приложения и содержит
3. единственный метод Execute(), который в качестве параметра принимает тип данных RequestContext и ничего не возвращает. При этом данный метод вызывается автоматически контроллером.



Создание контроллера на основе интерфейса IController

```
namespace System.Web.Routing
{
    public class RequestContext
    {
        public RequestContext();
        public RequestContext(HttpContextBase httpContext, RouteData routeData);

        public virtual HttpContextBase HttpContext { get; set; }
        public virtual RouteData RouteData { get; set; }
    }
}
```

Класс *RequestContext* – это обертка над *HttpContext*, которая применяется в MVC приложениях.

RequestContext содержит виртуальные автосвойства:

1. *HttpContext*, типа *HttpContextBase*, из которого можно извлечь информацию о текущем запросе, ссылке на объект сессии и т.д.
2. *RouteData* которое содержит информацию о текущем маршруте.



Создание контроллера на основе интерфейса IController

```
public abstract class HttpContextBase : IServiceProvider
{
    protected HttpContextBase();

    public virtual Exception[] AllErrors { get; }
    public virtual bool AllowAsyncDuringSyncStages { get; set; }
    public virtual HttpApplicationStateBase Application { get; }
    public virtual HttpApplication ApplicationInstance { get; set; }
    public virtual AsyncPreloadModeFlags AsyncPreloadMode { get; set; }
    public virtual Cache Cache { get; }
    public virtual IHttpHandler CurrentHandler { get; }
    public virtual RequestNotification CurrentNotification { get; }
    public virtual Exception Error { get; }
    public virtual IHttpHandler Handler { get; set; }
    public virtual ProfileBase Profile { get; }
    public virtual HttpRequestBase Request { get; }
    public virtual HttpResponseBase Response { get; }
    public virtual HttpServerUtilityBase Server { get; }
    public virtual HttpSessionStateBase Session { get; }
    public virtual bool SkipAuthorization { get; set; }
}
```

HttpContextBase – это абстрактный класс в котором находятся свойства необходимые для обработки запросов (Cache, Request, Response, Server, Session).



Создание контроллера на основе интерфейса IController

```
using System;
using System.Runtime.CompilerServices;

namespace System.Web.Routing
{
    public class RouteData
    {
        public RouteData();
        public RouteData(RouteBase route, IRouteHandler routeHandler);

        public RouteValueDictionary DataTokens { get; }
        public RouteBase Route { get; set; }
        public IRouteHandler RouteHandler { get; set; }
        public RouteValueDictionary Values { get; }

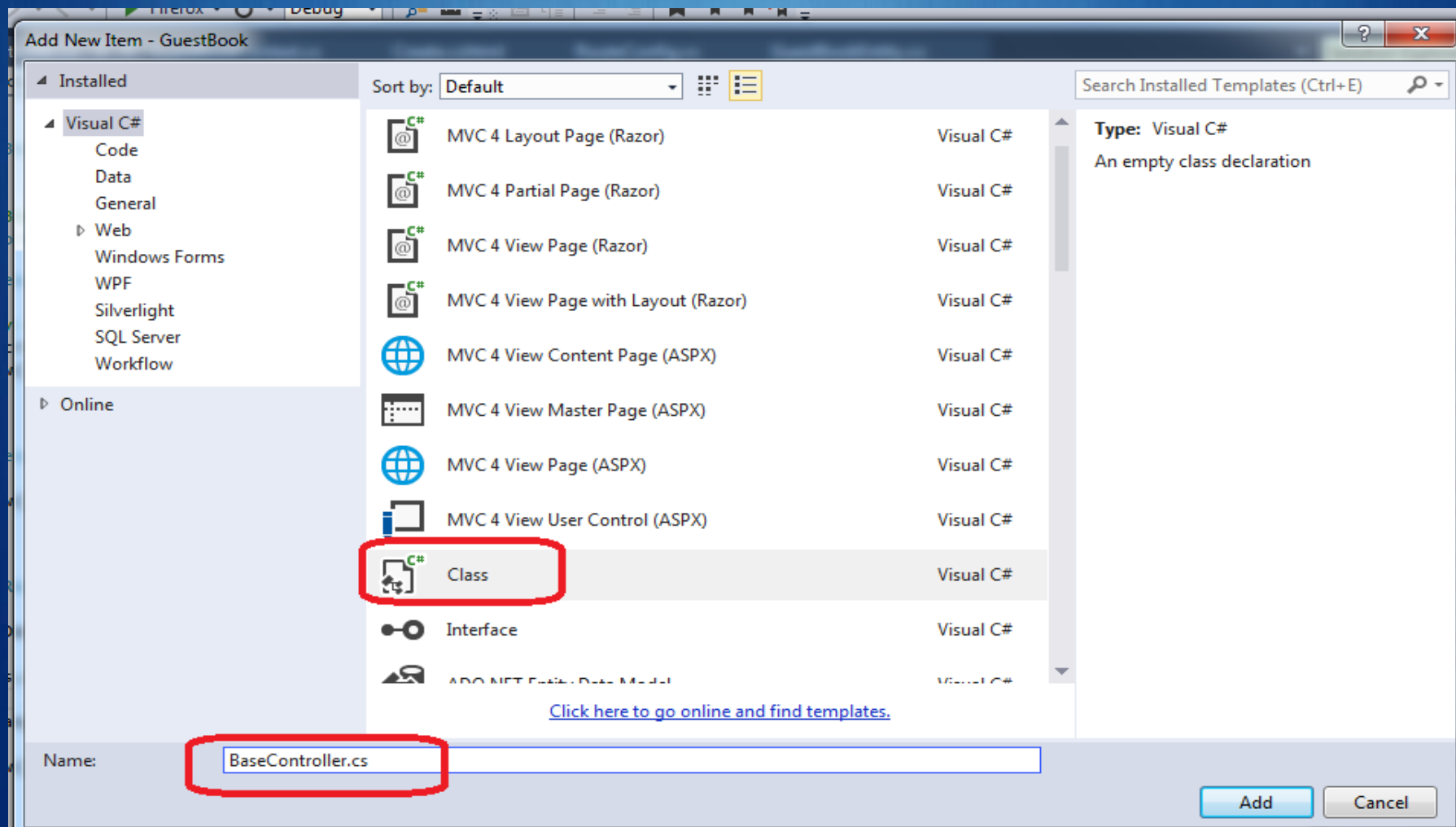
        public string GetRequiredString(string valueName);
    }
}
```

RouteData – это класс в которой находится информация связанная с маршрутизацией.

1. **Values**, который является коллекцией, используется для получения значений из сегментных переменных.
2. **RouteHandler** – это информация о том какой объект, т.е. какой Handler, сейчас обрабатывает запрос.
3. **Route** – это информация о маршруте, который был получен из таблицы маршрутов, зарегистрированных при настройке таблицы маршрутизации в файле RouteConfig.



Создание контроллера на основе интерфейса IController



Example 1

Создание контроллера на основе интерфейса IController

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Web;  
using System.Web.Mvc;
```

```
namespace GuestBook.Controllers
```

```
{
```

0 references

```
public class BaseController:IController
```

```
{
```

```
}
```

```
}
```



Implement interface 'IController'

Explicitly implement interface 'IController'

```
public class BaseController:IController  
{  
  
    0 references  
    public void Execute(System.Web.Routing.RequestContext requestContext)  
    {  
        throw new NotImplementedException();  
    }  
}
```



Создание контроллера на основе интерфейса IController

```
public class BaseController:IController  
{
```

0 references

```
public void Execute(System.Web.Routing.RequestContext requestContext)  
{  
    requestContext.HttpContext.Response.Write("<h1>Welcome from IController implemnt!</h1>");  
  
    string controller = requestContext.RouteData.Values["controller"].ToString();  
    string action = requestContext.RouteData.Values["action"].ToString();  
  
    requestContext.HttpContext.Response.Write(string.Format(  
        "controller:{0} action:{1}", controller, action));  
}  
}
```



Создание контроллера на основе интерфейса IController

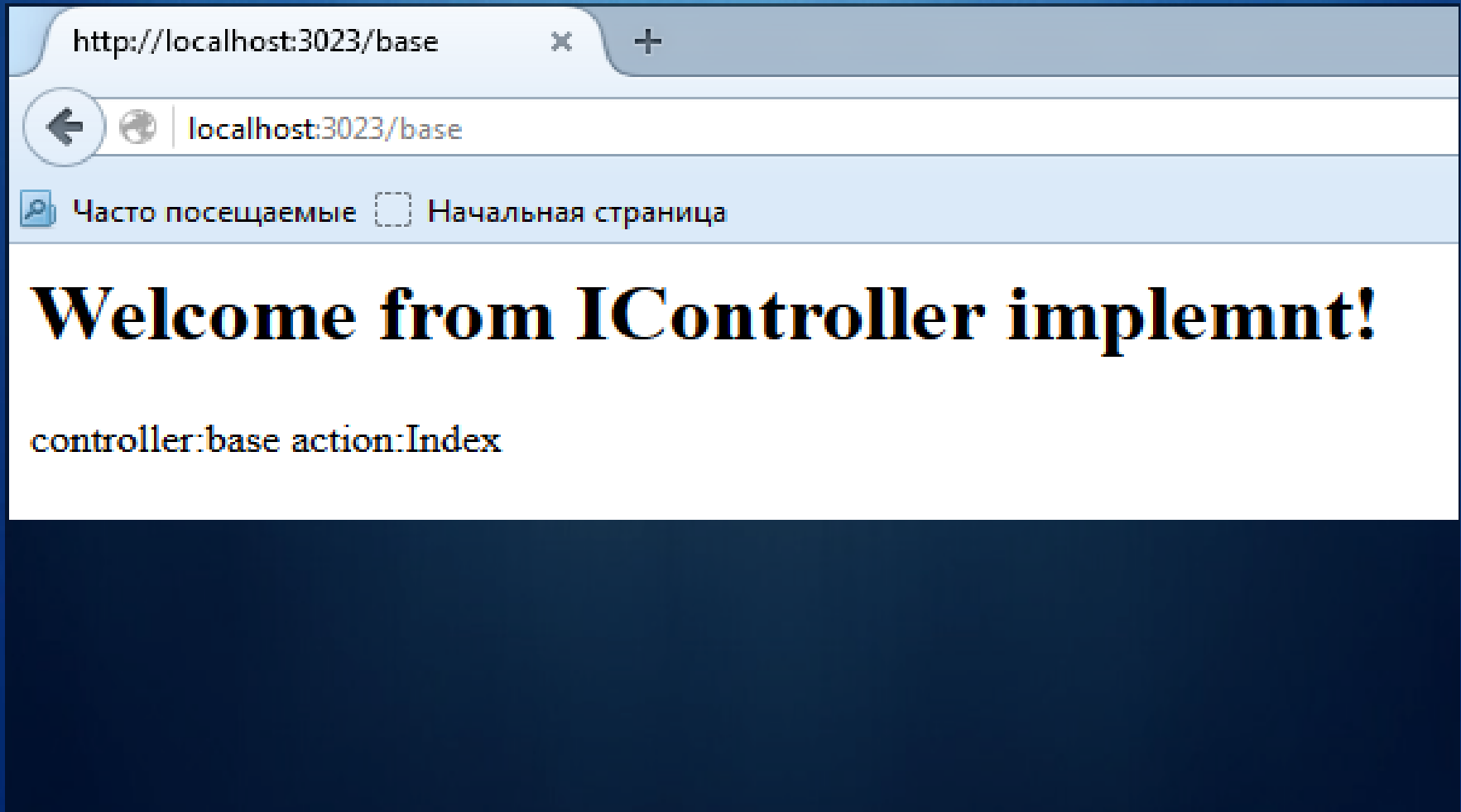
```
public class BaseController:IController  
{
```

0 references

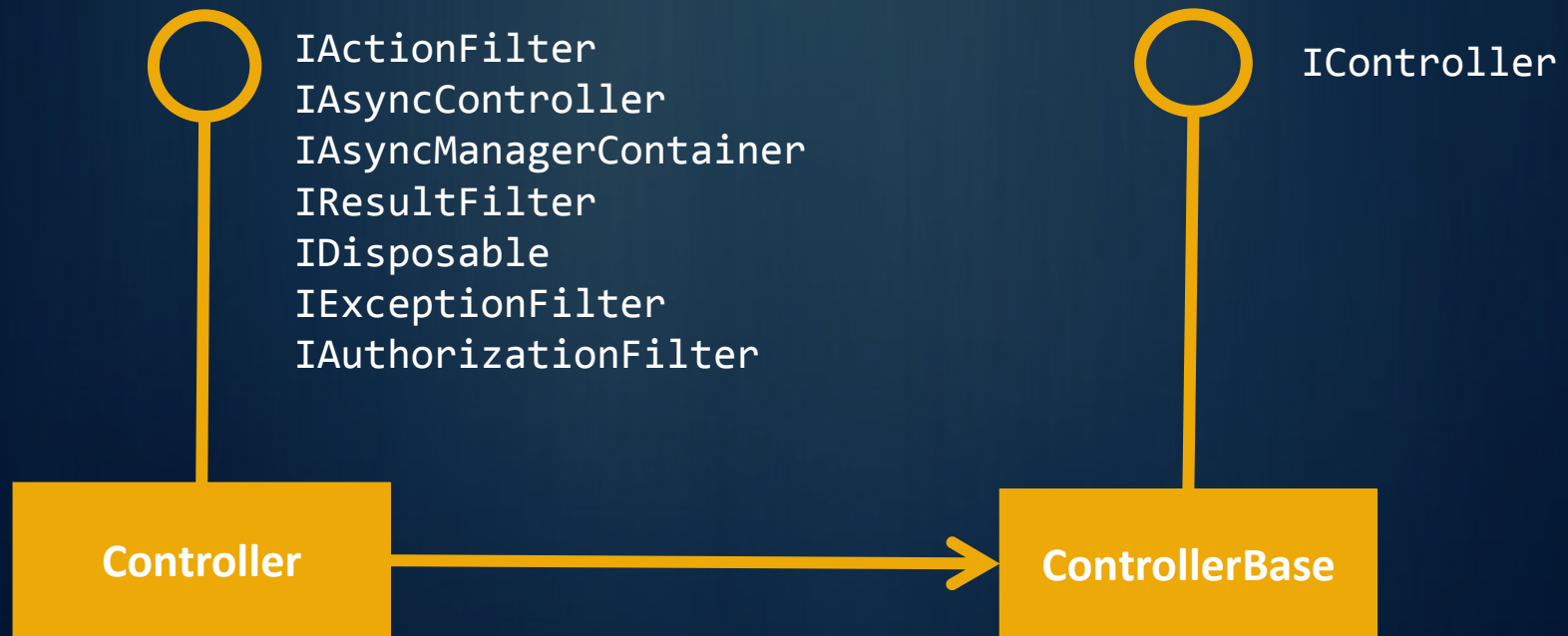
```
public void Execute(System.Web.Routing.RequestContext requestContext)  
{  
    requestContext.HttpContext.Response.Write("<h1>Welcome from IController implemnt!</h1>");  
  
    string controller = requestContext.RouteData.Values["controller"].ToString();  
    string action = requestContext.RouteData.Values["action"].ToString();  
  
    requestContext.HttpContext.Response.Write(string.Format(  
        "controller:{0} action:{1}", controller, action));  
}  
}
```



Создание контроллера на основе интерфейса IController



Создание контроллера на основе класса Controller



Создание контроллера на основе класса Controller

