授权认证服务设计说明

# 1,授权模式

本系统采用OAuth 2.0 开放式授权协议，采用授权与认证相分离的模式，采用OAuth 2.0协议中的密码授权模式。

# 2,系统角色

在 OAuth 2.0 授权体系中，有下面几种角色：

【授权服务器】：对客户端对资源服务器的访问进行授权；

【资源服务器】：提供API等资源服务的服务器，是资源的提供者，供客户端访问；

【客户端】：访问资源服务器的对象，它可以是一个客户端程序，也可以是另外一个Web站点，甚至是另外一个资源服务器，准确的说，它是“资源”的消费者。

【认证服务器】：验证客户端的身份，根据使用OAuth 2.0协议模式的不同验证方式也不同，在本例中，采用验证用户名和密码的模式。

【用户】：使用客户端的终端用户，用户需要首先授权给自己可信任的客户端程序或者Web站点，然后系统后续流程才能运行。

# 3,授权流程

本系统虽然提供了4种OAuth 2.0 的授权模式，但本例重点仅演示说明密码模式授权流程。

## 3.1,密码模式授权流程

首先看授权认证服务流程图：

Identity Server

Authorization Server

Resource Owner

(Web API)

Client

1,Post: username,password

3,Create:Access Token

2,Validate:username,password

5,Validate: Access Token

6,Set:User Identity

4,Use: Access Token

7, Response :Resource

OAuth 2.0 :

Rresource owner password credentials

1. 用户在客户端输入自己的登录账号（用户名和密码），然后客户端将用户的登录账号信息POST到授权服务器；
2. 授权服务器携带用户的登录账号，去认证服务器验证用户的身份；
3. 验证通过，授权服务器为客户端生成一个访问令牌；
4. 客户端携带此访问令牌，访问资源服务器；
5. 资源服务器去授权服务器验证客户端的访问令牌是否有效；
6. 如果访问令牌有效，授权服务器给资源服务器发送用户标识信息；
7. 资源服务器根据用户标识信息，处理业务请求，最后发送响应结果给客户端。

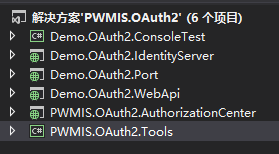
# 4,解决方案设计

## 4.1,程序集说明

根据OAuth 2.0授权体系中的角色，创建了对应的程序集项目。

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 角色 | 程序集名称 | 说明 |
| 1 | 授权服务器 | PWMIS.OAuth2.AuthorizationCenter | 授权中心  ASP.NET Web API+OWIN |
| 2 | 资源服务器 | Demo.OAuth2.WebApi | 提供API资源  ASP.NET Web API+OWIN |
| 3 | 客户端 | Demo.OAuth2.ConsoleTest | 控制台测试程序 |
| 4 |  | Demo.OAuth2.Port | 用户的Web入口  ASP.NET MVC 5.0 |
| 5 | 认证服务器 | Demo.OAuth2.IdentityServer | 简单登录账号认证  ASP.NET Web API |
| 6 |  | PWMIS.OAuth2.Tools | 提供OAuth2.0 协议访问的一些有用的工具类 |

解决方案截图：



## 4.2,授权服务器设计

### 4.2.1,客户端标识

在本授权体系中，所有客户端和资源服务器都必须有“客户端标识”，所有访问授权服务器的客户端都必须携带此客户端标识，这些表示存储在授权服务器的数据库中。因此，客户端标识信息被设计为一个具有持久化功能的实体类，本例采用SOD框架的实体类。

客户端标识信息（AuthClientInfoEntity）：

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 属性名称 | 类型 | 说明 |
| 1 | ClientId | String | 必须，客户端唯一标识号 |
| 2 | ClientSecret | String | 必须，客户端标识对应的密钥 |
| 3 | ClientHost | String | 可选，客户端的主机名地址 |
| 4 | RegDate | DateTime | 可选，拥有此客户端标识的对象的注册时间 |

