腾飞（Jesse）

MVC5 - ASP.NET Identity登录原理 - Claims-based认证和OWIN

在Membership系列的最后一篇引入了ASP.NET Identity，看到大家对它还是挺感兴趣的，于是来一篇详解登录原理的文章。本文会涉及到Claims-based（基于声明）的认证，我们会详细介绍什么是Claims-based认证，它与传统认证方式的区别，以及它的特点。同时我们还会介绍OWIN (Open Web Interface for .NET) 它主要定义了Web Server 和Web Application之间的一些行为，然后实现这两个组件的解耦（当然远不止这么点东西，我相信OWIN马上就会掀起一场血雨腥风）ASP.NET Identity是如何利用OWin实现登录的，都是干货，同学，你准备好学习了么？

目录

ASP.NET Identity登录原理

废话少说，我们直接切入正题。在上一篇从Membership到ASP.NET Identity，我们已经给了一个简单的实例，并且大致的描述了一下ASP.NET Identity的结构体系，但是ASP.NET Identity主要提供的功能是帮助我们管理用户，角色等信息，它主要负责的是存储这一块，也就是我们的信息存到哪里去的问题。但是用户是如何**实现登录**的？是**Forms认证**么？**用到Cookie了么**？**Cookie里面有保存明文信息**么（咳咳，最近某程旅游网好像很火？），接下来我们就来一一的回答这些问题。

在上一篇的例子中，我们可以简单的发现，要实现登录实际上只有简单的三行代码

```
private IAuthenticationManager AuthenticationManager
{
get { return HttpContext.GetOwinContext().Authentication; }
}

private async Task SignInAsync()
{
// 1. 利用ASP.NET Identity获取用户对象
var user = await UserManager.FindAsync("UserName", "Password");
// 2. 利用ASP.NET Identity获取identity 对象
var identity = await UserManager.CreateIdentityAsync(user, DefaultAuthenticationTypes.ApplicationC
// 3. 将上面拿到的identity对象登录
```

公告

昵称：腾飞（Jesse)
园龄：3年6个月
荣誉：推荐博客
粉丝：3719
关注：15
+加关注

随笔分类(50)

ASP.NET 安全系列(4)
C# 揭密(11)
C#移动跨平台开发(2)
Javascript 原理(4)
Owin系列(2)
Web(15)
测试(1)
程序人生(5)
分布式架构(4)
领域驱动(2)

最新评论

1. Re:暴力英语学习法 + 严格的目标管理 = 成功快速靠谱的学好英语
谢谢博主的分享，很有计划性的学习，以你为榜样！能否传一份相关的学习资料，780494008@qq.com，谢谢！
--xuweitaichi

2. Re:一不小心写了个WEB服务器
不赖。
--肖邦linux

3. Re:暴力英语学习法 + 严格的目标管理 = 成功快速靠谱的学好英语
1023870642@qq.com 博主我正在学英语求教学资源谢谢
--林夕朱娘金

4. Re:MVC5 - ASP.NET Identity登录原理 - Claims-based认证和OWIN
马克一个
--春華秋實

5. Re:暴力英语学习法 + 严格的目标管理 = 成功快速靠谱的学好英语
学习英语，学习编程，求楼主暴力英语资源~多谢！
1426032448@qq.com
--A001小峰

6. Re:MVC5 - ASP.NET Identity登录原理 - Claims-based认证和OWIN
手滑，点错了，实在不好意思
--Xi_SHENG

7. Re:MVC5 - ASP.NET Identity登录原理 - Claims-based认证和OWIN
最新版的.net core里又改了很多，不过就大多是dll名称换了
--岁月已走远

```
13  AuthenticationManager.SignIn(new AuthenticationProperties() { IsPersistent = true }, identity);
14  }
```

我们发现UserManager.CreateIdentityAsync返回给我们的对象是一个ClaimsIdentity,这又是一个什么玩意？它和我们原来所熟知的Identity对象有什么关联么？毕竟长的那么像，在深入之前，我们先要了解一下下面的概念。

什么是Claims-based（基于声明）的认证

首先这个玩意不是微软特有的，Claims-based认证和授权在国外被广泛使用，包括微软的ADFS，Google，Facebook等。 国内我就不知道了，没有使用过国内的第三方登录，有集成过QQ登录或者支付宝登录的同学可以解释一下。

Claims-based认证主要解决的问题?