```
namespace System.Web.Mvc
{
    public abstract class ControllerBase : IController
    {
        protected ControllerBase();

        public ControllerContext ControllerContext { get; set; }
        public TempDataDictionary TempData { get; set; }
        public bool ValidateRequest { get; set; }
        public IValueProvider ValueProvider { get; set; }
        public dynamic ViewBag { get; }
        public ViewDataDictionary ViewData { get; set; }

        protected virtual void Execute(RequestContext requestContext);
        protected abstract void ExecuteCore();
        protected virtual void Initialize(RequestContext requestContext);
    }
}
```



Создание контроллера на основе класса Controller

```
public abstract class Controller : ControllerBase, IActionFilter, IAuthorizationFilter, IDisposable,
{
    IExceptionHandler, IResultFilter, IAsyncController, IController, IAsyncManagerContainer

    protected Controller();

    public IActionInvoker ActionInvoker { get; set; }
    public AsyncManager AsyncManager { get; }
    protected internal ModelBinderDictionary Binders { get; set; }
    protected virtual bool DisableAsyncSupport { get; }
    public HttpContextBase HttpContext { get; }
    public ModelStateDictionary ModelState { get; }
    public ProfileBase Profile { get; }
    public HttpRequestBase Request { get; }
    public HttpResponseBase Response { get; }
    public RouteData RouteData { get; }
    public HttpServerUtilityBase Server { get; }
    public HttpSessionStateBase Session { get; }
    public ITempDataProvider TempDataProvider { get; set; }
    public UrlHelper Url { get; set; }
    public IPPrincipal User { get; }
    public ViewEngineCollection ViewEngineCollection { get; set; }
```

При создании контроллера путем наследования от базового класса Controller, то у нас появляется возможность обратиться к таким объектам как Request, Response, RouteData, HttpContext, Server.

Каждое из данных свойств предоставляет информацию о различных аспектах запроса. Эти свойства получают различные типы данных из экземпляра запроса ControllerContext (который может быть доступным через свойство Controller.ControllerContext)



Создание контроллера на основе класса Controller

```
public ActionResult Index()
{
    #region

    // Свойства контроллера для доступа к информации о запросе.
    // Request - данные о текущем HTTP запросе.
    // Response - данные о текущем HTTP ответе.
    // RouteData - данные маршрутизации для текущего запроса.
    // HttpContext - получение специфической информации о текущем HTTP запросе.
    // Server - объект с методами для обработки HTTP запроса.

    string userName = User.Identity.Name;
    string machineName = Server.MachineName;
    string clientIp = Request.UserHostAddress;

    string formData = Request.Form["data"];
    string queryStringData = Request.QueryString["data"];
    HttpCookie cookie = Request.Cookies["cookieName"];

    return Content(string.Format("machineName: {0} clientIp: {1}", machineName, clientIp));
}
```

1. User – в котором находится информация о объекте текущего пользователя, т.е. identity (идентичность, подлинность) текущего пользователя, с помощью которого можно получить логин пользователя, его роль в системе
2. Server - информация о текущем сервере, свойство MachineName это имя текущего сервера
метод GetLastError() - информация о последнем необработанном исключении в приложении
методы HtmlDecode() или HtmlEncode() кодировать и декодировать html код



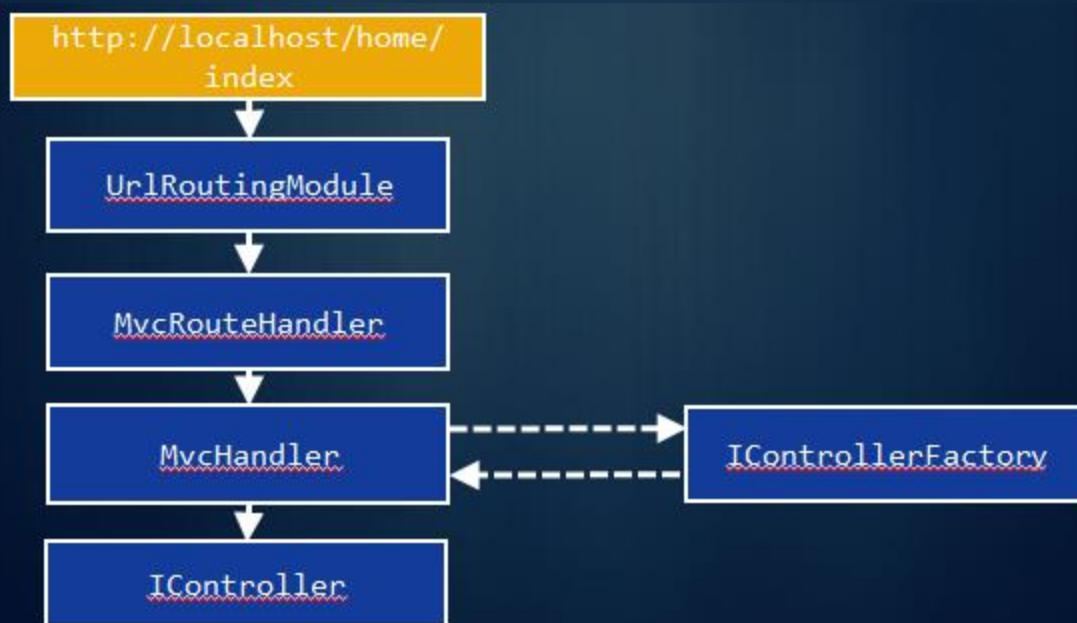
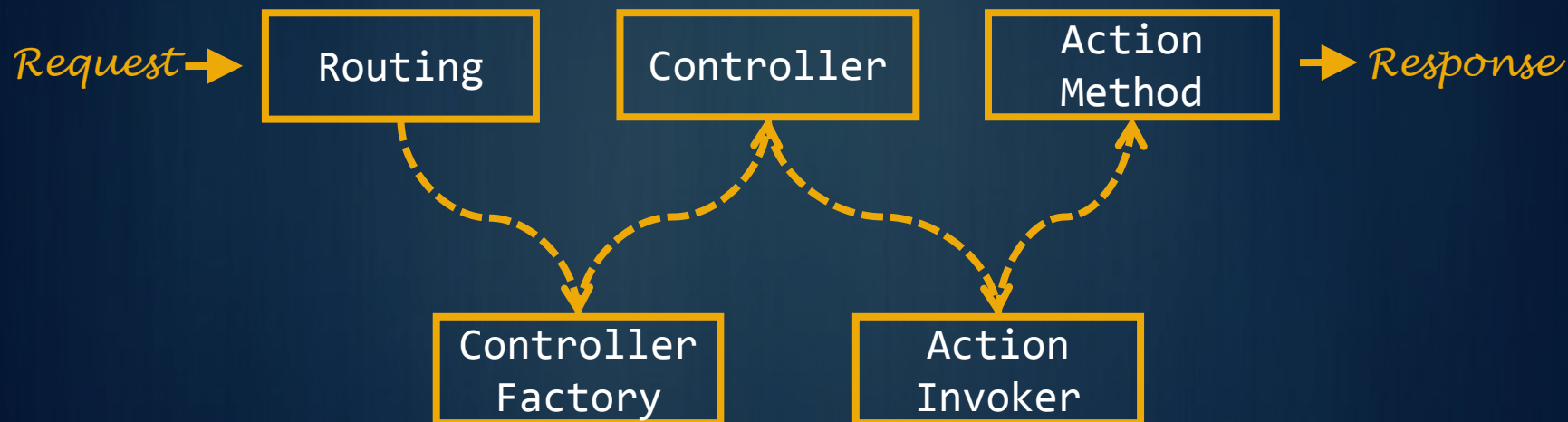
Часто используемые контекстные объекты

Свойство	Тип	Описание
Request.QueryString	NameValueCollection	Переменные GET, отправленные с этим запросом
Request.Form	NameValueCollection	Переменные POST, отправленные с этим запросом
Request.Cookies	HttpCookieCollection	Куки, отправленные браузером с этим запросом
Request.HttpMethod	string	HTTP метод (например, GET или POST), используемый для этого запроса
Request.Headers	NameValueCollection	Полный набор HTTP заголовков, отправленный с этим запросом
Request.Url	Uri	Запрашиваемый URL
Request.UserHostAddress	string	IP адрес пользователя, сделавшего запрос

Часто используемые контекстные объекты

Свойство	Тип	Описание
RouteData.Route	RouteBase	Выбранная запись из RouteTable.Routes для этого запроса
RouteData.Values	RouteValueDictionary	Активные роутовые параметры (как полученные из URL, так и значения по умолчанию)
HttpContext.Application	HttpApplicationStateBase	Состояние приложения
HttpContext.Cache	Cache	Кэш приложения
HttpContext.Items	IDictionary	Состояние текущего запроса
HttpContext.Session	HttpSessionStateBase	Состояние сессии пользователя
User	IPrincipal	Информация об аутентификации залогиненного пользователя

Компоненты конвейера обработки запроса



Создание контроллера на основе класса Controller

С классом **Controller** связаны следующие ключевые возможности:

- **Action methods** (методы действия): поведение контроллера разделено на множество методов (вместо того, чтобы иметь только один метод `Execute`). Каждый метод действия срабатывает для определенного, «своего» URL и вызывается с параметрами, извлеченными из входящего запроса.
- **Action results** (результаты действия): можно вернуть объект, описывая результат действия (например, отображение представления или перенаправление на другой URL или метод действия), который затем можно использовать по своему усмотрению. Разделение между указанием результатов и их выполнением упрощает модульное тестирование.
- **Filters** (фильтры): можно инкапсулировать повторяющиеся виды поведения (например, аутентификацию) в качестве фильтров, а затем добавлять каждый вид поведения в один или несколько контроллеров или методов действия, разместив **[Attribute]** в исходном коде.

Основы контроллеров

Guestbook Controller class

Base Controller class

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

Action method

*View() defined in
base Controller class*

Return type

Требования для action-метода

Чтобы рассматриваться в качестве действия, **action-метод** должен удовлетворять следующим требованиям:

- **Не может** быть статическим
- **Не может** быть методом расширения
- **Не может** быть конструктором или свойством
- **Не может** иметь открытый параметризованный тип
- **Не может** быть методом базового класса **Controller**
- **Не может** быть методом базового класса **ControllerBase**
- **Не может** содержать параметры **ref** или **out**
- **Не может** быть помечен атрибутом **NonAction**

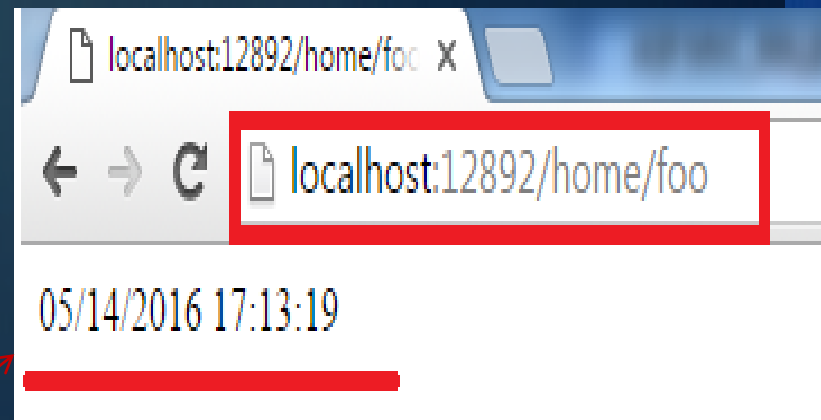


Требования для action-метода

```
namespace Controllers.Controllers
{
    References
    public class HomeController : Controller
    {
        //
        // GET: /Home/

        References
        public ActionResult Index()
        {
            return View();
        }

