### Особенности развертывания ASP NET веб-приложений

Классическое ASP NET MVC-приложение является ориентированным на развертывание на веб-сервере IIS. Оно включает в себя файл Global.asax, в котором определяют класс, наследующий от System.Web.HttpApplication. В этом классе конфигурируются отдельные компоненты MVC для необходимого поведения при обработке запросов. За маршрутизацию передачи HTTP-запросов определенным обработчикам отвечает класс Router.

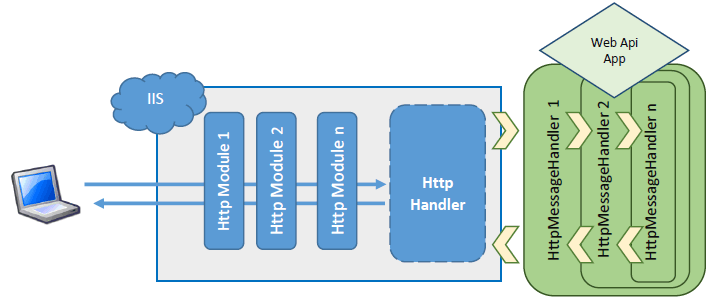
HTTP-обработчики тесно связаны с веб-сервером IIS, они используют классы HttpHandler и HttpContext для доступа к параметрам запроса и ответа. Отчасти такое неудобство произошло благодаря наследию от ASP NET Web Forms и использованию этих компонент повсеместно в MVC.

### &Rcy;&icy;&scy;&ucy;&ncy;&ocy;&kcy; 1. &Scy;&khcy;&iecy;&mcy;&acy; &ocy;&bcy;&rcy;&acy;&bcy;&ocy;&tcy;&kcy;&icy; HTTP-&zcy;&acy;&pcy;&rcy;&ocy;&scy;&ocy;&vcy; ASP NET MVC &vcy;&iecy;&bcy;-&pcy;&rcy;&icy;&lcy;&ocy;&zhcy;&iecy;&ncy;&icy;&yacy;

Таким образом, едва ли можно представить себе полноценное использование ASP NET как Open Source-проекта ввиду того, что размещать его можно только на IIS веб-сервере. В результате классическое ASP NET-приложение не может быть **самостоятельно развертываемым (self-hosted).**

Следующим шагом эволюции ASP NET стала разработка Web API. В нем компоненты System.Web используются не напрямую, а через адаптеры, также была устранена зависимость от IIS HTTP-обработчиков (HttpHandler) благодаря реализации слоя адаптеров.

Данный подход позволит избавиться от IIS зависимости, а Web API можно рассматривать в рамках Owin (Open Web Interface).



### Что такое Owin, и зачем он нам?

#### OWIN: Open Web Server Interface for .NET

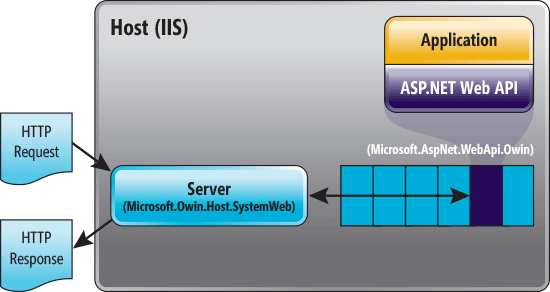
[Owin](http://owin.org/) (открытый веб сервер интерфейс для .NET) это ***спецификация*** с открытым исходным кодом, описывающая уровень абстракции между веб-сервером и компонентами приложения. Цель спецификации Owin является содействие простой, подключаемой архитектуры для веб-приложений на основе .NET и серверов, на которых они основываются.