### 4.2.2,用户账号

在本例中，只需要使用用户名和密码来标识一个唯一的用户，用此信息作为用户登录系统的账号。

用户账号信息（UserInfoEntity）：

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 属性名称 | 类型 | 说明 |
| 1 | ID | int | 可选，账户信息标识号 |
| 2 | UserName | String | 必须，登录的用户名 |
| 3 | Password | String | 必选，登录密码 |

### 4.2.3,登录结果

表示用户登录结果的简单对象，与用户账号不同的是，它不包含用户的登录密码，但是增加了登录成功后的用户角色信息登录失败的错误信息。

登录结果信息（LoginResultModel）：

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 属性名称 | 类型 | 说明 |
| 1 | ID | String | 必须，认证服务器返回的用户标识号 |
| 2 | UserName | String | 必须，认证服务器返回的登录用户名。如果认证失败，此项为空。 |
| 3 | Roles | String | 可选，授权服务器返回的该用户拥有的角色名称 |
| 4 | ErrorMessage | String | 可选，登录出错信息 |

### 4.2.4,身份认证模式

系统提供2种用户账号信息的认证模式：

#### 集成认证

采取此模式认证，等于是授权服务器与认证服务器集成在一起，本例的设计中，集成认证的实现采取的是通过数据库存储的用户账号来匹配客户端请求发来的用户账号。

要使用集成认证，首先需要在Web.config配置文件的appSettings 配置下面的内容：

<add key="IdentityLoginMode" value="**DataBase**"/>

然后，还需要配置IdentityRepository ，指定认证程序的持久化提供程序，当前系统默认使用SimpleIdentityRepository ：

<add key="IdentityRepository"

value="PWMIS.OAuth2.AuthorizationCenter.Repository.SimpleIdentityRepository,PWMIS.OAuth2.AuthorizationCenter"/>

这个配置指定了提供程序的名字是：

*PWMIS.OAuth2.AuthorizationCenter.Repository.SimpleIdentityRepository*

该提供程序所在的程序集名字是：

*PWMIS.OAuth2.AuthorizationCenter*

在授权服务器内部，IdentityRepositoryFactory 将根据此配置，获得认证服务持久化提供程序对象实例，访问数据库。

*注意：如果你需要实现自己的认证服务持久化对象，可以在你的提供程序中实现 IIdentityRepository 接口，然后修改IdentityRepository 的配置内容。*

最后，需要配置一个名字为 OAuth2的数据库连接信息：

<connectionStrings>

<!--OAuth2 此数据库配置了OAuth2资源服务器，客户端等访问授权服务器所必须的客户端标识和秘钥。-->

<add name ="OAuth2"

connectionString="Data Source=(LocalDB)\v11.0;AttachDbFilename=~\App\_Data\OAuth2DB.mdf;Integrated Security=True"

providerName="SqlServer"/>

</connectionStrings>

这个连接配置将使用SqlServer数据库存储授权认证等信息，当然也可以使用其它数据库种类，具体请参考[SOD框架](http://www.cnblogs.com/bluedoctor)的使用帮助。

#### 认证服务器

有时候，用户的身份认证是一个独立的系统，比如一个独立的管理系统它已经实现了用户身份认证的逻辑，或者采用了邮件身份认证，AD域身份认证等，我们可以将这些不同的认证方法统一封装成一个WebAPI服务对外提供，这些WebAPI运行在独立的认证服务器内。

认证过程：

授权服务器将客户端发来请求中的用户账号，转发到认证服务器去进行用户身份认证，由认证服务器返回认证结果给授权服务器，授权服务器再根据认证是否成功，是否授权客户端对资源服务器的访问。

使用认证服务器：

要使用认证服务器，首先需要在Web.config配置文件的appSettings 配置下面的内容：

<add key="IdentityLoginMode" value="**WebAPI**"/>

系统默认情况下采用认证服务器认证，所以下面这这种配置效果一样：

<add key="IdentityLoginMode" value=""/>

### 4.2.5,OWIN身份认证

在传统的ASP.NET Forms 身份认证中，不能灵活的设置身份认证内容，特别是不能解决跨域登录认证问题。ASP.NET Web API OWIN 根据OAuth2.0 协议实现了一套身份认证和授权功能，通过基于申明的身份认证方式，能够简化传统的认证过程，很好的解决这些问题。

在本例中，重写了OWIN的OAuth2.0授权提供程序的实现类：

public class OpenAuthorizationServerProvider : OAuthAuthorizationServerProvider

{

//...