        References
        public DateTime Foo()
        {
            return DateTime.Now;
        }
    }
}
```

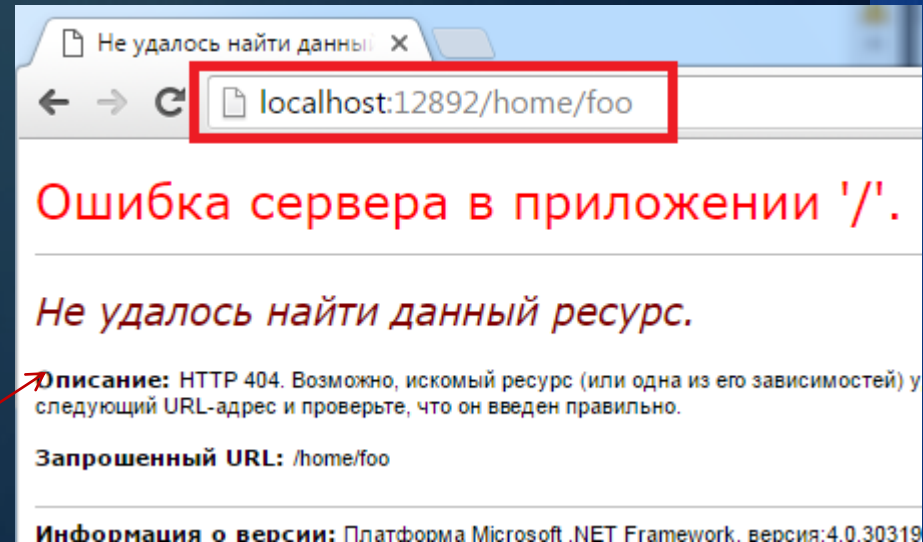


Требования для action-метода

```
namespace Controllers.Controllers
{
    // references
    public class HomeController : Controller
    {
        //
        // GET: /Home/

        // references
        public ActionResult Index()
        {
            return View();
        }

        [NonAction]
        // references
        public DateTime Foo()
        {
            return DateTime.Now;
        }
    }
}
```



Передача данных в контроллер

Передача данных в контроллер осуществляется тремя способами:

1. С помощью **формы или Javascript кода**, все данные полученные от клиента извлекаются с помощью `Request.Form [ключ]`
2. С помощью **переменных сегмента в url**, связанные *с системой маршрутизации*. Данные извлекаются с помощью свойстве `RouteData` и коллекции `Values` по ключу - имя переменной сегмента
3. С помощью **строки запроса QueryString**, который тоже передается через адресную строку, после адреса сервера обработки установить знак вопроса и далее добавить неограниченное количество параметров, которые идут по шаблону `имя параметра равно значению` и если параметров несколько, то мы их разделяем `&` . Данные извлекаются через свойства `Request` и берем коллекцию `QueryString` по имени параметра `message` из адресной строки.

Передача данных в контроллер

```
namespace _04_DataFromRequest
```

```
{
```

```
1 reference
```

```
public class RouteConfig
```

```
{
```

```
1 reference
```

```
public static void RegisterRoutes(RouteCollection routes)
```

```
{
```

```
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
```

```
    routes.MapRoute(  
        name: "Default",  
        url: "{controller}/{action}/{id}",  
        defaults: new { controller = "Home",  
                        action = "Index",  
                        id = UrlParameter.Optional }  
    );  
}
```

```
    name: "Default",
```

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

```
    defaults: new { controller = "Home",
```

```
                    action = "Index",
```

```
                    id = UrlParameter.Optional }
```

```
    );
```

```
}
```

```
}
```

```
}
```

04_DataFromRequest

Properties

References

App_Data

App_Start

FilterConfig.cs

RouteConfig.cs

WebApiConfig.cs

Controllers

DataController.cs

HomeController.cs

Models

Views

Data

Index.cshtml

Home

Index.cshtml

Web.config

Global.asax

packages.config

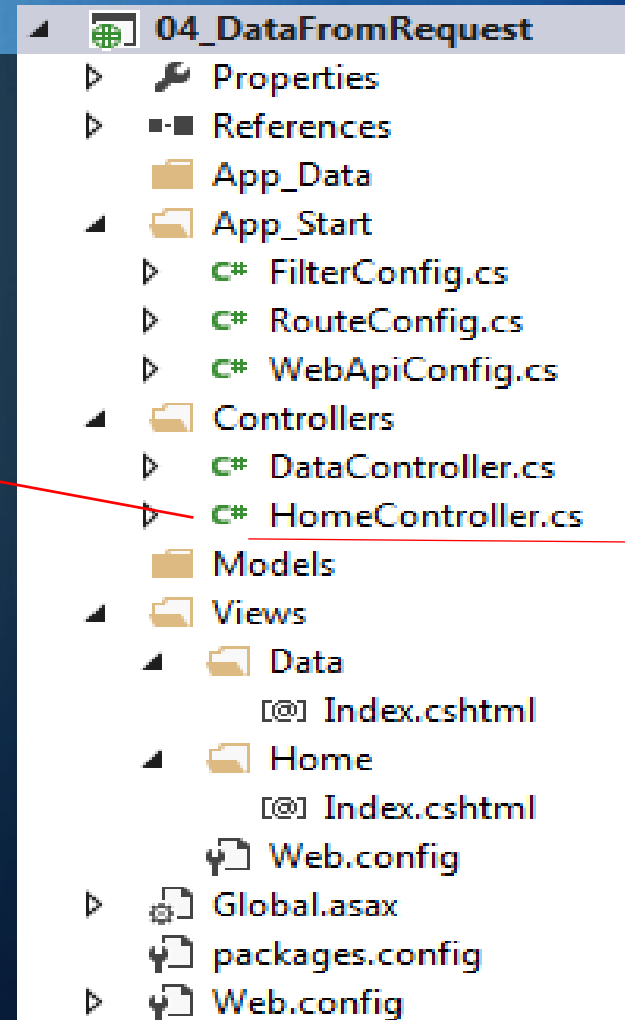
Web.config



Передача данных в контроллер

```
namespace _04_DataFromRequest.Controllers
{
    0 references
    public class HomeController : Controller
    {
        //
        // GET: /Home/

        0 references
        public ActionResult Index()
        {
            return View();
        }
    }
}
```



Передача данных в контроллер

Index.cshtml

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
        <!--Данные отправляются с помощью POST запроса-->
        <form action="/Data/PostInformation" method="post">
            <input type="text" name="message" />
            <input type="submit" value="Отправить на сервер" />
        </form>

        <!--Данные отправляются с помощью GET запроса, как данные в маршруте-->
        <a href="/Data/RouteInformation/hello-world-routedata">Route Data</a>

        <br />
        <!--Данные отправляются с помощью GET запроса, как данные адресной строки-->
        <a href="/Data/QueryInformation?message=hello-world-querystring">QueryString Data</a>
    </div>
</body>
</html>
```


- 04_DataFromRequest
 - Properties
 - References
 - App_Data
 - App_Start
 - FilterConfig.cs
 - RouteConfig.cs
 - WebApiConfig.cs
 - Controllers
 - DataController.cs
 - HomeController.cs
 - Models
 - Views
 - Data
 - Index.cshtml
 - Home
 - Index.cshtml
 - Web.config



Передача данных в контроллер

Передача данных с помощью *формы* или Javascript кода, все данные полученные от клиента извлекаются с помощью Request.Form [ключ]

```
<!--Данные отправляются с помощью POST запроса-->  
<form action="/Data/PostInformation" method="post">  
  <input type="text" name="message" />  
  <input type="submit" value="Отправить на сервер" />  
</form>
```




```
// POST: /Data/DataInformation/  
public ActionResult PostInformation()  
{  
  // Чтение данных, которые передаются с помощью POST запроса.  
  ViewBag.Message = Request.Form["message"];  
  return View("Index");  
}
```



Передача данных в контроллер

Передача данных с помощью *формы* или Javascript кода, все данные полученные от клиента извлекаются с помощью Request.Form [ключ]

```
<!--Данные отправляются с помощью POST запроса-->  
<form action="/Data/PostInformation" method="post">  
  <input type="text" name="message" />  
  <input type="submit" value="Отправить на сервер" />  
</form>
```



```
// POST: /Data/DataInformation/  
public ActionResult PostInformation()  
{  
  // Чтение данных, которые передаются с помощью POST запроса.  
  ViewBag.Message = Request.Form["message"];  
  return View("Index");  
}
```



Передача данных в контроллер

Передача данных с помощью **переменных сегмента в url**, связанные с *системой маршрутизации*. Данные извлекаются с помощью свойства `RouteData` и коллекции `Values` по ключу - имя переменной сегмента

```
<!--Данные отправляются с помощью GET запроса, как данные в маршруте-->  
<a href="/Data/RouteInformation/hello-world-routedata">Route Data</a>
```

```
public ActionResult RouteInformation()  
{  
    // Чтение данных, которые передаются  
    //с помощью GET запроса, как данные в маршруте.  
    ViewBag.Message = RouteData.Values["id"];  
    return View("Index");  
}
```

```
routes.MapRoute(  
    name: "Default",  
    url: "{controller}/{action}/{id}",  
    defaults: new { controller = "Home",  
                    action = "Index",  
                    id = UrlParameter.Optional }  
);
```

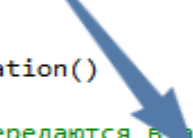


Передача данных в контроллер

Передача данных с помощью **строки запроса QueryString**, который передается через адресную строку, после адреса сервера обработки установить знак вопроса и далее добавить неограниченное количество параметров, которые идут по шаблону имя параметра равно значению и если параметров несколько, то мы их разделяем & . Данные извлекаются через свойства Request и берем коллекцию QueryString по имени параметра message из адресной строки.

```
<!--Данные отправляются с помощью GET запроса, как данные адресной строки-->  
...<a href="/Data/QueryInformation?message=hello-world-querystring">QueryString Data</a>
```

```
// GET: /Data/QueryInformation/  
public ActionResult QueryInformation()  
{  
    // Чтение данных, которые передаются в адресной строке.  
    ViewBag.Message = Request.QueryString["message"];  
    return View("Index");  
}
```



Передача данных в контроллер

The screenshot shows a Visual Studio IDE with the following components:

- Main Editor (Index.cshtml):**

```
@{  
    Layout = null;  
}  
  
<!DOCTYPE html>  
  
<html>  
<head>  
    <meta name="viewport" content="width=device-width"/>  
    <title>DataController</title>  
</head>  
<body>  
    <div>  
        Message: @ViewBag.Message  
    </div>  
</body>  
</html>
```
- Right-Hand Pane (Solution Explorer):**
 - 04_DataFromRequest
 - Properties
 - References
 - App_Data
 - App_Start
 - FilterConfig.cs
 - RouteConfig.cs
 - WebApiConfig.cs
 - Controllers
 - DataController.cs
 - HomeController.cs
 - Models
 - Views
 - Data
 - [@] Index.cshtml
 - Home
 - [@] Index.cshtml



Передача данных в контроллер

Index.cshtml

@{
Layout = null;
}

<!DOCTYPE html>

<html>
<head>
 <meta name="viewport"
 <title>Index</title>
</head>
<body>
 <div>
 <!--Данные отправляются с помощью POST запроса-->
 <form action="/Data/PostInformation" method="post">
 <input type="text" name="message" />
 <input type="submit" value="Отправить на сервер" />
 </form>

 <!--Данные отправляются с помощью GET запроса, как данные в маршруте-->
 Route Data

 <!--Данные отправляются с помощью GET запроса, как данные адресной строки-->
 QueryString Data
 </div>
</body>

Index

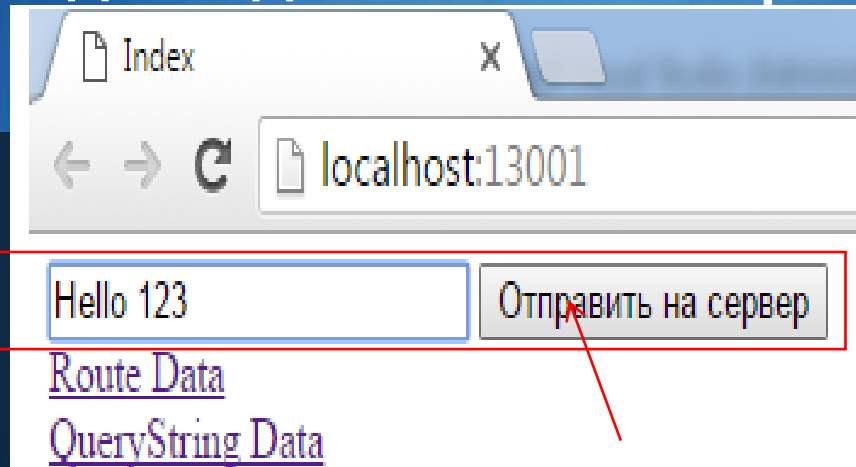
localhost:13001

Отправить на сервер

Route Data

QueryString Data

Передача данных в контроллер



Index x

localhost:13001

Hello 123

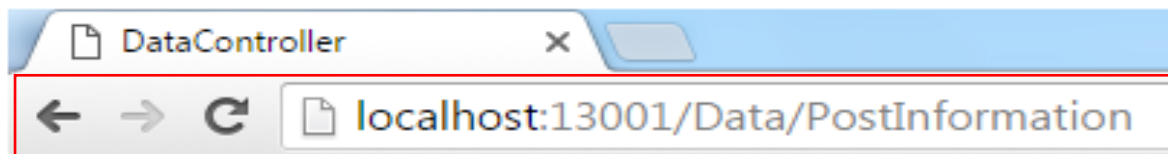
Отправить на сервер

[Route Data](#)