***Owin -*спецификация,*не*реализация.*Таким образом, Owin описывает минимальный набор типов, и один*делегат*приложения, через которые будет происходить взаимодействие между компонентами приложения и сервером.***

Owin предоставляет следующие [общие определения](http://owin.org/spec/spec/owin-1.0.0.html#Definition) для программных элементов в приложении:

Стек состоит из следующих слоев(первый вариант)

* ***Server****- HTTP-сервер, который непосредственно общается с клиентом, а затем использует семантику Owin для обработки запросов.*
* ***Web Application*** *- конкретное приложение, возможно, построенное на вершине веб-платформы, которая запускается с помощью Owin совместимых серверов.*
* ***Middleware****- набор компонентов, обрабатывающих запросы в конвейере OWIN.*
* ***Host****- процесс, который обрабатывает приложение. Если в качестве сервера приложения используется IIS, то подобным процессом является процесс самого сервера IIS. Однако им может быть и сам файл приложения. Основная его задача - запуск, загрузка компонентов OWIN и корректное завершение.*



*«Зачем лишний уровень абстракции и в без того* [*перегруженный конвейер*](http://www.msdr.ru/48/) *IIS/ASP.NET Незачем, просто это одна из реализаций организации работы приложения по спецификации OWIN с использованием IIS/ASP.NET.»*

***Host*** в Owin-совместимом приложении это *словарь окружающей среды (*Environment Dictionary) в виде ***IDictionary <String, Object>, содержащий все необходимые сведения о request, response***. Спецификация определяет Owin как минимальный набор ключей и значений, которые должны присутствовать в словаре. Однако, серверы, принимающей среды, промежуточного ПО и код приложения может добавить дополнительные данные или изменить данные в словаре.

Для ASP.NET разработчика, контекст HTTP означает конкретный объект класса **System.Web.HttpContext** –и только в последних версиях ASP.NET была частично заменен на **HttpContextBase.**

**Спецификация Owin определяет абстракцию - это Func<IDictionary<string, object>, Task>;**

Взаимодействие сводится к использованию функций с сигнатурой одного единственного делегата, так называемого ***делегата приложения***. Который имеет следующий вид:

**using** AppFunc = Func<IDictionary<**string**, **object**>, Task>;

Входной параметр имеющий тип **IDictionary<string, object>** и называемый – *словарь окружения,* который содержит все данные (переменные сервера, данные запроса и т.п.). ***Он не может иметь значение null***, а так же быть пустым.

Словарь Owin-частично приведен ниже.

|  |  |  |
| --- | --- | --- |
| **Ключ** | **Требуется** | **Объект** |
| **owin.RequestBody** | Да | Содержит объект потока, наполненное тела запроса. |
| **owin.RequestHeaders** | Да | Содержит словарь заголовков HTTP запроса. Фактический тип установлен как IDictionary <String, String []>. |
| **owin.RequestMethod** | Да | Содержит строку, которая указывает метод HTTP запроса: GET, POST или другим общим HTTP глаголы. |
| **owin.RequestPath** | Да | Содержит путь к пути запроса. Путь строка относительно корня сайта. |
| **owin.RequestPathBase** | Да | Содержит часть пути запроса, соответствующего корня сайта. |
| **owin.RequestProtocol** | Да | Содержит имя и версию протокола. В качестве примера, это может быть HTTP / 1.0 или HTTP / 1.1. |
| **owin.RequestQueryString** | Да | Содержит строку запроса компонент запроса HTTP без ведущих "?". Он содержит пустую строку, если нет строки запроса не найдено. |
| **owin.RequestScheme** | Да | Содержит схему URI например HTTP или HTTPS. |
| **owin.ResponseBody** | Да | Содержит объект потока для получения ответа. |
| **owin.ResponseHeaders** | Да | Содержит словарь заголовков HTTP ответа. Фактический тип установлен как IDictionary <String, String []>. |
| **owin.ResponseStatusCode** | Нет | Содержит целое число, обозначающее код статуса HTTP ответа. |
| **owin.ResponseReasonPhrase** | Нет | Содержит причина фраза ассоциируется данный код состояния |
| **owin.ResponseProtocol** | Нет | Содержит имя и версию протокола. В качестве примера, это может быть HTTP / 1.0 или HTTP / 1.1. |
| **owin.CallCancelled** | Нет | Содержит информацию, указывающую, был ли запрос отменен или прерван. |
| **owin.Version** | Нет | Содержит строку, которая указывает версию спецификации Owin. |

В таблице содержит только базовый набор ключей словаря, имеющих отношение к веб-серверам и приложениям необходимых для общения;

можно добавлять пользовательские данные, если это необходимо.

