Host开发

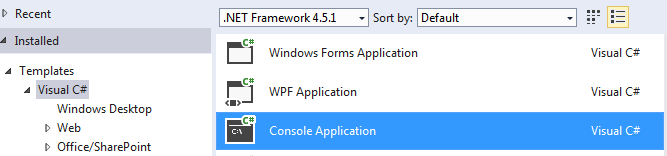
如上篇博文提及,Host具有如下特点:

实现一个宿主进程

负责Server的启动和关闭

负责Middleware和Application的装载

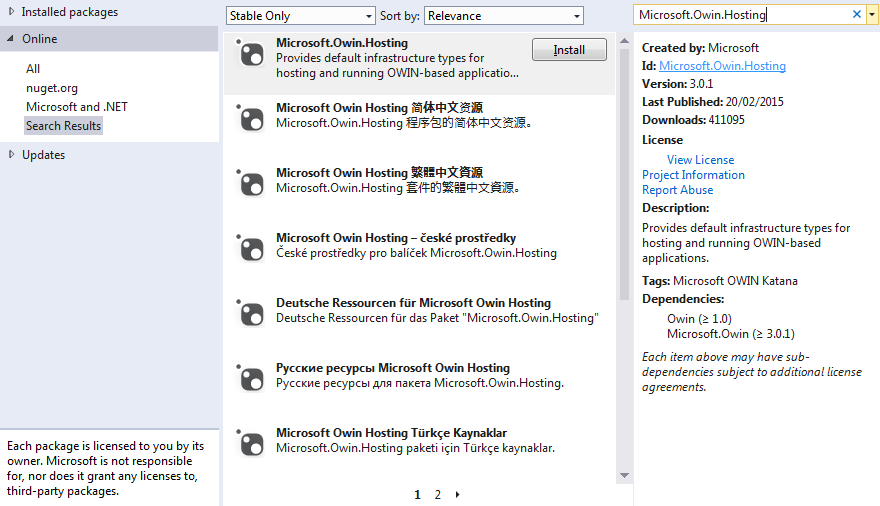
最简单的宿主进程就是Console Application,那么我们从建立Console程序开始



注意这个程序和它的入口类(一般是Program.cs)就是我们所说的Host宿主进程的实现.

而具体的Host和Server的实现我们就需要借助Kanata项目的实现了.

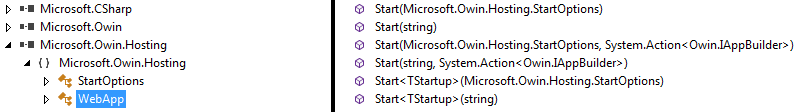
首先通过Nuget获取Kanata的Host实现. 包名为: Microsoft.Owin.Hosting



注: 目前版本为3.0.1, Owin的2个核心组件:  Owin和Microsoft.Owin会被同时载入.

WebApp.Start

Host的主要用途是启动和关闭Server, 这个功能的实现类就是在Microsoft.Owin.Hosting下的WebApp类, 请看类结构:



我们可以看到,WebApp只有若干个Start函数,辅助于StartOptions类,依靠这2个类,就能完成Host对Server的启动和关闭.

Start函数有好几个重载,我们来看一个最为典型的实现

public static System.IDisposable Start(Microsoft.Owin.Hosting.StartOptions options, System.Action<IAppBuilder> startup)

Options参数定义了Server启动所需的参数.

Startup参数是一个以Owin.IAppbuilder接口为参数的函数,通过这个函数,完成对Server所需Middleware的加载工作,这也是Host的关键作用之一!

函数返回一个实现IDisposable接口的类, 可以预见的是,当这个类Dispose的时候,被这个函数启动的Server也会同时关闭和消亡.

StartOptions

我们先来看下这个StartOptions中2个比较关键的属性:

public System.Collections.Generic.IList<string> Urls { get; }

Urls参数是以http标准url为格式字符串来定义Server监听的HostName和Port.