对比我们传统的Windows认证和Forms认证，claims-based认证这种方式将认证和授权与登录代码分开，将认证和授权拆分成另外的web服务。活生生的例子就是我们的qq集成登录，未必qq集成登录采用的是claims-based认证这种模式，但是这种场景，千真万确就非常适合claims-based认证。

Claims-based认证的主要特点：

- 将认证与授权拆分成独立的服务
- 服务调用者(一般是网站)，不需要关注你如何去认证，你用Windows认证也好，用令牌手机短信也好，与我无关。
- 如果用户成功登录的话，认证服务(假如是QQ) 会返回给我们一个令牌。
- 令牌当中包含了服务调用者所需要的信息，用户名，以及角色信息等等。

总的来说就是，我再也不用管你怎么登录，怎么样去拿你有哪些角色了，我只需要把你跳到那个登录站点上，然后它返回给我令牌信息，我从令牌上获取需要的信息来确定你是谁，你拥有什么角色就可以了。

进一步理解Claims-based 认证

为了让大家进一步理解Claims-based认证，我们从一个普通的登录场景开始说起，拿QQ集成登录来举例。

1. 用户跑到我们的网站来访问一个需要登录的页面
2. 我们的网站检测到用户没有登录，返回一个跳转到QQ登录页的响应（302 指向QQ登录页面的地址并加上一个返回的链接页面，通常是returnUrl=）
3. 用户被跳转到指定QQ的登录页面
4. 用户在QQ登录页面上输入用户名和密码，QQ会到自己的数据库中查询，一旦登录成功，会返回一个跳转到我们站点的响应（302指向我们的网站页面）
5. 用户被跳转到我们网站的一个检测登录的页面，我们可以拿到用户的身份信息，建立ClaimsPrinpical和ClaimsIdentity对象，生成cookie等。
6. 我们再把用户带到指定的页面，也就是returnUrl，那是用户登录前最后一次访问的页面

8. Re:暴力英语学习法 + 严格的目标管理 = 成功快速靠谱的学好英语
学习英语，学习编程，求楼主资源~多谢！836171900@qq.com
--Cici_淘

9. Re:Membership三步曲之入门篇 - Membership基础示例
@T林飘叶在.net core的发展历程中，有一段时间asp.net里的Web.config文件被抛弃了，后来又加回来了。...
--岁月已走远

10. Re:从Membership 到 .NET4.5 之 ASP.NET Identity
@小鸟哥哥 可以的呀，在微软官方有示例，不过不是用的QQ号，是用得MS账户、Google账户、Facebook账户，我照着做了下，确实可以，就是OAuth应用。...
--岁月已走远

11. Re:从Membership 到 .NET4.5 之 ASP.NET Identity
讲得很好，不过提醒观众一点：最新的asp.net mvc core里已经通过依赖注入来完成AccountController中UserManager的初始了，下面这段代码看不到了。new UserMa.....
--岁月已走远

12. Re:暴力英语学习法 + 严格的目标管理 = 成功快速靠谱的学好英语
躺枪了啊，一直想学好英语断断续续没学好，905044086@qq.com能发我一份资料吗谢谢，我看要资料的圆友很多，大神可以放到百度网盘里啊
--薛陈磊

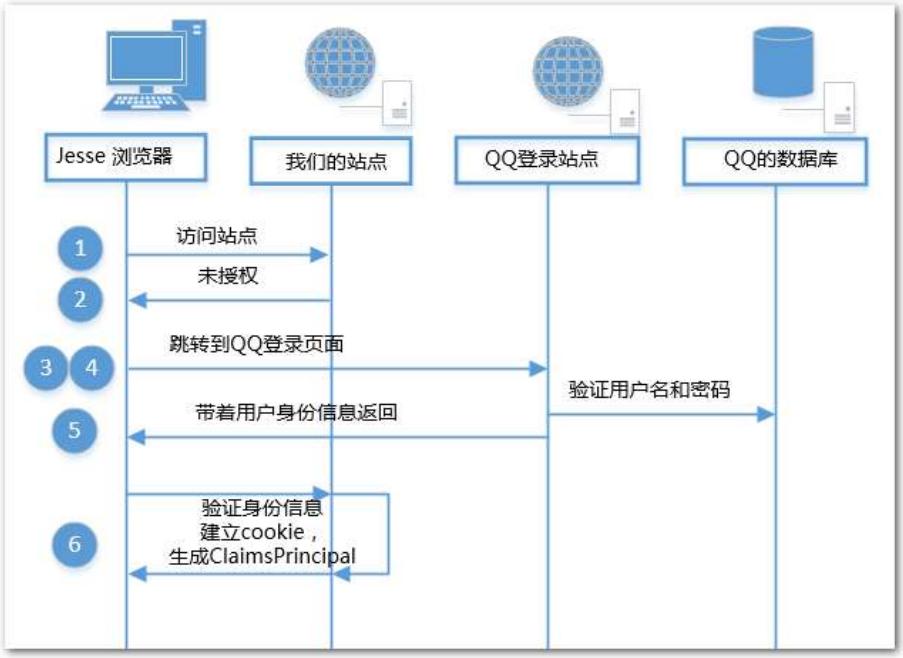
13. Re:Windows平台分布式架构实践 - 负载均衡（下）
@ Jesse Liu 我的系统是windows server 2008 R2 的 安装了所有的组件之后 我的web farm 只有三个图标 能否发一下安装包给我呢！急用 37360533@qq.....
--雷雷LL