Выходной параметр имеет тип **Task** – инкапсулирует некоторую выполняемую задачу. На основе данного делегата строятся компонеты ***приложения и объединяются в цепочку, тем самым формируется конвейер обработки запросов OWIN, куда передаётся запрос на обработку***. Из типа возвращаемого делегатом уже видно, что он будет работать в ***асинхронном режиме***.

***Microsft OWIN или проект под названием Katana является реализацией спецификации OWIN компанией Microsoft. Исходный код проекта полность открытый и находится на CodePlex по адресу*** [***https://katanaproject.codeplex.com/***](https://katanaproject.codeplex.com/)





OWIN показанная на диаграммах как промежуточный слой это основное связующее звено в приложении промежуточного слоя, представленное в виде единственного интерфейса **IAppBuilder**.

**namespace** Owin

{

**using** System;

**using** System.Collections.Generic;

**public** **interface** IAppBuilder

{

**object** Build(Type returnType);

IAppBuilder New();

IAppBuilder Use(**object** middleware, **params** **object**[] args);

IDictionary<**string**, **object**> Properties { **get**; }

}

}

***Интерфейс не является частью спецификации.***

IAppbuilder имеет ***4 метода для добавления промежуточного программного обеспечения в конвейер OWIN(не учитывая трех остальных!).*** Один в Owin.dll

namespace Owin

{

public interface IAppBuilder

{

//Removed for brevity

IAppBuilder Use(object middleware, params object[] args);

}

}

и еще 3 в качестве методов расширения в Microsoft.Owin.dll

namespace Owin

{

public static class AppBuilderUseExtensions

{

public static void Run(this IAppBuilder app, Func<Microsoft.Owin.IOwinContext, System.Threading.Tasks.Task> handler);

public static IAppBuilder Use(this IAppBuilder app, Func<Microsoft.Owin.IOwinContext, Func<System.Threading.Tasks.Task>, System.Threading.Tasks.Task> handler);

public static IAppBuilder Use<T>(this IAppBuilder app, params object[] args);

}

}

Создадим ***свой self host***

*Создаем простое консольное приложение SelfHost*

*Manage NuGet Package добавляем Microsoft.Owin, Microsoft.Owin.Hosting, Microsoft.Owin.Host.HttpListener.*

Сборки

* OWIN, содержит один едиственный интерфейс – IAppBuilder, который был описан выше.
* Microsoft.Owin – содержит множество вспомагательных типов, которые облегчают создание разных компонетов Owin. Например: OwinRequest, OwinResponse, OwinContext и т.д.
* Microsoft.Owin.Hosting содержит типы необходимые для хостинга и запуска приложения.
* Microsoft.Owin.Host.HttpListener содержит дефолтовый сервер OwinHttpListener проекта Katana. Это обёртка вокруг стандартного ***System.Net.HttpListener***.

class Program

{

static void Main(string[] args)

{

using (Microsoft.Owin.Hosting.WebApp.Start<Startup1>("http://localhost:8080"))

{

Console.WriteLine("Сервер запущен. Нажмите любую клавишу для завершения работы...");

Console.ReadLine();

}

}

}

**Для конфигурации приложения при запуске, Katana ищет класс Startup. Добавляем OWIN Startup**

public class Startup1

{

private const string htmlText = "<html>" +

"<head>" +

"<title>Hello OWIN</title>" +

"</head>" +

"<body>" +

"<h1>Simple Owin Application<h1>" +

"</body>" +

"</html>";

public void Configuration(IAppBuilder app)

{

// 1 вариант

app.Run(context =>

{

context.Response.ContentType = "text/html";

return context.Response.WriteAsync(htmlText);

});

}

}

Используем метод расширения Run класса ***AppBuilderUseExtensions***

Можно также использовать метод IAppBuilder.Use (объект middleware, params object [] args) из Owin.dll для создания очень простого промежуточного ПО, чтобы приветствовать пользователя. Метод IAppBuilder.Use () принимает объект реализации промежуточного программного обеспечения и *необязательные* аргументы данных для промежуточного программного обеспечения.