}

有关OWin 的更多信息，请参考下列相关资料：

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

<http://www.cnblogs.com/xishuai/p/aspnet-webapi-owin-oauth2.html>

### 4.2.6,访问令牌

如果授权服务器“认证”通过了客户端的身份信息，就可以向客户端颁发访问令牌。不过，伴随该令牌的响应信息还包括刷新令牌的凭据等内容。

#### 令牌响应信息

在本系统中，令牌响应信息类型（TokenResponse）定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 属性名称 | 类型 | 说明 |
| 1 | AccessToken | String | 访问令牌，默认10秒有效 |
| 2 | RefreshToken | String | 刷新访问令牌的凭据 |
| 3 | TokenType | String | 令牌的类型 |
| 4 | expires\_in | String | 令牌有效期剩余的时间，单位秒。 |

#### 令牌有效期

访问令牌是很重要的信息，它代表对资源访问的授权许可，如果令牌泄露出去，第三方使用此令牌来访问系统可能造成严重后果，因此，理想情况下，令牌一旦使用过一次就需要作废，这样别人即使嗅探到此令牌然后也无法使用了。不过实际情况下由于网络的不稳定性无法确保令牌一次使用成功，因此合理的方案就是对令牌使用设置一个较短的有效期，在本例中，有效期是10秒。

注意：如果你需要调试程序，10秒显然太短，几个断点跟踪代码10秒就过去了，要修改此有效期，请看到如下代码：

public partial class Startup

{

public void ConfigureAuth(IAppBuilder app)

{

var OAuthOptions = new OAuthAuthorizationServerOptions

{

AllowInsecureHttp = true,

AuthenticationMode = AuthenticationMode.Active,

TokenEndpointPath = new PathString("/token"), //获取 access\_token 授权服务请求地址

AuthorizeEndpointPath = new PathString("/authorize"), //获取 authorization\_code 授权服务请求地址

AccessTokenExpireTimeSpan = TimeSpan.FromSeconds(10), //access\_token 过期时间

Provider = new OpenAuthorizationServerProvider(), //access\_token 相关授权服务

AuthorizationCodeProvider = new OpenAuthorizationCodeProvider(), //authorization\_code 授权服务

RefreshTokenProvider = new OpenRefreshTokenProvider() //refresh\_token 授权服务

};

app.UseOAuthBearerTokens(OAuthOptions); //表示 token\_type 使用 bearer 方式

}

public void Configuration(IAppBuilder app)

{

// For more information on how to configure your application, visit http://go.microsoft.com/fwlink/?LinkID=316888

ConfigureAuth(app);

var configuration = new HttpConfiguration();

WebApiConfig.Register(configuration);

app.UseWebApi(configuration);

}

}

上面的代码中，AccessTokenExpireTimeSpan = TimeSpan.FromSeconds(10) 设置的就是令牌有效期。

#### 申请令牌

根据OWIN 的Startup类的ConfigureAuth 方法，

OAuthAuthorizationServerOptions.TokenEndpointPath 定义了获取令牌的请求地址为

“/token”

使用HttpClient 调用申请令牌的方法代码示例：

httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(

"Basic",

Convert.ToBase64String(Encoding.ASCII.GetBytes(clientId + ":" + clientSecret)));

var cancelTokenSource = new CancellationTokenSource(10000);

var response = await httpClient.PostAsync("/token", new FormUrlEncodedContent(parameters), cancelTokenSource.Token);

var responseValue = await response.Content.ReadAsStringAsync();

具体请看PWMIS.OAuth2.Tools.OAuthClient.GetToken 的方法实现。

#### 刷新令牌

前面说过为了保证安全性，令牌的使用有效期很短，当下次要访问资源服务器令牌过期了怎么办？所以需要一个令牌刷新方法，而刷新令牌必须携带上传颁发的令牌中的“刷新凭据”。