[QueryString Data](#)

```
<!--Данные отправляются с помощью POST запроса-->  
<form action="/Data/PostInformation" method="post">  
  <input type="text" name="message" />  
  <input type="submit" value="Отправить на сервер" />  
</form>
```

```
// POST: /Data/DataInformation/  
public ActionResult PostInformation()  
{  
  // Чтение данных, которые передаются с помощью POST запроса.  
  ViewBag.Message = Request.Form["message"];  
  return View("Index");  
}
```



DataController x

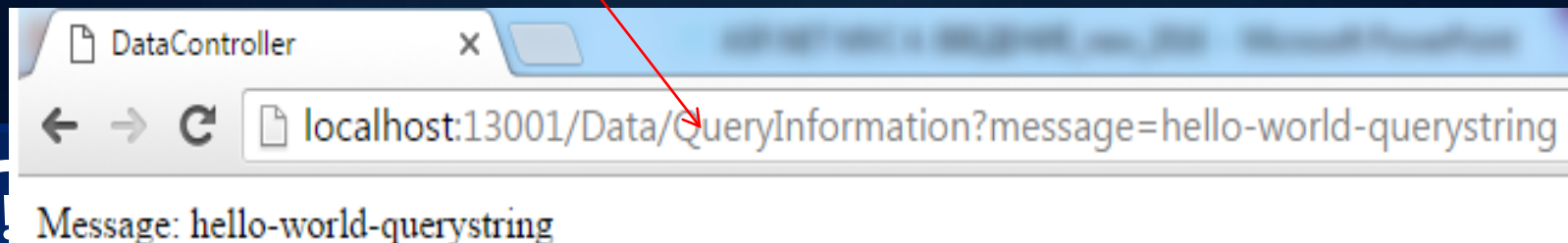
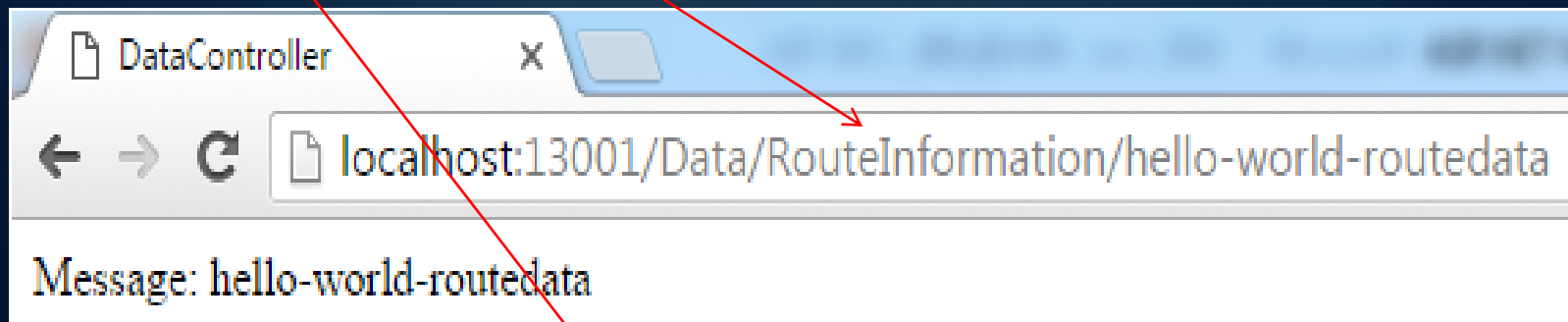
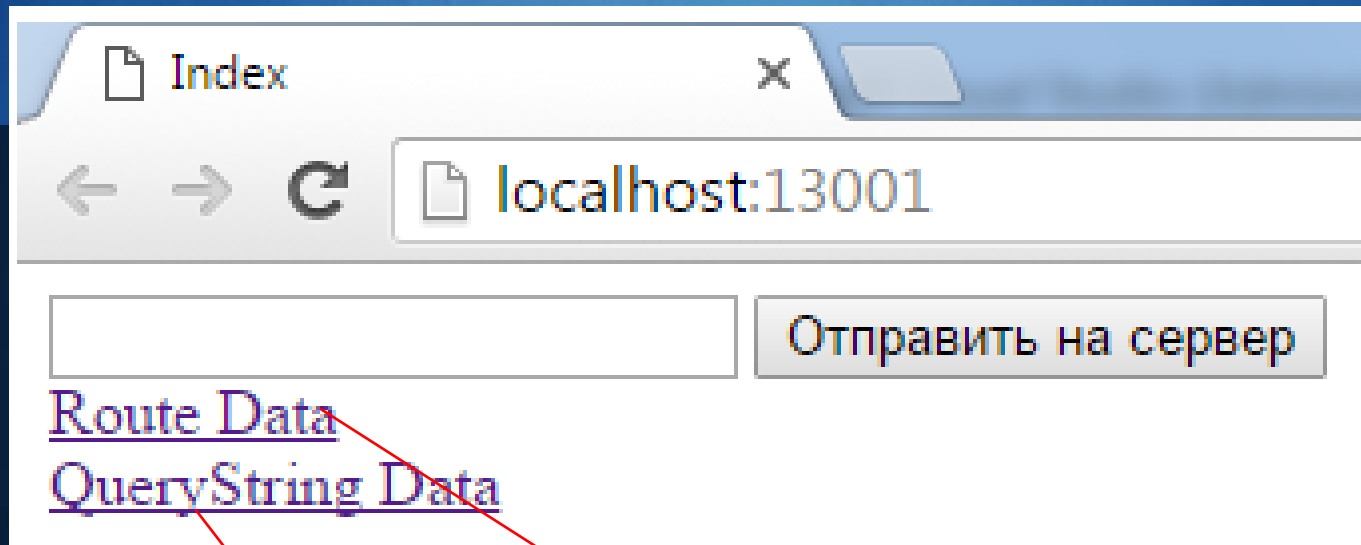
localhost:13001/Data/PostInformation

Message: Hello 123



Example 4

Передача данных в контроллер



Получение данных из набора контекстных объектов

Методам контроллеров, как правило, необходим доступ к входящим данным, таким как значения строки запроса, значения форм или параметры, полученные системой маршрутизации из входящего URL. Есть три основных способа доступа к этим данным:

- **Получить данные из набора контекстных объектов**

`Request.QueryString["key"]` значения из строки запроса

`RouteData.Values["key"]` параметры, полученные системой маршрутизации

`Request.Form["key"]` значения форм

- **Получить данные, переданные в качестве параметров методу действия**
- **Явно вызвать связывание данных модели**

`UpdateModel(model);`

`TryUpdateModel(model);`

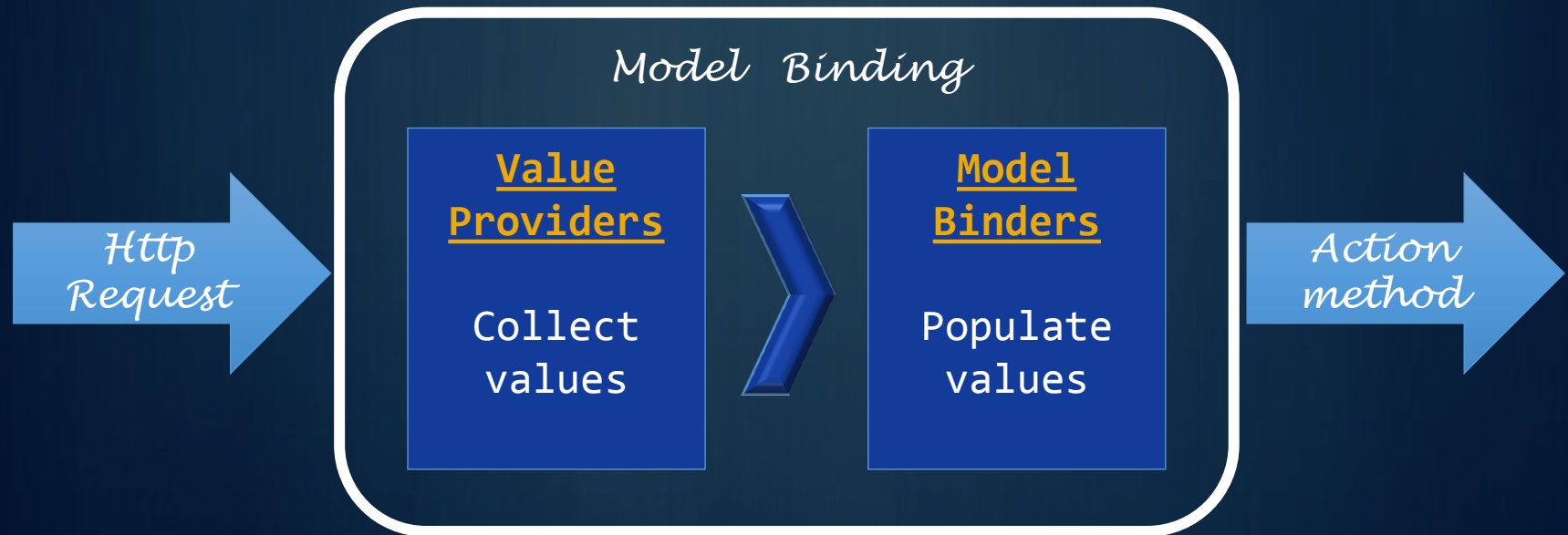
`TryUpdateModel(model);`

Связывание данных модели и провайдеры значений

Action invoker (активатор действия) - компонент, который вызывает action-методы. Перед тем, как вызвать action-метод, необходимо заполнить значениями параметры метода. **ControllerActionInvoker** (активатор действия по умолчанию) для заполнения параметров использует привязчики модели (model binder), которые осуществляют привязку модели. Все привязчики модели реализуют интерфейс **IModelBinder**.

```
public interface IModelBinder
{
    object BindModel (ControllerContext controllerContext,
                     ModelBindingContext bindingContext);
}
```

Связывание данных модели и провайдеры значений



Базовый класс **Controller** получает значения для параметров метода действия с помощью MVC компонентов, называемых **провайдерами значений** и **механизмами связывания данных модели**

Связывание данных модели и провайдеры значений

Базовый класс **Controller** получает значения для параметров метода действия с помощью MVC компонентов, называемых **провайдерами значений** и **механизмами связывания данных модели**

Встроенные провайдеры значений получают элементы из **Request.Form**, **Request.QueryString**, **Request.Files** и **RouteData.Values** и передают эти значения механизмам связывания данных, которые пытаются привязать их к типам, которые методы действий требуют в качестве параметров

Связывание данных модели и провайдеры значений

Поставщик данных – это класс, который выполняет поиск значения в определенной части входящего запроса

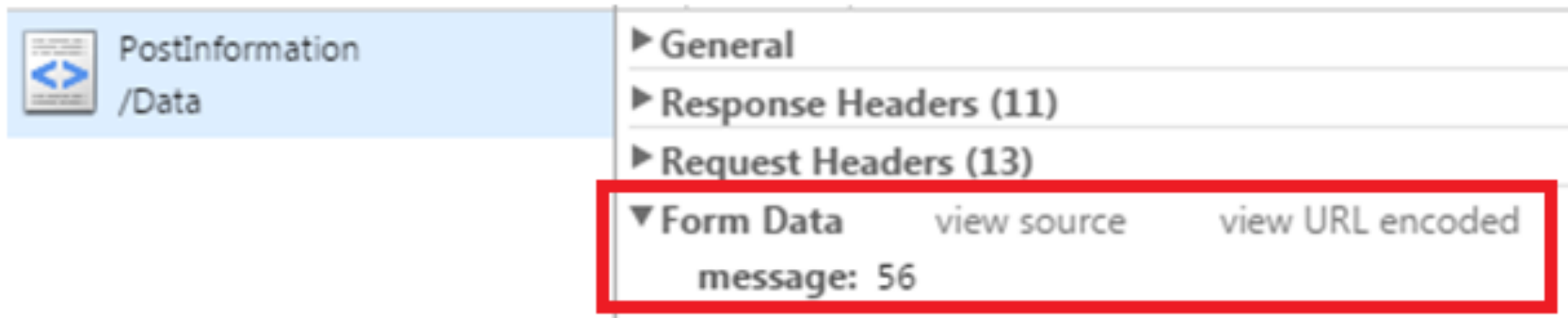
- **FormValueProvider** – значения, переданные в HTML элементах form
- **RouteDataValueProvider** – значения из маршрутов приложения
- **QueryStringValueProvider** – данные включенные в строку запроса
- **HttpFileCollectionValueProvider** – файлы загруженные как часть запроса

```
public interface IModelBinderProvider
{
    IModelBinder GetBinder(Type modelType);
}
```

Связывание данных модели и провайдеры значений

Правила наименования

```
<!-- Данные отправляются с помощью POST запроса -->
<form action="/Data/PostInformation" method="get">
  <input type="number" name="message" />
  <input type="submit" value="Отправить на сервер" />
</form>
public ActionResult PostInformation(int message)
{
    ViewBag.Message = message;
    return View("Index");
}
```



The screenshot shows the Visual Studio IDE. On the left, the Solution Explorer displays the 'PostInformation' controller under the '/Data' namespace. On the right, the 'Server Explorer' pane shows the 'Form Data' tab selected, which displays the form data submitted by the browser. The data is shown as 'message: 56'. The 'Form Data' tab is highlighted with a red rectangle.