// 2 вариант

app.Use(new Func<AppFunc, AppFunc>(next => (async context =>

{

using (var writer = new StreamWriter(context["owin.ResponseBody"] as Stream))

{

await writer.WriteAsync(htmlText);

}

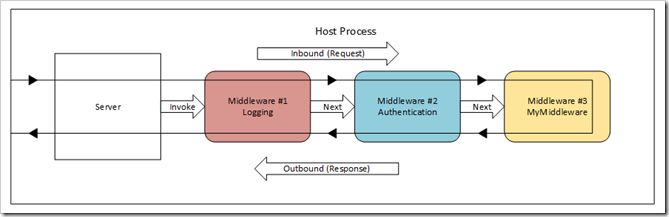
await next.Invoke(context);

})));

Где using AppFunc = Func<IDictionary<string, object>, Task>;

Объектное промежуточное ПО может быть ссылкой делегата на метод реализации промежуточного программного обеспечения или встроенную промежуточную реализацию. Реализация должна соблюдать синтаксис промежуточного программного обеспечения owin для добавления в конвейер Func <in AppFunc, out AppFunc>

### Таким образом мы реализуем ковеер обработки



Отдельные компоненты приложений (middleware) выполняют свои обязанности по обработке, когда вызывается **AppFunc** delegate. Для того, чтобы поддерживать конвеер, каждый компонент middleware также отвечает за вызов следующего компонента в цепи или намеренно не вызывает следующий компонент и замыкает последовательность.

Еще один пример

Middleware Delegate Signature:

Func<AppFunc, AppFunc>

//Func<AppFunc, AppFunc>

//Example Middleware as Function:

public AppFunc MyMiddleWare(AppFunc next)

{

AppFunc appFunc = async (IDictionary<string, object> environment) =>

{

// Do something with the incoming request:

var response = environment["owin.ResponseBody"] as Stream;

using (var writer = new StreamWriter(response))

{

await writer.WriteAsync("<h1>Hello from My First Middleware</h1>");

}

// Call the next Middleware in the chain:

await next.Invoke(environment);//!!!

};

return appFunc;

}

Добавим еще два

public AppFunc MyOtherMiddleWare(AppFunc next)

{

AppFunc appFunc = async (IDictionary<string, object> environment) =>

{

// Do something with the incoming request:

var response = environment["owin.ResponseBody"] as Stream;

using (var writer = new StreamWriter(response))

{

await writer.WriteAsync("<h1>Hello from My Second Middleware</h1>");

}

// Call the next Middleware in the chain:

await next.Invoke(environment);

};

return appFunc;

}

public AppFunc ThirdMiddleware(AppFunc next)

{

AppFunc appFunc = async (IDictionary<string, object> environment) =>

{

// Do something with the incoming request:

var response = environment["owin.ResponseBody"] as Stream;

using (var writer = new StreamWriter(response))

{

await writer.WriteAsync("<h1>Hello from My Third Middleware</h1>");

}

// Call the next Middleware in the chain:

await next.Invoke(environment);

};

return appFunc;

}

В Startup1

public void Configuration(IAppBuilder app)

{

var middleware = new Func<AppFunc, AppFunc>(MyMiddleWare);

var otherMiddleware = new Func<AppFunc, AppFunc>(MyOtherMiddleWare);

var thirdMiddleware = new Func<AppFunc, AppFunc>(ThirdMiddleware);

app.Use(middleware);

app.Use(otherMiddleware);

app.Use(thirdMiddleware);

}

Если закоментировать в MyOtherMiddleWare ***await next.Invoke(environment);***, то конвеер прервется.