例如如下代码：

var tokenResponseTwo =

oAuthCenterClient.GetToken("refresh\_token", tokenResponse.RefreshToken).Result；

在OAuthClient 类中，封装了刷新令牌的方法，具体请参看 OAuthClient. RefreshToken 的实现。

### 4.2.7,数据库

授权服务器需要访问数据库，它记录了客户端的标识信息等数据。

程序默认使用的是SQL Server 2012 LocalDB，它是一个非常轻量级的数据库，安装包大小只有33M多，如果你使用的是VS2013，默认已经安装；如果使用的是其它版本的VS开发工具或者要发布部署到服务器，需要下载安装，下载地址：

<https://www.microsoft.com/zh-cn/download/details.aspx?id=29062>

安装后，找到安装目录，运行命令：

C:\Program Files\Microsoft SQL Server\110\Tools\Binn>SqlLocalDb info

输出：

MSSQLLocalDB

Projects

v11.0

这里表示本机有3个SQLLoalDb的实例。

在本系统中，可以使用v11.0,它对应下面的连接字符串：

<add name ="OAuth2"

connectionString="Data Source=(LocalDB)\v11.0;AttachDbFilename=~\App\_Data\OAuth2DB.mdf;Integrated Security=True" providerName="SqlServer"/>

如果没有，可以执行下面的命令创建一个：

SqlLocalDb create "v11.0"

例如：

C:\Program Files\Microsoft SQL Server\110\Tools\Binn>SqlLocalDb create "v11.0"

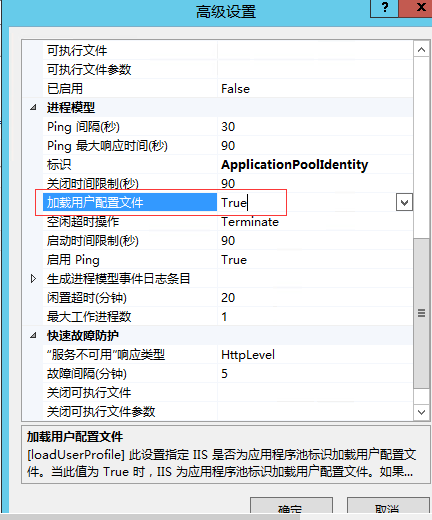
已使用版本 11.0 创建 LocalDB 实例“v11.0”。

经过这样的配置以后，Web.config文件中的数据库连接配置即可使用。

如果你安装了VS2017，可以使用第二个被注释掉的连接字符串。

注意：

Localdb要配合iis使用,在IIS的应用程序池的高级设置里，“加载用户配置”=true



## 4.3,资源服务器设计

### 4.3.1,WebAPI资源服务器

ASP.NET Web API +OWIN 可以很方便的验证客户端发送的访问令牌，只需重写 DelegatingHandler 对象，获得客户端发生的令牌，然后向授权服务器检查此令牌的有效性。例如这里重写此对象为

public class AuthenticationHandler : DelegatingHandler

{

protected override System.Threading.Tasks.Task<HttpResponseMessage> SendAsync

(HttpRequestMessage request,

System.Threading.CancellationToken cancellationToken)

{

if (request.Headers.Authorization != null &&

request.Headers.Authorization.Parameter != null)

{

string token = request.Headers.Authorization.Parameter;

//... 其它代码略

}

}

//其它代码略

}

然后，将这个令牌添加到HttpClient对象的请求头里面，

\_httpClient.DefaultRequestHeaders.Authorization =

new AuthenticationHeaderValue("Bearer", token);

如果授权服务器检查此token有效，那么就可以访问授权服务器获得其它信息。

最后，将AuthenticationHandler 添加到OWIN的起始方法中：

public class Startup

{

public void Configuration(IAppBuilder app)

{

HttpConfiguration config = new HttpConfiguration();

ConfigureOAuth(app);

WebApiConfig.Register(config);

config.MessageHandlers.Add(new **AuthenticationHandler**());

app.UseCors(Microsoft.Owin.Cors.CorsOptions.AllowAll);

app.UseWebApi(config);