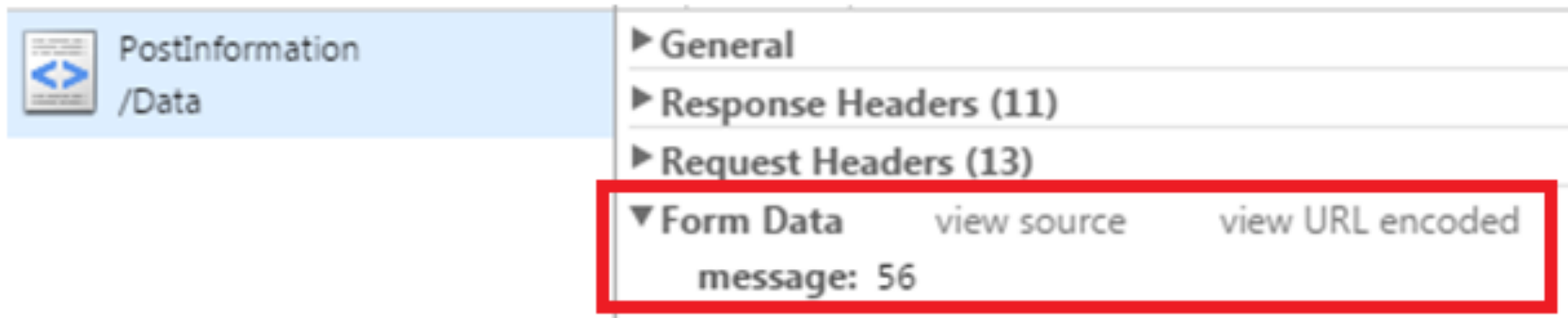
Form Data	view source	view URL encoded
message: 56		



Связывание данных модели и провайдеры значений

Правила наименования

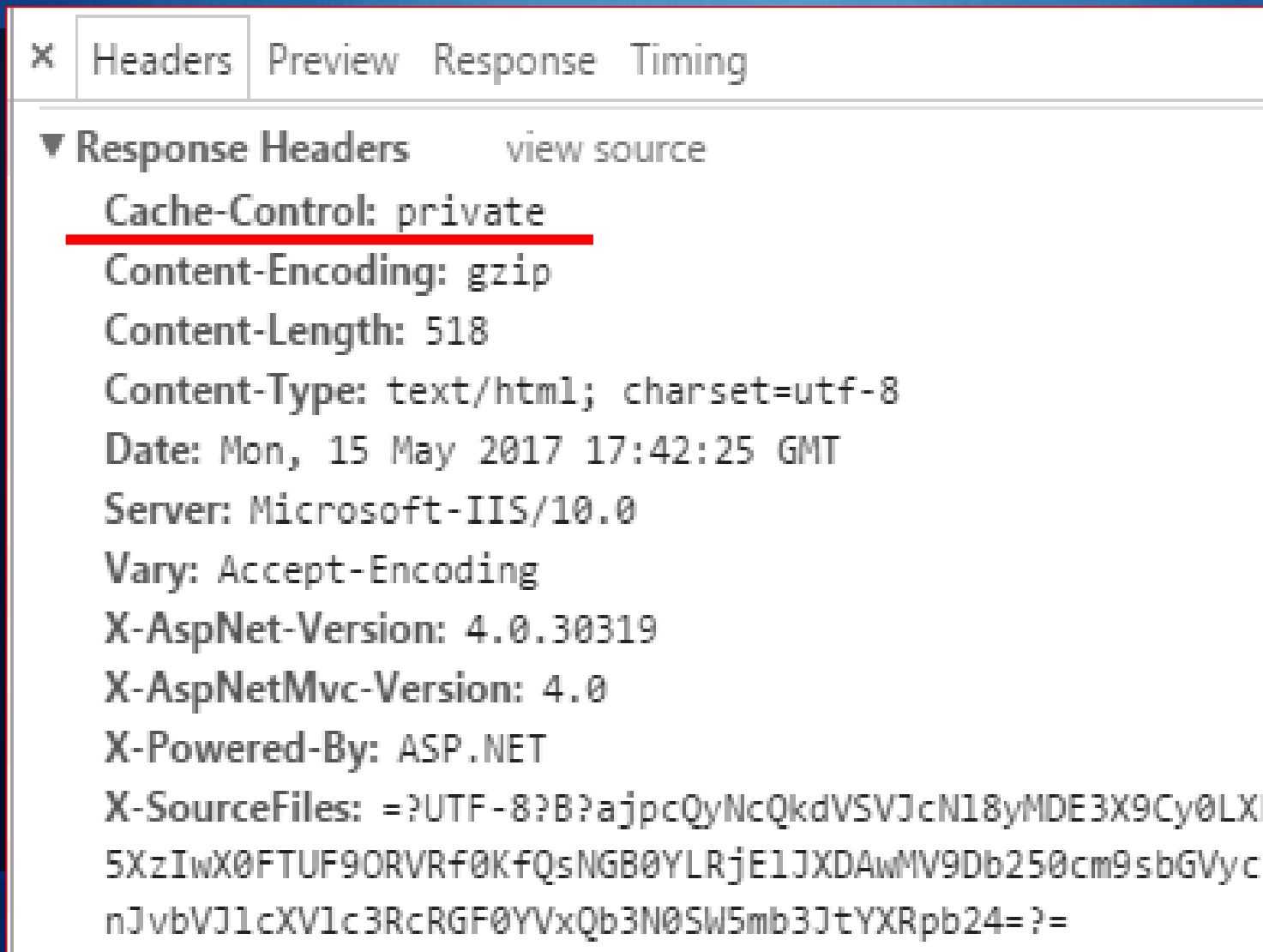
```
<!-- Данные отправляются с помощью POST запроса -->
<form action="/Data/PostInformation" method="get">
  <input type="number" name="message" />
  <input type="submit" value="Отправить на сервер" />
</form>
public ActionResult PostInformation(int message)
{
    ViewBag.Message = message;
    return View("Index");
}
```



The screenshot shows the Visual Studio IDE. On the left, a file explorer pane displays 'PostInformation /Data'. On the right, a 'Server Explorer' pane is open, showing the 'PostInformation' view. The 'Form Data' section is expanded and highlighted with a red rectangle, displaying the value 'message: 56'. The 'Request Headers' section is also visible, showing 13 headers.



Связывание данных модели и провайдеры значений

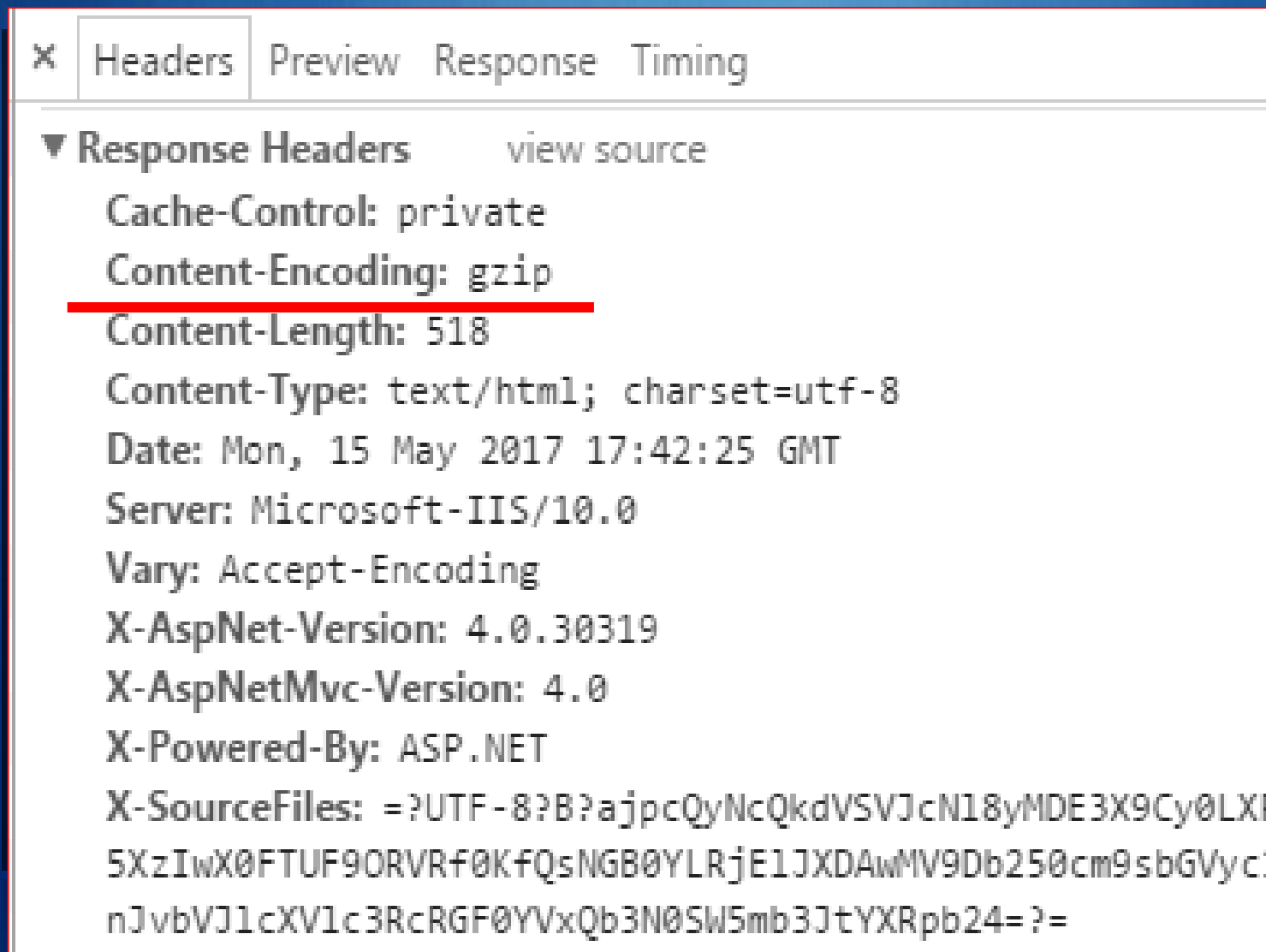


X Headers Preview Response Timing

▼ Response Headers [view source](#)

- Cache-Control: private
- Content-Encoding: gzip
- Content-Length: 518
- Content-Type: text/html; charset=utf-8
- Date: Mon, 15 May 2017 17:42:25 GMT
- Server: Microsoft-IIS/10.0
- Vary: Accept-Encoding
- X-AspNet-Version: 4.0.30319
- X-AspNetMvc-Version: 4.0
- X-Powered-By: ASP.NET
- X-SourceFiles: =?UTF-8?B?ajpcQyNcQkdVSVJcN18yMDE3X9Cy0LXF5XzIwX0FTUF9ORVRf0KfQsNGB0YLRjE1JXDAMV9Db250cm9sbGVyc1nJvbVJlcXVlc3RcRGF0YVxQb3N0SW5mb3JtYXRpb24=?=

Связывание данных модели и провайдеры значений



X Headers Preview Response Timing

▼ Response Headers [view source](#)

- Cache-Control: private
- Content-Encoding: gzip
- Content-Length: 518
- Content-Type: text/html; charset=utf-8
- Date: Mon, 15 May 2017 17:42:25 GMT
- Server: Microsoft-IIS/10.0
- Vary: Accept-Encoding
- X-AspNet-Version: 4.0.30319
- X-AspNetMvc-Version: 4.0
- X-Powered-By: ASP.NET
- X-SourceFiles: =?UTF-8?B?ajpcQyNcQkdVSVJcN18yMDE3X9Cy0LXF5XzIwX0FTUF9ORVRf0KfQsNGB0YLRjE1JXDAMV9Db250cm9sbGVyc1nJvbVJlcXVlc3RcRGF0YVxQb3N0SW5mb3JtYXRpb24=?=

Связывание данных модели и провайдеры значений

▼ Request Headers [view source](#)

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

Accept-Encoding: gzip, deflate, br

Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.6,en;q=0.4

Cache-Control: max-age=0

Connection: keep-alive

Content-Length: 11

Content-Type: application/x-www-form-urlencoded

Host: localhost:13061

Origin: http://localhost:13061

Referer: http://localhost:13061/

Upgrade-Insecure-Requests: 1

Связывание данных модели и провайдеры значений

Правила наименования

```
routes.MapRoute(  
    name: "Default",  
    url: "{controller}/{action}/{id}",  
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }  
);  
  
public ActionResult RouteInformation(string id)  
{  
    ViewBag.Message = id;  
    return View("Index");  
}
```

Связывание данных модели и провайдеры значений

Правила наименования

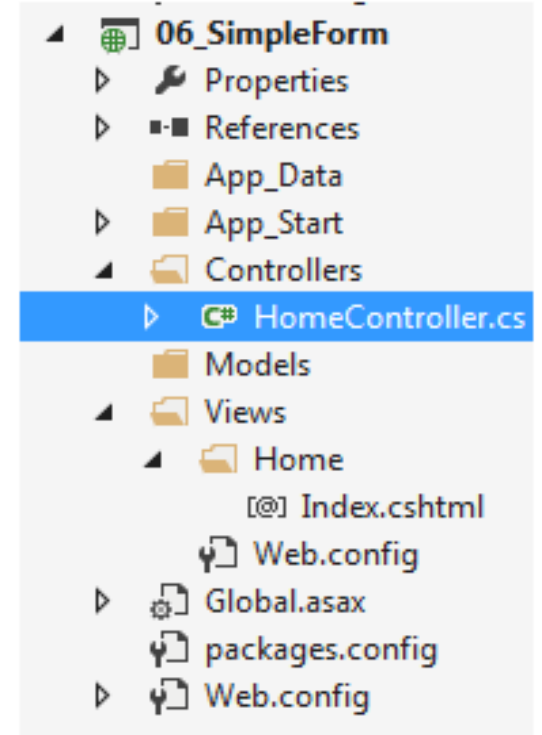
```
<a href="/Data/QueryInformation?message=hello-world-querysting">  
  
public ActionResult QueryInformation(string message)  
{  
    ViewBag.Message = message;  
    return View("Index");  
}
```