标准格式是 http(s)://hostname:port

Urls可以加多个,表示支持不同的hostname和port映射. 另外, 还支持[http://\*:9000](http://*:9000/)

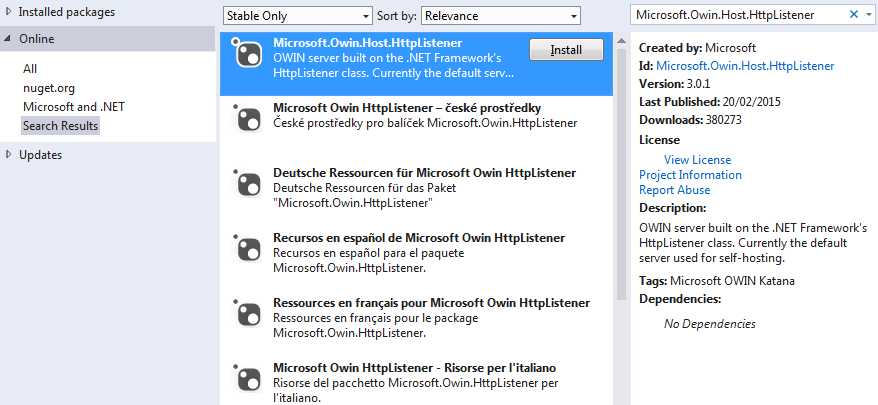
public string ServerFactory { set; get; }

关键的来了,这个就是Server实现类的assembly name

Server实现和引用

Http Server并不是一两句代码可以实现的, Server实现包名为 :  Microsoft.Owin.Host.HttpListener

首先还是使用Nuget获取该Server组件



安装,并引入项目,有了Server的实现,下面我们来完成StartOptions的定义代码:

//初始化StartOptions参数

StartOptions options = new StartOptions();

//服务器Url设置

options.Urls.Add("http://localhost:9000");

options.Urls.Add("http://192.168.1.1:8080");

//Server实现类库设置

options.ServerFactory = "Microsoft.Owin.Host.HttpListener";

需要注意的是,我这里特地放了2个port不同的地址,这2个Url都能起到效果; 另外Microsoft.Owin.Host.HttpListener 其实是默认的Host ServerFatory实现(最后一行代码可以省略);

Startup函数和使用

Startup函数非常的简单,就是一个只有一个没有返回值的,只有一个Owin.IAppBuilder参数的函数(函数名任意,不一定要叫Startup). 如以下代码:

private static void Startup(Owin.IAppBuilder app)

{

//这里通过app句柄,为当前Server加入所有需要的middleware

}

Server的启动和关闭

有了Options和Startup函数,我们就可以启动我们的Server了,整合的代码如下:

[IMG_262](https://www.cnblogs.com/zergcom/p/javascript:void(0);)

/// <summary>

/// Owin Host 主进程入口函数

/// </summary>

static void Main()

{

//初始化StartOptions参数

StartOptions options = new StartOptions();

//服务器Url设置

options.Urls.Add("http://localhost:9000");

options.Urls.Add("http://192.168.1.1:8080");

//Server实现类库设置

options.ServerFactory = "Microsoft.Owin.Host.HttpListener";

//以当前的Options和Startup启动Server

using (WebApp.Start(options, Startup))

{

//显示启动信息,通过ReadLine驻留当前进程

Console.WriteLine("Owin Host/Server started,press enter to exit it...");

Console.ReadLine();

}//Server在Dispose中关闭

}

private static void Startup(Owin.IAppBuilder app)

{

//这里通过app句柄,为当前Server加入所有需要的middleware

}

[IMG_263](https://www.cnblogs.com/zergcom/p/javascript:void(0);)

Server的关闭很简单,当using的结尾触发Dispose时,该Server被自动关闭.

Server是独立线程运行的,所以宿主进程必须驻留,这里用Console.ReadLine()驻留当前线程,保证Server长期运行.

其他注意点

很多机器上出现权限不够情况,请用管理员账号运行VS2013.

总结下,我们开发了一个以Console Application为宿主进程的程序,通过Kanata的Host实现Microsoft.Owin.Hosting启动了它自身的Server实现

Kanata的Host和Server实现完全可以被自己或者第三方的实现所取代,前提是,符合OWIN的标准.