<http://beginor.github.io/2015/01/24/oauth2-server-with-owin.html>

# 使用 OWIN 搭建 OAuth2 服务器

## 关于 OAuth

维基百科中对 OAuth 的描述如下：

OAuth（开放授权）是一个开放标准，允许用户让第三方应用访问该用户在某一网站上存储的私密的资源（如照片，视频，联系人列表），而无需将用户名和密码提供给第三方应用。

OAuth允许用户提供一个令牌，而不是用户名和密码来访问他们存放在特定服务提供者的数据。每一个令牌授权一个特定的网站（例如，视频编辑网站)在特定的时段（例如，接下来的2小时内）内访问特定的资源（例如仅仅是某一相册中的视频）。这样，OAuth让用户可以授权第三方网站访问他们存储在另外服务提供者的某些特定信息，而非所有内容。

很多知名网站都采用支持[OAuth2](https://tools.ietf.org/html/rfc6749)认证， 允许第三方应用接入， 客户端接入 OAuth2 服务器这方面的资料已经很多了， 但是关于怎么搭建自己的 OAuth 服务器这方面的资料则比较少， 接下来就介绍一下怎么用微软的 OWIN 中间件搭建自己的 OAuth 服务， 实现 OAuth2 框架中的认证服务器和资源服务器 。

## 使用 OWIN 搭建 OAuth2 认证服务器

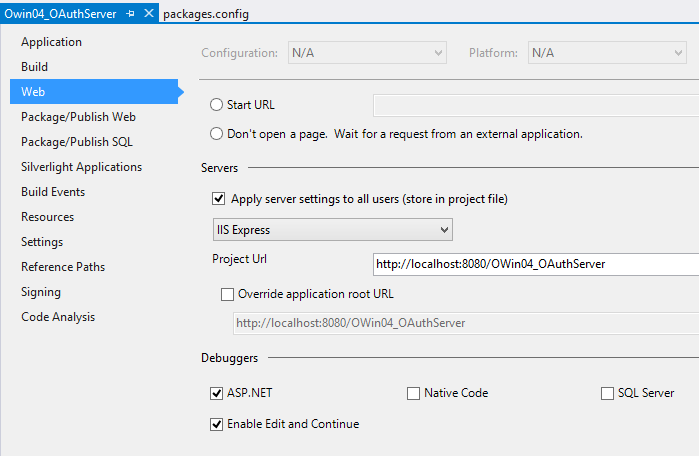
认证服务器指 [authorization server](https://tools.ietf.org/html/rfc6749#section-1.1) ， 负责在资源所有者 （最终用户） 通过认证之后， 向客户端应用颁发凭据 (code) 和对客户端授权 (access\_token) 。

### 新建空的 Web 项目

这一步很容易， 只要用 Visual Studio 新建一个空的 Web 项目， 并用 nuget 管理器添加下面几个 package：

* Owin
* Microsoft.Owin
* Microsoft.Owin.Host.SystemWeb
* Microsoft.Owin.Security
* Microsoft.Owin.Security.Cookies
* Microsoft.Owin.Security.OAuth
* Microsoft.AspNet.Mvc

打开项目属性， 设置项目使用 IIS Express ， 虚拟目录为 /OWin04\_OAuthServer , 如下图所示：



这个配置是可选的， 也可以设置成你自己喜欢的值。

### 配置 OWIN OAuth 中间件

添加一个 OWIN Startup Class ， 名称为 Startup.cs ， 如下所示：

using Microsoft.Owin;

using Owin;

[assembly: OwinStartup(typeof(Owin04\_OAuthServer.Startup))]

namespace Owin04\_OAuthServer {

public partial class Startup {

public void Configuration(IAppBuilder app) {

ConfigureAuth(app);

}

}

}

在项目中添加 App\_Start 目录， 并在这个目录添加一个部分类文件 Startup.Auth.cs ， 在这个文件中来实现上面的 ConfigureAuth 方法：

using Owin04\_Consts;

using Microsoft.Owin.Security.OAuth;

using System.Threading.Tasks;

using Microsoft.Owin.Security.Infrastructure;

namespace Owin04\_OAuthServer {

partial class Startup {

public const string AuthenticationType = "OAuth2";

private void ConfigureAuth(IAppBuilder app) {

}

}

}

既然要认证用户， 首先启用 Cookie 认证， 在 ConfigureAuth 方法中添加下面的代码：

app.UseCookieAuthentication(new CookieAuthenticationOptions {

AuthenticationType = AuthenticationType,

AuthenticationMode = AuthenticationMode.Passive,

LoginPath = new PathString(Paths.LoginPath),

LogoutPath = new PathString(Paths.LogoutPath)