Связывание данных модели и провайдеры значений

```
namespace _06_SimpleForm.Controllers
{
    ссылка 0
    public class HomeController : Controller
    {
        ссылка 0
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        ссылка 0
        public ActionResult Index(int x, int y)
        {
            ViewBag.Result = x + y;

            return View();
        }
    }
}
```

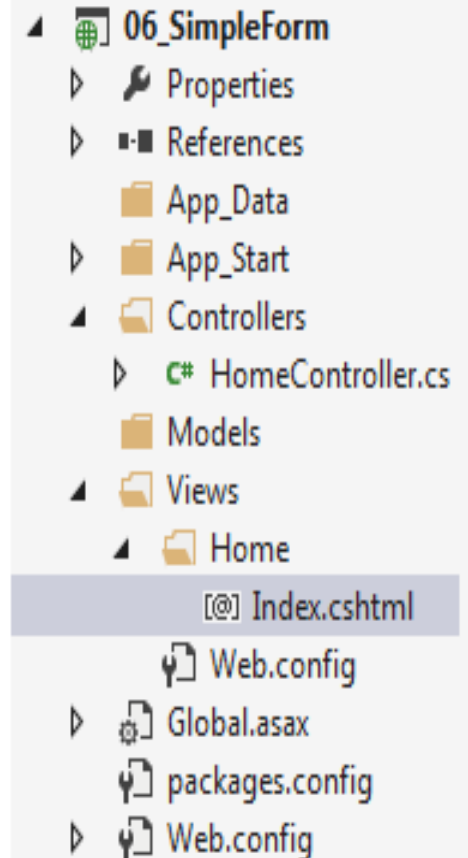


Связывание данных модели и провайдеры значений

```
<body>
<div>
  <form action="/Home/Index" method="post">
    <label>X</label> <input type="text" name="x" /> <br />
    <label>Y</label> <input type="text" name="y" /> <br />

    @if (ViewBag.Result != null)
    {
      <p>@ViewBag.Result</p>
    }

    <input type="submit" name="operation" value="Get Result" />
  </form>
</div>
</body>
```



Связывание данных модели и провайдеры значений

```
namespace _06_SimpleForm.Controllers
{
    ссылка 0
    public class HomeController : Controller
    {
        [HttpPost]
        ссылка 0
        public ActionResult Index(int x, int y)
        {
            ViewBag.Result = x + y;

            return View();
        }
    }
}
```

Index

localhost:15091

X 7

Y 8

Get Result

```
<form action="/Home/Index" method="post">
    <label>X</label> <input type="text" name="x" /> <br />
    <label>Y</label> <input type="text" name="y" /> <br />

    @if (ViewBag.Result != null)
    {
        <p>@ViewBag.Result</p>
    }

    <input type="submit" name="operation" value="Get Result" />
</form>
```



Связывание данных модели и провайдеры значений

```
namespace _06_SimpleForm.Controllers
{
    ссылка 0
    public class HomeController : Controller
    {
        [HttpPost]
        ссылка 0
        public ActionResult Index(int x, int y)
        {
            ViewBag.Result = x + y;

            return View();
        }
    }
}
```

Index

localhost:15091/Home/Index

X

Y

15

Get Result

```
<form action="/Home/Index" method="post">
    <label>X</label> <input type="text" name="x" /> <br />
    <label>Y</label> <input type="text" name="y" /> <br />

    @if (ViewBag.Result != null)
    {
        <p>@ViewBag.Result</p>
    }

    <input type="submit" name="operation" value="Get Result" />
</form>
```



Связывание данных модели и провайдеры значений

```
namespace _06_SimpleForm.Controllers
{
    ссылка 0
    public class HomeController : Controller
    {
        [HttpPost]
        ссылка 0
        public ActionResult Index(int x, int y)
        {
            ViewBag.Result = x + y;

            return View();
        }
    }
}
```

Index

localhost:15091

X

Y 8

Get Result

```
<form action="/Home/Index" method="post">
    <label>X</label> <input type="text" name="x" /> <br />
    <label>Y</label> <input type="text" name="y" /> <br />

    @if (ViewBag.Result != null)
    {
        <p>@ViewBag.Result</p>
    }

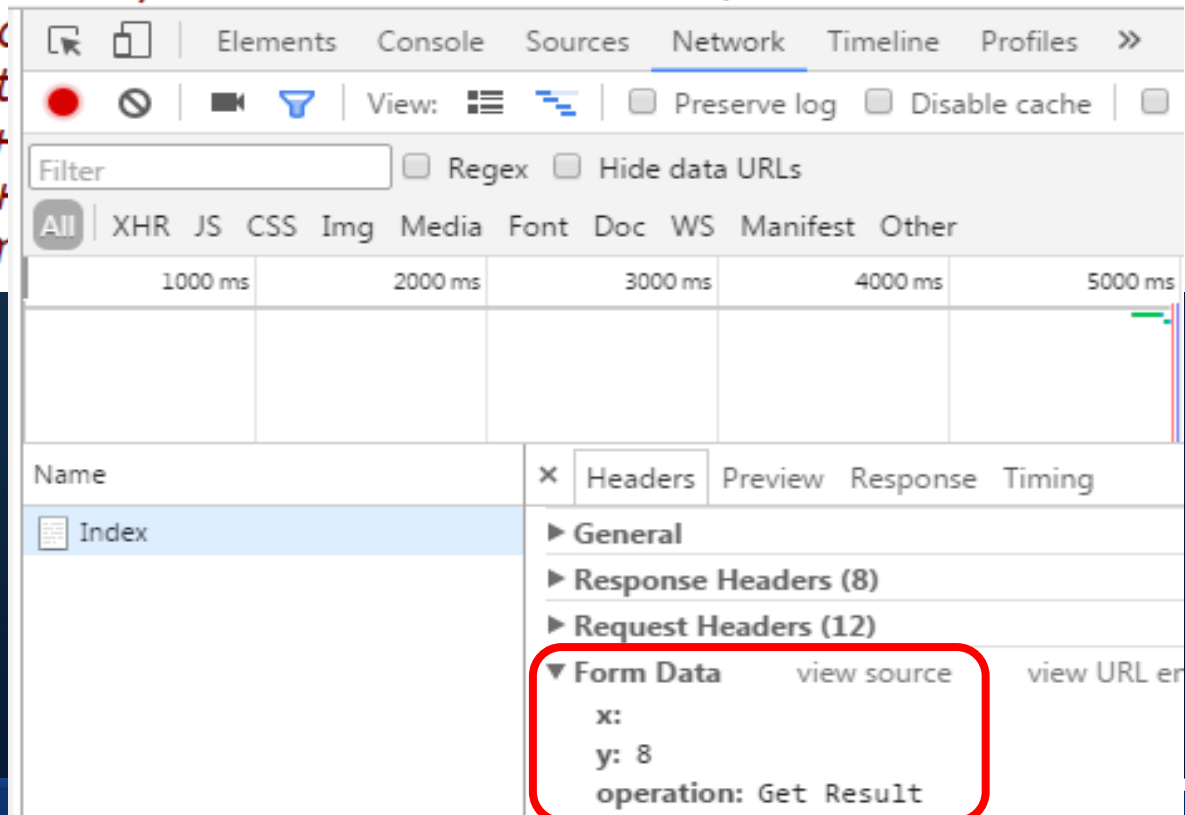
    <input type="submit" name="operation" value="Get Result" />
</form>
```



Связывание данных модели и провайдеры значений

Ошибка сервера в приложении '/'.

Словарь параметров содержит запись со значением *NULL* для параметра "x" типа "System.Int32", не допускающего значение *NULL*, для метода "System.Web.Mvc.Action_06_SimpleForm.Controller" должен иметь ссылку на ресурс, который должен быть объявлен в URL. Имя параметра: param



Example 6

Связывание данных модели и провайдеры значений

[HttpPost]

ссылка 0

```
public ActionResult Index(int? x, int? y)
{
    ViewBag.Result = x + y;

    return View();
}
```

Index

localhost:15091/Home/Index

X

Y

Get Result

[HttpPost]

ссылка 0

```
public ActionResult Index(int x = 5, int y = 5)
{
    ViewBag.Result = x + y;

    return View();
}
```

Index

localhost:15091/Home/Index

X

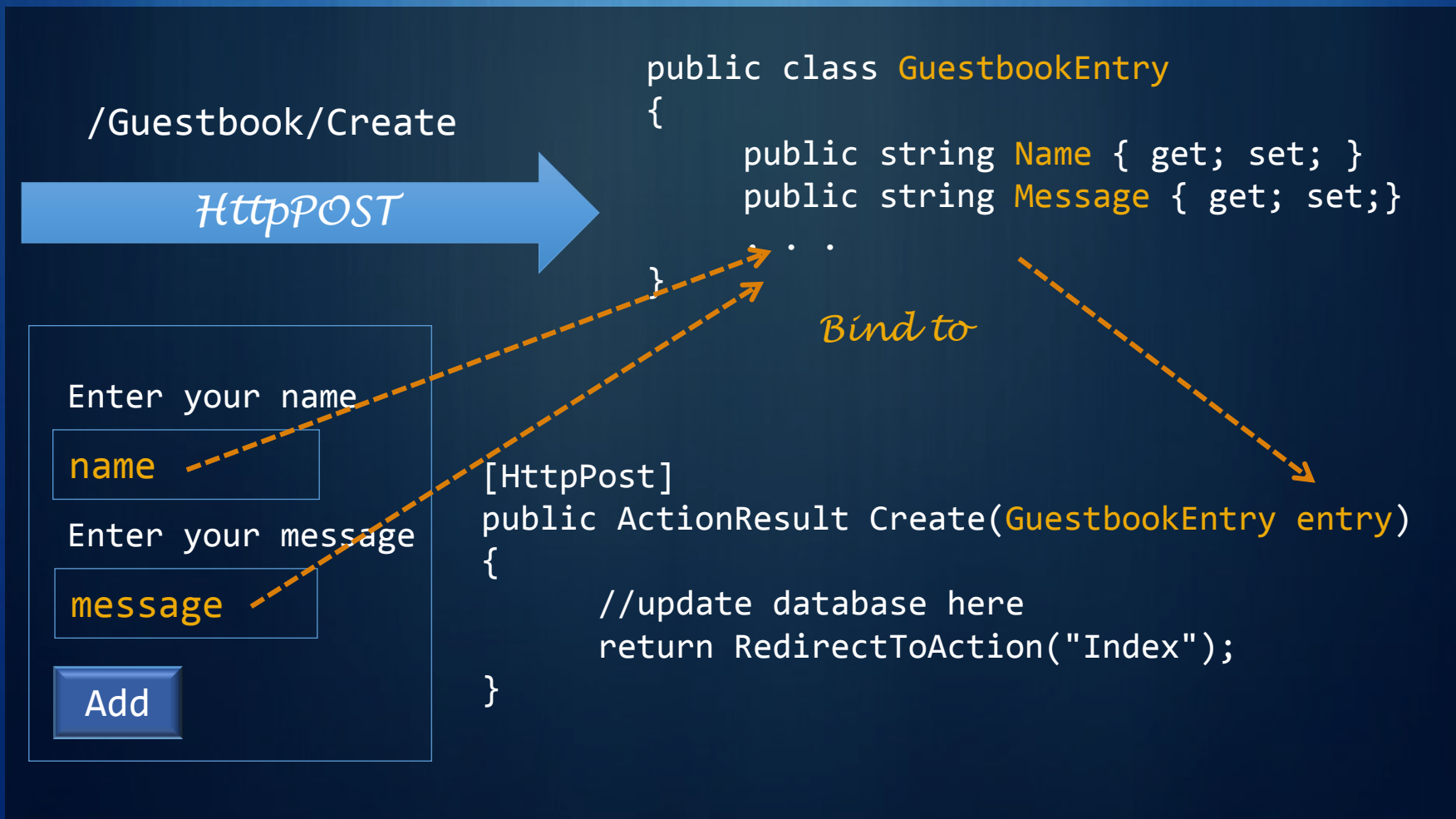
Y

10

Get Result



Связывание данных модели и провайдеры значений



Связывание данных модели и провайдеры значений

References

```
public ActionResult Create(GuestBookEntity guestBook)
{
    guestBook.DateAdd = DateTime.Now;

    ctx.Entries.Add(guestBook);

    ctx.SaveChanges();

    return View();
}
```


Связывание данных модели и провайдеры значений

Введите имя

Введите комментарий

Hello ASP.NET MVC

Add

Ожидание localhost...		
Elements Console Sources Network Timeline Profiles		
View: [List Icon] [Tree Icon] <input type="checkbox"/> Preserve log <input checked="" type="checkbox"/> Disable cache		
2000 ms 4000 ms 6000 ms		
Name		Method
Path		
/ata39a586/ed40e382dc4f56e0/b4234/artery/signalk		
 <u>Create</u>		POST
<u>/GuestBook</u>		

Связывание данных модели и провайдеры значений

The screenshot shows the Chrome DevTools Network tab. The left sidebar lists several network requests, with the 'Create' request to '/GuestBook' selected. The main panel shows the 'Headers' tab, which is highlighted with a red box. Under the 'Request Headers' section, a warning icon indicates 'Provisional headers are shown'. The headers listed are: Accept: text/html,application/xhtml+xml,application/xml;q=0.9, Content-Type: application/x-www-form-urlencoded, Origin: http://localhost:3023, Referer: http://localhost:3023/GuestBook/Create, Upgrade-Insecure-Requests: 1, User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (X-DevTools-Emulate-Network-Conditions-Client-Id: 31614B62-00E1-4), and X-DevTools-Emulate-Network-Conditions-Client-Id: 31614B62-00E1-4. The 'Form Data' section is expanded and highlighted with a red box, showing two fields: 'Name: Guest77' and 'Message: Hello ASP.NET MVC'.