14. Re:暴力英语学习法 + 严格的目标管理 = 成功快速靠谱的学好英语
1025724128@qq.com 跪求博主可以发我一份。
--白云苍狗喂

15. Re:暴力英语学习法 + 严格的目标管理 = 成功快速靠谱的学好英语
英语一直是我的痛啊，求大神分享一份 nanwanwang@hotmail.com
--wanwanwang

阅读排行榜

1. 暴力英语学习法 + 严格的目标管理 = 成功快速靠谱的学好英语(72225)
2. MVC5 - ASP.NET Identity登录原理 - Claims-based认证和OWIN(45147)
3. async & await 的前世今生（Updated）(34666)
4. 从Membership 到 .NET4.5 之 ASP.NET Identity(27993)
5. bootstrap + requireJS+ director+ knockout + web API = 一个时髦的单页程序(27430)
6. Windows平台分布式架构实践 - 负载均衡(26346)
7. 一不小心写了个WEB服务器(23782)
8. 异步编程 In .NET(23072)
9. Windows平台分布式架构实践 - 负载均衡（下）(14611)



简单的来说，就是把登录的代码（验证用户，获取用户信息）拆分成独立的服务或组件。

ASP.NET 下的 Claims-based认证实现

说完什么是Claims-based认证之后，我们接下来就可以看看ClaimsIdentity以及ClaimsPrincipal这两个类，他们是.NET下Claims-based认证的主要基石。当然正如我们所想，他们继承了接口IIdentity和IPrincipal。

IIdentity封装用户信息

这个接口很简单，它只包含了三个最基本的用户身份信息。

```
public interface IIdentity
{
    string AuthenticationType { get; }
    bool IsAuthenticated { get; }
    string Name { get; }
}
```

IPrincipal 代表着一个安全上下文

这个安全上下文对象包含了上面的identity以及一些角色和组的信息,每一个线程都会关联一个Principal的对象，但是这个对象是属性进程或者AppDomain级别的。ASP.NET自带的 RoleProvider就是基于这个对象来实现的。

CalimsIdentity和ClaimsPrincipal

在System.Security.Claims命名空间下去，我们可以发现这两个对象。下面我们来做一个小例子，这个小例子会告诉我们这两个对象是如何进行认证和授权的。我们要做的demo很简单，建一个空的mvc站点，然后加上一个HomeController，和两个Action。并且给这个HomeController打上Authroize的标签，但是注意我们没有任何登录的代码，只有这个什么也没有的Controller和两个什么也没有的Action。

```
[Authorize]
public class HomeController : Controller
{
    [Authorize(Roles = "Users")]
    public ActionResult Index()
    {
        return View();
    }

    [Authorize(Roles = "Managers")]
    public ActionResult Manager()
    {
        return View();
    }
}
```

当然，结果也是可想而知的，我们得到了401页面，因为我们没有登录。

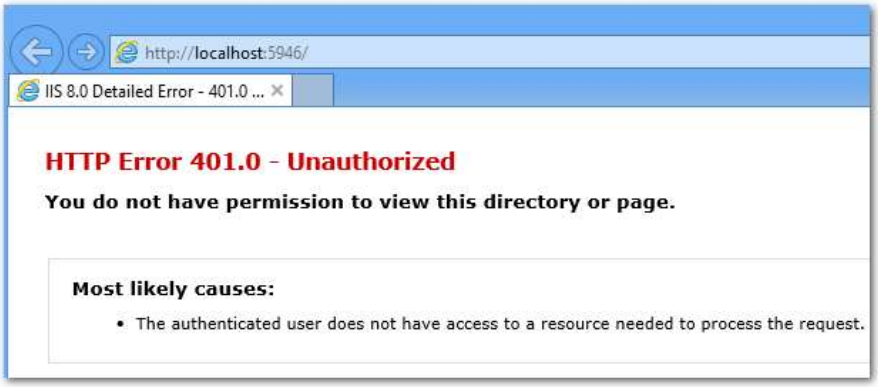
10. Membership三步曲之入门篇 - Membership基础示例(14369)
11. 初探领域驱动设计（1）为复杂业务而生(13548)
12. 我想和大家说说心里话(13417)
13. 由浅入深表达式树（一）创建表达式树(13264)
14. 背后的故事之 - 快乐的Lambda表达式（一）(13220)
15. 由浅入深表达式树（完结篇）重磅打造Linq To 博客园(10542)