Web API – разрабатывалась Microsoft по спецификации OWIN, и это означает, что мы можем ***захостить*** сервис не только в IIS, но и в консольном или другом приложении!!!

https://docs.microsoft.com/en-us/aspnet/web-api/overview/hosting-aspnet-web-api/use-owin-to-self-host-web-api

*Создаем простое консольное приложение 03Step-WebAPI*

Через Manage package NuGet добавляем пакет

### Microsoft.AspNet.WebApi.OwinSelfHost

Добавим в наше приложение класс

|  |  |  |
| --- | --- | --- |
|  |  | public class Startup  {  public void Configuration(IAppBuilder app)  {  // Configure Web API for self-host.  HttpConfiguration config = new HttpConfiguration();  config.Routes.MapHttpRoute(  name: "DefaultApi",  routeTemplate: "api/{controller}/{id}",  defaults: new { id = RouteParameter.Optional }  );  app.UseWebApi(config);  }  } |

Добавляем FruitsController

public class FruitsController : ApiController//using System.Web.Http;

{

static List<string> fruits = new List<string>

{

"Apple", "Banana", "Orange", "Cocos"

};

public IEnumerable<string> GetAll()

{

return fruits;

}

public HttpResponseMessage Get(int id)

{

if (id < fruits.Count)

return Request.CreateResponse(HttpStatusCode.OK, fruits[id]);

return Request.CreateResponse(HttpStatusCode.NotFound, "Item not found");

}

public HttpResponseMessage Post([FromBody]string fruit)

{

fruits.Add(fruit);

HttpResponseMessage msg = Request.CreateResponse(HttpStatusCode.Created, fruit);

// Location заголовок стоит создавать, если новый элемент был создан

msg.Headers.Location = new Uri(Request.RequestUri + "/" + (fruits.Count - 1));

return msg;

}

public HttpResponseMessage Put(int id, [FromBody]string value)

{

if (id > fruits.Count)

{

return Request.CreateResponse(HttpStatusCode.NotFound);

}

else

{

fruits[id] = value;

return Request.CreateResponse(HttpStatusCode.OK, fruits[id]);

}

}

public HttpResponseMessage Delete(int id)

{

if (id > fruits.Count)

{

return Request.CreateResponse(HttpStatusCode.NotFound);

}

else

{

string deleted = fruits[id];

fruits.RemoveAt(id);

return Request.CreateResponse(HttpStatusCode.OK, deleted);

}

}

}

Остается только в консоли захостить наш сервис

class Program

{

static void Main(string[] args)

{

string baseAddress = "http://localhost:9000/";

// Start OWIN host

using (WebApp.Start<Startup>(url: baseAddress))

{

// Create HttpCient and make a request to api/values

HttpClient client = new HttpClient();

var response = client.GetAsync(baseAddress + "api/values").Result;

Console.WriteLine(response);

Console.WriteLine(response.Content.ReadAsStringAsync().Result);

Console.ReadLine();

}

}

}

в файл Startup

HttpConfiguration config = new HttpConfiguration();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

app.UseWebApi(config);

Для обработки статического контента – файлов html, js, css –необходимо добавить пакет

### Microsoft.Owin.StaticFiles, Microsoft.Owin.FileSystems

И в файл Startup

HttpConfiguration config = new HttpConfiguration();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

app.UseFileServer(new FileServerOptions()

{

FileSystem = new PhysicalFileSystem(GetDirectory("Content")),

EnableDirectoryBrowsing = true,

RequestPath = new Microsoft.Owin.PathString("/html")

});

app.UseFileServer(new FileServerOptions()

{

FileSystem = new PhysicalFileSystem(GetDirectory("Scripts")),

EnableDirectoryBrowsing = true,

RequestPath = new Microsoft.Owin.PathString("/scripts")

});

app.UseWebApi(config);