}

}

### 4.3.2,其它资源服务器

有时候，我们的资源服务器并不是ASP.NET WebAPI的，可能是比较老旧的ASP.NET MVC 3.0或者ASP.NET WebForms，甚至是Java，PHP的一个Web站点，这些资源服务器无法使用OWIN，所以这里统一称为“其它资源服务器”。

对这类资源服务器，如何获取它的访问令牌呢？

实际上，Token 信息就在HTTP请求头的Authorization 里面，使用下面的代码即可获取到：

//HttpRequestBase request 或者 HttpRequest request

if (request.Headers.AllKeys.Contains("Authorization"))

{

var headValue = request.Headers["Authorization"];

string[] headValueArr = headValue.Split(' ');

string authType = headValueArr[0];

string token = headValueArr[1];

//然后向授权服务器验证此token

}

这段代码可以放到要访问的资源服务器特定的API请求方法内，也可以放到站点相应的拦截器里面。获取到客户端发来的Token后，向授权服务器验证，如果验证通过，资源服务器授予此次访问许可即可。

### 4.3.3,访问资源服务器

访问资源服务器之前，必须先获取访问令牌，请看 【授权服务器设计.访问令牌.申请令牌】的内容。获取到访问令牌后，可将它存储到当前会话或者C/S客户机的进程内存中，下次要访问资源服务器的时候，刷新下令牌，然后再用新令牌去访问。

具体过程如下示例代码：

[Authorize]

public async Task<ActionResult> Biz()

{

OAuthClient oc = new OAuthClient();

TokenResponse token = Session["token"] as TokenResponse;

if (token == null)

return Redirect("/Logon");

var resourceClient = await oc.GetResourceClient(token);

Session["token"] = oc.CurrentToken;

var responseTwo = await resourceClient.GetAsync("/api/values");

if (responseTwo.StatusCode != HttpStatusCode.OK)

{

ViewBag.Message = "访问WebAPI 失败。";

}

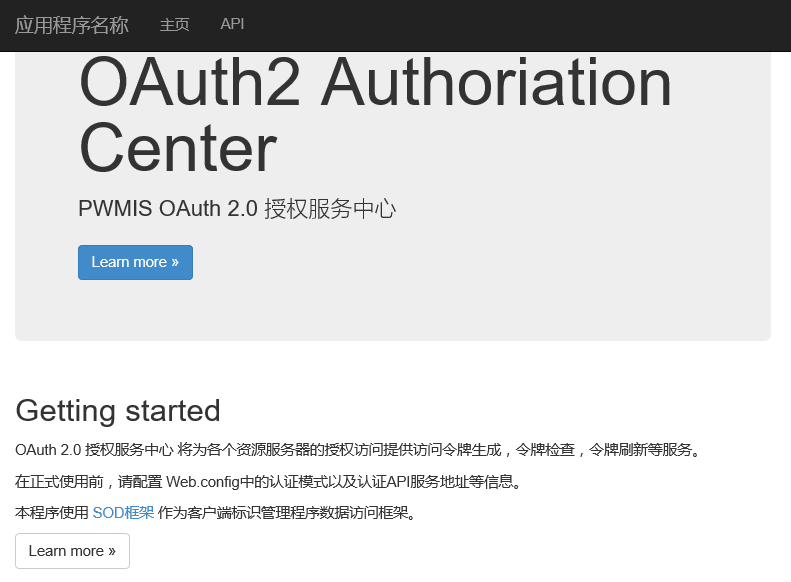
ViewBag.Message = await responseTwo.Content.ReadAsStringAsync();

return View();