});

上面的代码使用 CookieAuthenticationOptions 来启用 Cookie 认证中间件， 这个中间件有点儿类似于 Asp.Net 的表单认证 (FormsAuthentication) , 点击这里[查看 CookieAuthenticationOptions 的 MSDN 文档](https://msdn.microsoft.com/en-us/library/microsoft.owin.security.cookies.cookieauthenticationoptions(v=vs.113).aspx)。

接下来启用 OAuth 认证服务器中间件， 代码如下：

app.UseOAuthAuthorizationServer(new OAuthAuthorizationServerOptions {

AuthorizeEndpointPath = new PathString(Paths.AuthorizePath),

TokenEndpointPath = new PathString(Paths.TokenPath),

ApplicationCanDisplayErrors = true,

AllowInsecureHttp = true,

// Authorization server provider 控制认证服务器的生命周期

Provider = new OAuthAuthorizationServerProvider {

OnValidateClientRedirectUri = ValidateClientRedirectUri,

OnValidateClientAuthentication = ValidateClientAuthentication,

OnGrantResourceOwnerCredentials = GrantResourceOwnerCredentials,

OnGrantClientCredentials = GrantClientCredetails

},

// Authorization code provider 创建和接收认证码

AuthorizationCodeProvider = new AuthenticationTokenProvider {

OnCreate = CreateAuthenticationCode,

OnReceive = ReceiveAuthenticationCode,

},

// Refresh token provider 创建和接收访问凭据访问凭据

RefreshTokenProvider = new AuthenticationTokenProvider {

OnCreate = CreateRefreshToken,

OnReceive = ReceiveRefreshToken,

}

});

通过上面的 UseOAuthAuthorizationServer 扩展方法来配置 OAuth 认证中间件， OAuthAuthorizationServerOptions ， 这个类有几个重要的属性， 说明如下：

* **AuthorizeEndpointPath** : 客户端应用将用户浏览器重定向到用户同意颁发令牌或代码的地址， 必须以前倒斜杠 “/” 开始， 例如： /Authorize ；
* **TokenEndpointPath** : 客户端应用可以直接访问并得到访问令牌的地址， 必须以前倒斜杠 “/” 开始， 例如： /Token ， 如果想客户端颁发了 [client\_secret](https://tools.ietf.org/html/rfc6749#appendix-A.2) ， 那么客户端必须将其发送到这个地址；
* **ApplicationCanDisplayErrors** : 如果希望在 /Authorize 这个地址显示自定义错误信息， 则设置为 true ， 只有当浏览器不能被重定向到客户端时才需要， 比如 client\_id 和 redirect\_uri 不正确； /Authorize 节点可以通过提取添加到 OWIN 环境的 oauth.Error 、 oauth.ErrorDescription 和 oauth.ErrorUri 属性来显示错误； 如果设置为 false ， 客户端浏览器将会被重定向到默认的错误页面；
* **AllowInsecureHttp** : 如果允许客户端的 return\_uri 参数不是 HTTPS 地址， 则设置为 true 。
* **Provider** : 应用程序提供和 OAuth 认证中间件交互的 IOAuthAuthorizationServerProvider 实例， 通常可以使用默认的 OAuthAuthorizationServerProvider ， 并设置委托函数即可。
* **AuthorizationCodeProvider** : 提供返回给客户端能且只能使用一次的认证码， 出于安全性考虑， OnCreate/OnCreateAsync 生成的认证码**必须**只能在 OnReceive/OnReceiveAsync 使用一次；
* **RefreshTokenProvider** : 刷新令牌， 如果这个属性没有设置， 则不能从 /Token 刷新令牌。

想了解更多信息， 可以[查看 OAuthAuthorizationServerOptions 的 MSDN 文档](https://msdn.microsoft.com/en-us/library/microsoft.owin.security.oauth.oauthauthorizationserveroptions(v=vs.113).aspx)。

### 用户管理与登录

OAuth 并不关注用户的管理， 在 ASP.NET 中， 应该有 Membership 或者 Identity 来完成， 但是 OAuth 又依赖于用户登录， 在这里仅创建一个简单的登录视图来实现用户登录的功能， 登录视图的代码如下：