Name	Path	Headers	Preview	Response	Timing
	poll?transport=longPolling... /afa39a5867ed40e382dc4f...				
	Create /GuestBook				
	poll?transport=longPolling... /afa39a5867ed40e382dc4f...				
	poll?transport=longPolling... /afa39a5867ed40e382dc4f...				
	poll?transport=longPolling... /afa39a5867ed40e382dc4f...				
	poll?transport=longPolling... /afa39a5867ed40e382dc4f...				
	poll?transport=longPolling... /afa39a5867ed40e382dc4f...				
	poll?transport=longPolling... /afa39a5867ed40e382dc4f...				

General
Request URL: http://localhost:3023/GuestBook/Create

Request Headers
⚠ Provisional headers are shown
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
Content-Type: application/x-www-form-urlencoded
Origin: http://localhost:3023
Referer: http://localhost:3023/GuestBook/Create
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (X-DevTools-Emulate-Network-Conditions-Client-Id: 31614B62-00E1-4
X-DevTools-Emulate-Network-Conditions-Client-Id: 31614B62-00E1-4

Form Data view source view URL encoded
Name: Guest77
Message: Hello ASP.NET MVC

Связывание данных модели и провайдеры значений

```
public ActionResult Create(GuestBookEntity guestBook)
```

```
{
```

```
    guestBook.DateAdd = DateTime.Now;
```

```
    ctx.Entries.Add(guestBook);
```

```
    ctx.SaveChanges();
```

```
    return View();
```

```
}
```

guestBook {GuestBook.Models.GuestBo	
DateAdd	{01.01.0001 0:00:00}
Id	0
Message	Q - "Hello ASP.NET MVC"
Name	Q - "Guest77"

Связывание данных модели и провайдеры значений

Применение явной привязки модели

[HttpPost]

0 references

public ActionResult SetData()

1_3_SetData

{

try

{

GuestBookEntity guestBook = new GuestBookEntity();
UpdateModel(guestBook); //TryUpdateModel(guestBook);

guestBook.DateAdd = DateTime.Now;

ctx.Entries.Add(guestBook);

ctx.SaveChanges();

}

catch

{

Debug.WriteLine("ERROR!");

}

return View("Create");

}

Связывание данных модели и провайдеры значений

Применение явной привязки модели

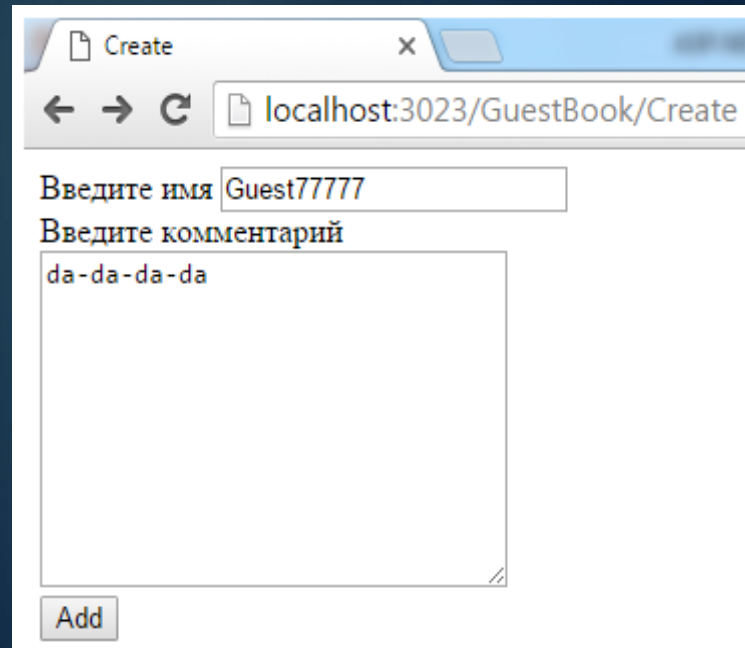
```
<form action="SetData" method="post">
  <div>
    <label>Введите имя </label>
    <input type="text" name="Name" id="Name" />
  </div>

  <div><label for="Message">Введите комментарий </label></div>
  <div>
    <textarea rows="10" cols="30" name="Message" id="Message"></textarea>
  </div>

  <div><input type="submit" value="Add" /></div>
</form>
```

Связывание данных модели и провайдеры значений

Применение явной привязки модели



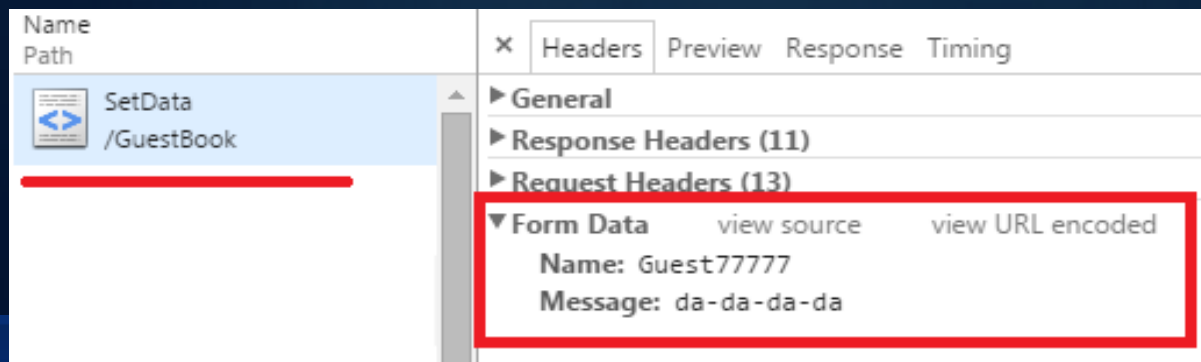
Create

localhost:3023/GuestBook/Create

Введите имя

Введите комментарий

Add



Name	Path
SetData	/GuestBook

Headers Preview Response Timing

- General
- Response Headers (11)
- Request Headers (13)
- Form Data** [view source](#) [view URL encoded](#)
 - Name: Guest77777
 - Message: da-da-da-da

Связывание данных модели и провайдеры значений

Применение явной привязки модели

Top Entries

Комментарий: da-da-da-da

Имя пользователя: Guest77777

Дата: 17.01.2016 19:20:59

Комментарий: da-da-da-da

Имя пользователя: Guest77777

Дата: 17.01.2016 19:10:37

Комментарий: GUEST-First

Имя пользователя: guest456

Дата: 17.01.2016 19:07:48

Комментарий: wwwwwwwwwwwwwwwwwwwww

Имя пользователя: guest55555

Дата: 17.01.2016 17:58:02

[Add new Entry](#)

Связывание данных модели и провайдеры значений

Для каждого отдельного параметра запроса может существовать свой привязчик модели. При просмотре параметров метода действия активатор действий ищет для каждого типа параметра соответствующий привязчик и вызывает его метод **BindModel**. В случае, если соответствующего данному типу привязчика не обнаружится, то используется привязчик по умолчанию - **DefaultModelBinder**.

Привязчик использует специальные компоненты - **поставщики значений (value provider)** для поиска значений в различных частях запроса.

Связывание данных модели и провайдеры значений

Привязчик `DefaultModelBinder` используется по умолчанию, если для данного типа не определен другой привязчик. Чтобы получить значения для параметров, привязчик просматривает следующие объекты строго по порядку:

- `Request.Form`. Значения, предоставленные пользователем в элементе HTML form
- `RouteData.Values`. Значения, полученные через маршруты приложения
- `Request.QueryString`. Данные строки запроса из URL
- `Request.Files`. Файлы, загруженные как часть запроса

Если данные не найдены, параметры ссылочного типа получают значение `null`, а для параметров типов значений генерируется исключение `InvalidOperationException`.

При конвертировании данных из словаря `Request.Form` применяются настройки языковой культуры сервера

Связывание данных модели и провайдеры значений

Если привязчик **DefaultModelBinder** не может найти значение параметра ссылочного типа, все равно будет вызван метод действия, но с использованием значения null для этого параметра. Если значение не может быть найдено для параметра значимого типа, то будет сгенерировано исключение, и метод действия вызван не будет :

- **Параметры значимого типа являются обязательными.** Чтобы сделать их опциональными, нужно либо указать значение по умолчанию, либо заменить тип параметра на **nullable** (например, int? или DateTime?). Таким образом, MVC сможет передать null, если значение будет не доступно.
- **Параметры ссылочного типа не являются обязательными.** Чтобы сделать их обязательными (чтобы убедиться, что передается значение не-null), нужно добавить код в начало метода действия, который не принимает значения null. Например, если значение равно null, выбрасывается исключение **ArgumentNullException**

Связывание данных модели и провайдеры значений

Когда параметр action-метода является сложным типом, тогда связыватель использует отражение (reflection), чтобы получить набор public свойств и затем по очереди привязать к каждому из них значение.

Чтобы помочь связывателю, можно использовать html helper-методы, например `Html.TextBoxFor(m => m.Title)`, или же у html элементов задавать атрибут `name`.

Для настройки привязки к модели можно использовать атрибут `[Bind]`, применяемый либо к параметру, либо к классу модели. При помощи `[Bind]` можно указать свойства объекта для применения или игнорирования привязки. Привязка к модели работает не только в случае скалярных объектов, но и для коллекций. Данные формы могут быть приняты действием с параметром `List<>`, `Dictionary<>`.

Создание выходных данных

Результаты действий в MVC Framework используются для разделения **заявлений о намерениях** и **выполнения намерений**

При использовании контроллеров, унаследованных от **System.Web.Mvc.Controller**, action-методы возвращают объект для описания результата своей работы. Как правило, используется класс ActionResult и его наследники. Метод-действие может возвращать произвольный объект или быть объявленным как void. В первом случае на основе результата создается объект класса ContentResult, во втором – возвращается объект EmptyResult.

Система action-результатов реализует шаблон проектирования **команда**. Когда MVC Framework получает объект типа ActionResult из action-метода, он вызывает у этого объекта метод ExecuteResult. Реализация ActionResult взаимодействует с объектом Response и генерирует необходимые выходные данные

Создание выходных данных

```
public abstract class ActionResult
{
    protected ActionResult();
    public abstract void ExecuteResult(ControllerContext
                                context);
}
```

Создание выходных данных

Тип	Описание	Вспомогательные методы
ViewResult	Отображает указанный шаблон представления или шаблон по умолчанию	View
PartialViewResult	Отображает указанный шаблон частичного представления или шаблон по умолчанию	PartialView
RedirectToRouteResult	Работает с HTTP перенаправлением 301 или 302 на метод действия или конкретный роут, генерируя URL в соответствии с вашей конфигурацией	RedirectToAction RedirectToActionPermanent RedirectToRoute RedirectToRoutePermanent

Создание выходных данных

Тип	Описание	Вспомогательные методы
RedirectResult	Работает с HTTP перенаправлением 301 или 302 на конкретный URL	Redirect RedirectPermanent
HttpUnauthorizedResult	Устанавливает ответный код HTTP статуса на 401 (что означает "не авторизирован"), который вызывает активный механизм аутентификации (form-аутентификацию или Windows-аутентификацию), чтобы попросить посетителя войти в систему	Нет

Создание выходных данных

Тип	Описание	Вспомогательные методы
HttpNotFoundResult	Возвращает HTTP ошибку 404— Not found	HttpNotFound
HttpStatusCodeResult	Возвращает указанный HTTP код	Нет
ContentResult	Возвращает необработанные текстовые данные в браузер, возможно установление заголовка типа содержимого	Content
FileResult	Является базовым классом для всех объектов, пишущих бинарный ответ во выходной поток. Предназначен для отправки файлов	File