}

# 5,DEMO运行效果

## 5.1,授权服务器

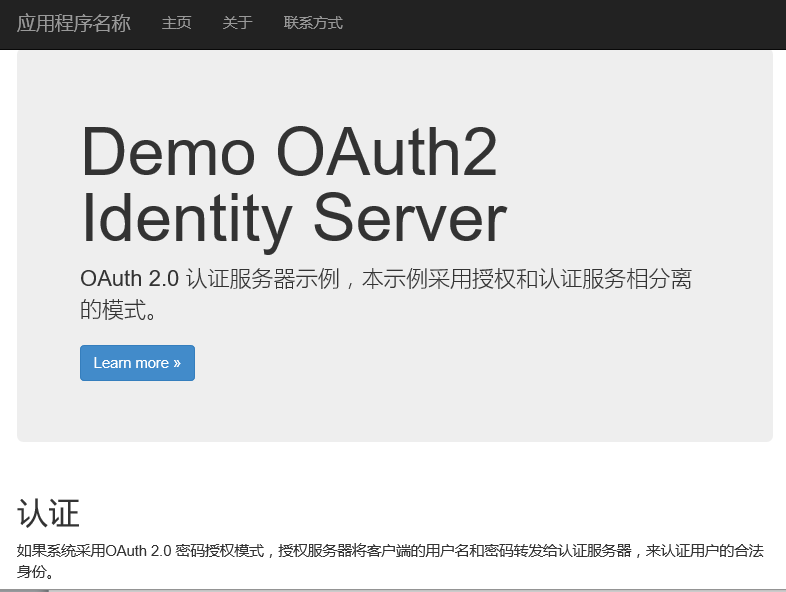




## 5.2,资源服务器



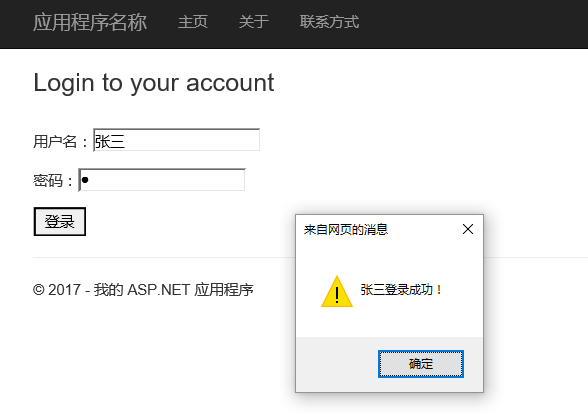
## 5.3,认证服务器



## 5.4,用户入口站点



点击“业务操作页面”的“操作”按钮，第一次，将导航到登录页：



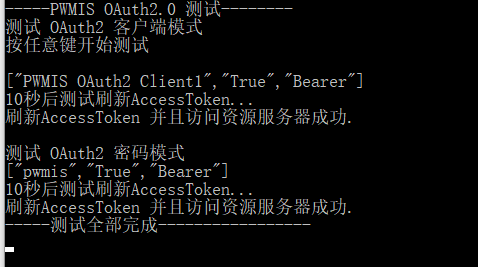
由于采用的是模拟的认证服务器，它假设任何用户都是合法的，所以这里返回登录成功。



之后，进入到“业务操作页”。回到主页，重新进入“业务操作页”，不再需要登录。

此页面的内容，来自于资源服务器。

## 5.5,测试控制台



程序将测试客户端模式和密码模式，并且测试刷新访问令牌的功能。

# 6,其它

## 6.1,展望

基于OAuth 2.0 的授权认证方案可以适用于ASP.NET MVC ,ASP.NET Web API 以及其它客户端程序。本次DEMO解决方案的Port项目，展示了通过Web后端授权访问资源服务器的能力，这为前后端分离的Web开发模式，主站代理访问后端多个API的方式提供了可能性。在此基础上，还可以实现不依赖于域名的多站点单点登录。

## 6.2,参考资源

本示例源码地址：

在线文档：

参考资源：

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

<http://www.cnblogs.com/Hai--D/p/6187051.html>

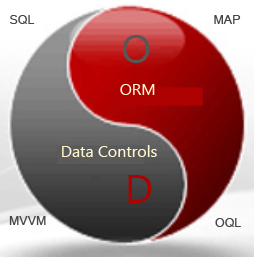
<http://www.cnblogs.com/xishuai/p/aspnet-webapi-owin-oauth2.html>

<http://www.07net01.com/2016/12/1753733.html>

<http://www.cnblogs.com/zhyp/p/5513355.html>

## 6.3,技术支持

SOD开发团队：

[](http://www.pwmis.com/sqlmap/index.htm)

<http://www.pwmis.com/sqlmap>

2017年7月19日