<form method="POST" action="account/login">

<input id="login-username" type="text" name="username" />

<input id="login-password" type="password" name="password" />

<input id="login-remember" type="checkbox" name="isPersistent" value="1"/>

<input id="login-signin" type="submit" name="submit.Signin" value="Login" />

</form>

对应的 AccountController 代码如下：

[Authorize]

public class AccountController : Controller {

[AllowAnonymous]

public ActionResult Login() {

var authentication = HttpContext.GetOwinContext().Authentication;

if (Request.HttpMethod == "POST") {

var isPersistent = !string.IsNullOrEmpty(Request.Form.Get("isPersistent"));

// 作为示例程序， 这里没有对用户进行验证， 直接登录用户输入的账户。

if (!string.IsNullOrEmpty(Request.Form.Get("submit.Signin"))) {

authentication.SignIn(

new AuthenticationProperties { IsPersistent = isPersistent },

new ClaimsIdentity(new[] { new Claim(ClaimsIdentity.DefaultNameClaimType, Request.Form["username"]) }, Startup.AuthenticationType)

);

}

}

return View();

}

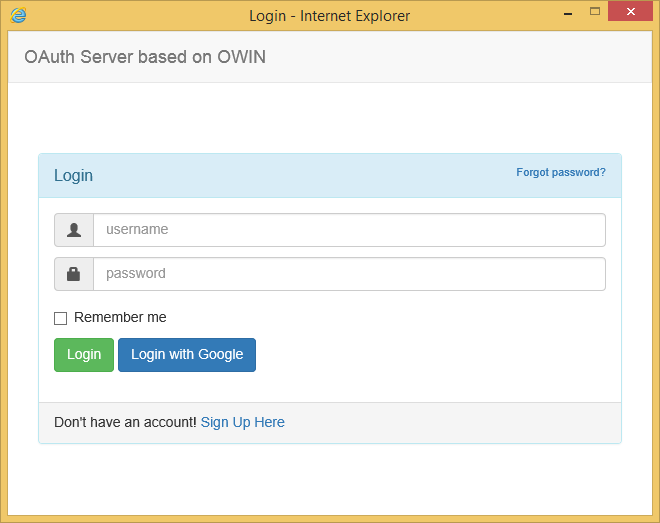
public ActionResult Logout() {

return View();

}

}

以上只是部分代码， 登录页面实际看起来是这样的:



## 创建受保护资源服务

作为例子， 本文创建一个简单的 WebAPI 项目，向通过 OAuth 授权认证的第三方应用返回用户信息。

仍然是新建一个空的 Web 项目， 添加下面的 NuGet 包：

* Owin
* Microsoft.Owin
* Microsoft.Owin.Host.SystemWeb
* Microsoft.Owin.Security
* Microsoft.Owin.Security.OAuth
* Microsoft.AspNet.WebApi.Owin
* Microsoft.AspNet.WebApi.Core
* Microsoft.AspNet.WebApi.Client

在 Startup.cs 的 Configuration 方法中添加下面的代码， 启用 Bearer 验证和 WebAPI ：

public void Configuration(IAppBuilder app) {

app.UseOAuthBearerAuthentication(new OAuthBearerAuthenticationOptions());

var config = new HttpConfiguration();

config.Formatters.Clear();

config.Formatters.Add(new JsonMediaTypeFormatter());

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

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

defaults: new { id = RouteParameter.Optional }

);

app.UseWebApi(config);

}

再创建一个 UserController 类， 返回一些示例信息， 代码如下：

[Authorize]

public class UserController : ApiController {

public object Get() {

var identity = User.Identity as ClaimsIdentity;

var infos = identity.Claims.Where(claim => claim.Type == "urn:oauth:scope")

.Select(claim => claim.Value)

.ToDictionary(s => s, s => s + " value is xxx.");

return new { name = identity.Name, infos };

}

}

为了能让资源服务器识别认证服务器颁发的令牌， 需要配置两个应用的 machineConfig 为相同的 key ， 如下所示：

<machineKey

decryptionKey="C11B54C2F10E4689AC59A84F79CDB494AE326344F26B1DC5"

validation="SHA1"

validationKey="7E1457A6E6DF475AA972D2106C0A2C3A44BC023F3E274B6FB598A1265C3C5374EA17DC9669C143BDB125E319164438974061AFCAA42A4478A07C3EA093517A48" />

到现在为止， 基于 OWIN 的 OAuth 认证服务器和资源服务器已经建好了， 接下来会另起一篇文章说明怎么使用这两个服务器。