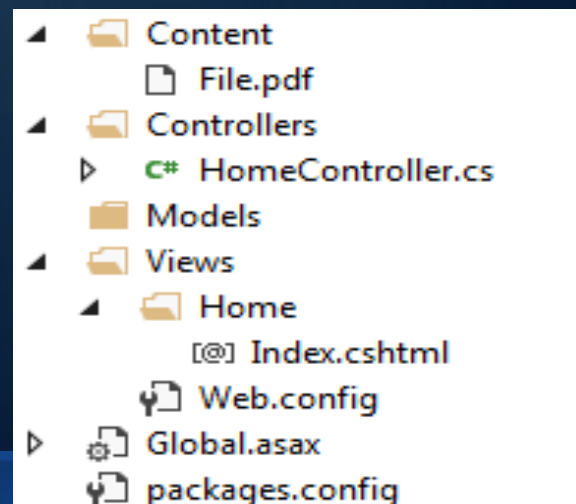
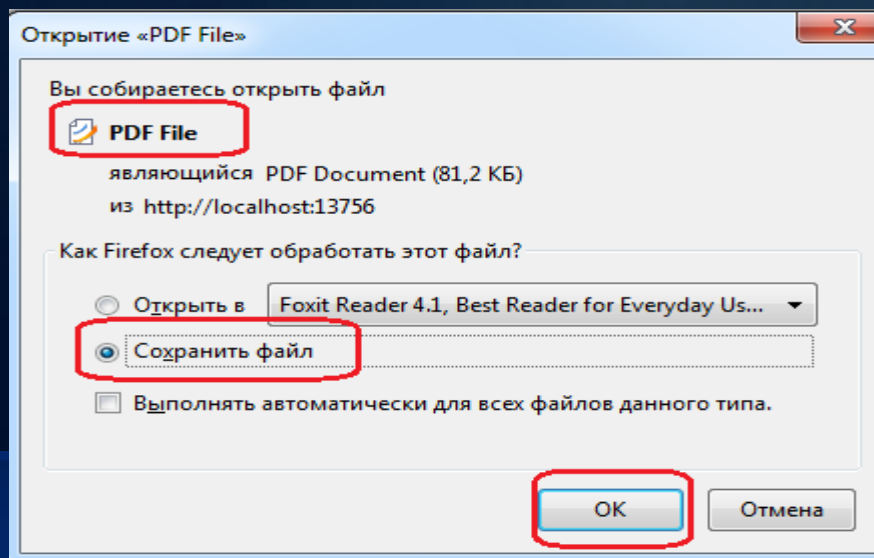
Создание выходных данных

Тип	Описание	Вспомогательные методы
EmptyResult	Ничего не делает, отправляет пустой ответ	None
JsonResult	Возвращает в качестве ответа объект или набор объектов в формате JSON	Json
JavaScriptResult	Возвращает в ответ в качестве содержимого код JavaScript	JavaScript

Создание выходных данных

```
public ActionResult DownloadFile()
{
    string filename = Server.MapPath("/Content/File.pdf");//полный путь к файлу
    string contentType = "application/pdf"; //тип файла который мы будем отдавать пользователю
                                           //MIME Type image/png image/jpg

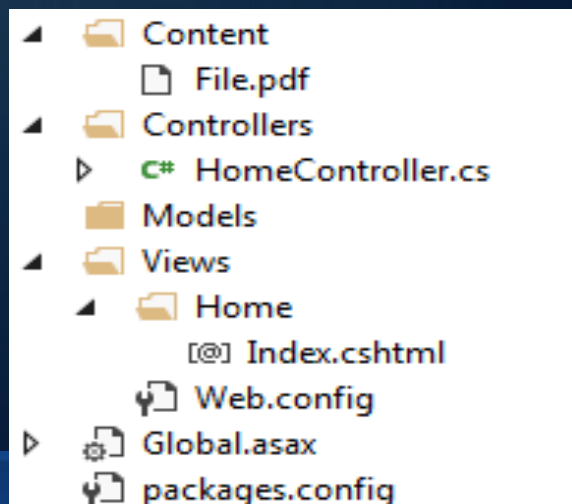
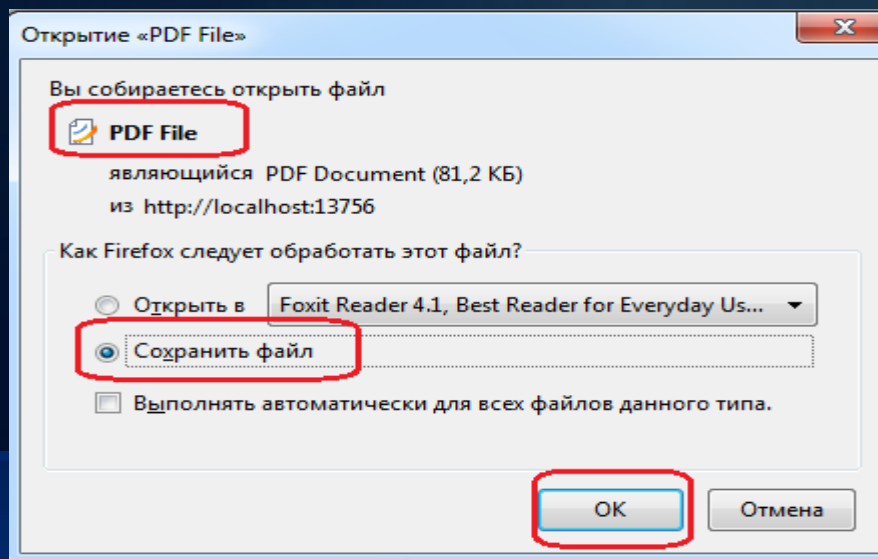
    string downloadName = "PDF File";
    // Если имя файла для скачивания не указано и если
    // браузер поддерживает тип файла, файл откроется в самом браузере.
    //downloadName = null;
    return File(filename, contentType, downloadName);
}
```



Создание выходных данных

```
public ActionResult DownloadFile()
{
    string filename = Server.MapPath("/Content/File.pdf");//полный путь к файлу
    string contentType = "application/pdf"; //тип файла который мы будем отдавать пользователю
                                                //MIME Type image/png image/jpg

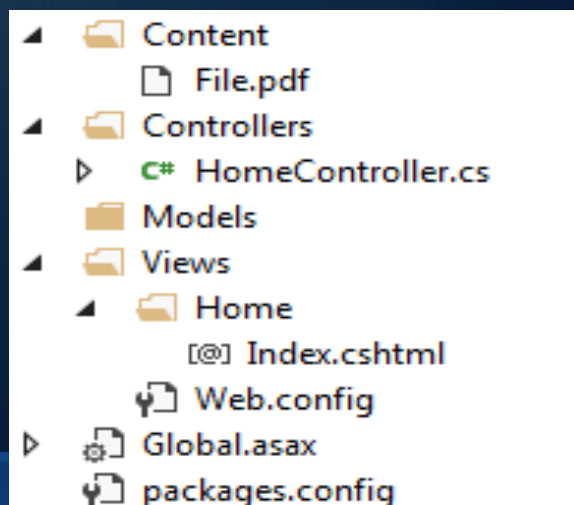
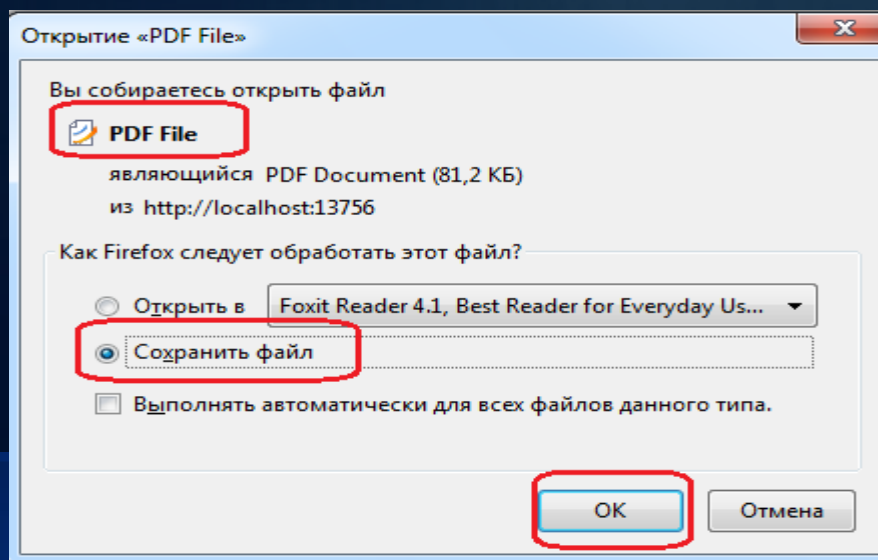
    string downloadName = "PDF File";
    // Если имя файла для скачивания не указано и если
    // браузер поддерживает тип файла, файл откроется в самом браузере.
    //downloadName = null;
    return File(filename, contentType, downloadName);
}
```



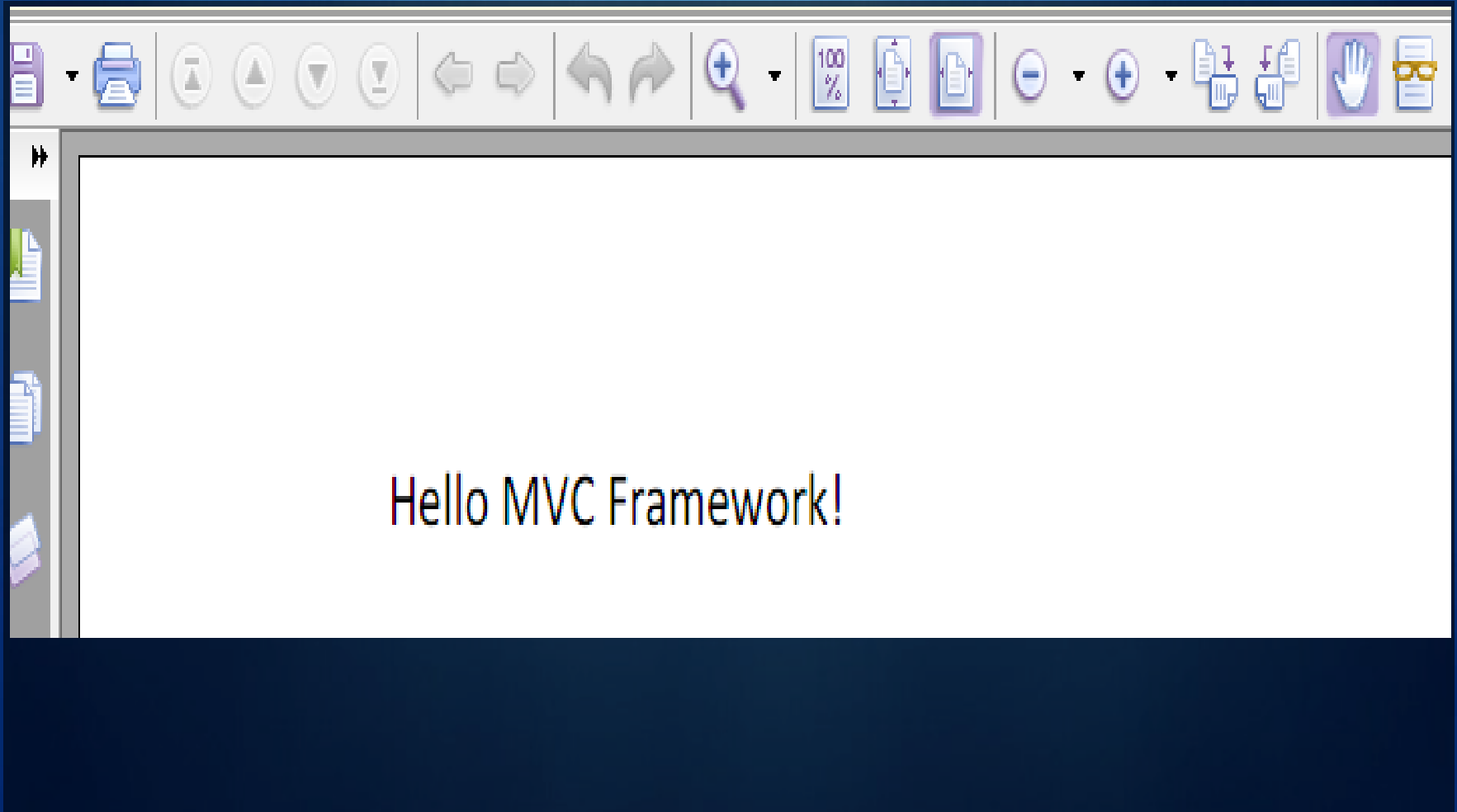
Создание выходных данных

```
public ActionResult DownloadFile()
{
    string filename = Server.MapPath("/Content/File.pdf");//полный путь к файлу
    string contentType = "application/pdf"; //тип файла который мы будем отдавать пользователю
                                           //MIME Type image/png image/jpg

    string downloadName = "PDF File";
    // Если имя файла для скачивания не указано и если
    // браузер поддерживает тип файла, файл откроется в самом браузере.
    //downloadName = null;
    return File(filename, contentType, downloadName);
}
```



Создание выходных данных



Создание выходных данных

```
public ActionResult DownloadBytes()
{
    string filename = Server.MapPath("/Content/File.pdf");
    string contentType = "application/pdf";

    byte[] data = System.IO.File.ReadAllBytes(filename);

    return File(data, contentType);
}
```

0 references

```
public ActionResult DownloadStream()
{
    string filename = Server.MapPath("/Content/File.pdf");
    string contentType = "application/pdf";

    FileStream stream = System.IO.File.OpenRead(filename);

    return File(stream, contentType);
}
```

Создание выходных данных

```
public ActionResult DownloadBytes()
{
    string filename = Server.MapPath("/Content/File.pdf");
    string contentType = "application/pdf";

    byte[] data = System.IO.File.ReadAllBytes(filename);

    return File(data, contentType);
}
```

0 references

```
public ActionResult DownloadStream()
{
    string filename = Server.MapPath("/Content/File.pdf");
    string contentType = "application/pdf";

    FileStream stream = System.IO.File.OpenRead(filename);

    return File(stream, contentType);
}
```

Создание выходных данных

```
public ActionResult DownloadBytes()
{
    string filename = Server.MapPath("/Content/File.pdf");
    string contentType = "application/pdf";

    byte[] data = System.IO.File.ReadAllBytes(filename);

    return File(data, contentType);
}
```

0 references

```
public ActionResult DownloadStream()
{
    string filename = Server.MapPath("/Content/File.pdf");
    string contentType = "application/pdf";

    FileStream stream = System.IO.File.OpenRead(filename);

    return File(stream, contentType);
}
```