评论排行榜

1. 暴力英语学习法 + 严格的目标管理 = 成功快速靠谱的学好英语(1248)
2. Windows平台分布式架构实践 - 负载均衡(163)
3. 异步编程 In .NET(155)
4. 初探领域驱动设计（1）为复杂业务而生(142)
5. async & await 的前世今生（Updated）(131)
6. 我想和大家说说心里话(111)
7. MVC5 - ASP.NET Identity登录原理 - Claims-based认证和OWIN(101)
8. Windows平台分布式架构实践 - 负载均衡（下）(86)
9. 一不小心写了个WEB服务器(80)
10. 背后的故事之 - 快乐的Lambda表达式（一）(65)
11. 从Membership 到 .NET4.5 之 ASP.NET Identity(64)
12. bootstrap + requireJS+ director+ knockout + web API = 一个时髦的单页程序(60)
13. 上一家公司倒闭，为什么我又来了创业公司？(57)
14. ASP.NET安全(56)
15. C#移动跨平台开发（1）环境准备(55)

推荐排行榜

1. 暴力英语学习法 + 严格的目标管理 = 成功快速靠谱的学好英语(1359)
2. 异步编程 In .NET(414)
3. async & await 的前世今生（Updated）(317)
4. Windows平台分布式架构实践 - 负载均衡(308)
5. MVC5 - ASP.NET Identity登录原理 - Claims-based认证和OWIN(270)
6. 一不小心写了个WEB服务器(211)
7. 背后的故事之 - 快乐的Lambda表达式（一）(177)
8. 从Membership 到 .NET4.5 之 ASP.NET Identity(161)
9. Membership三步曲之入门篇 - Membership基础示例(155)
10. bootstrap + requireJS+ director+ knockout + web API = 一个时髦的单页程序(150)
11. Windows平台分布式架构实践 - 负载均衡（下）(139)
12. 我想和大家说说心里话(136)
13. 谈谈我心目中理想的牛人(118)
14. 初探领域驱动设计（1）为复杂业务而生(118)
15. 由浅入深表达式树（一）创建表达式树(103)

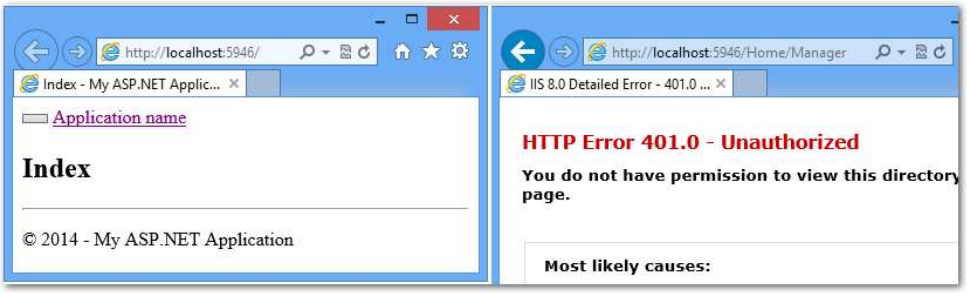


下面我们就来实现登录，这里的登录非常简单，我们手动去创建这个ClaimsIdentity和ClaimsPrincipal对象，然后将Principal对象指给当前的HttpContext.Current.User。

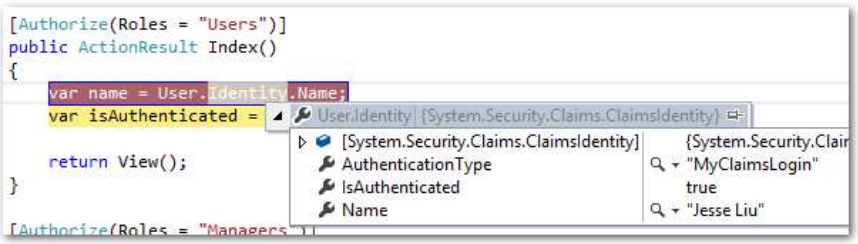
```
protected void Application_AuthenticateRequest()
{
    var claims = new List<Claim>();
    claims.Add(new Claim(ClaimTypes.Name, "Jesse Liu"));
    claims.Add(new Claim(ClaimTypes.Role, "Users"));
    var identity = new ClaimsIdentity(claims, "MyClaimsLogin");

    ClaimsPrincipal principal = new ClaimsPrincipal(identity);
    HttpContext.Current.User = principal;
}
```

我们在Global.asax中添加了Application_AuthenticateRequest方法，也就是每次MVC要对用户进行认证的时候都会进到我们这个方法里面，然后我们就这样神奇的把用户给登录了。



当然，我们没有Home/Manager的访问权限，因为我们上面只给了用户Users的Role。



现在大家知道ClaimsIdentity和ClaimsPrincipal是如何使用了么？这里要**注意**一下的是，我们没有设置IsAutheiticated为true，在.NET4.5以前，对于GenericIdentity只要设置它的Name的时候IsAutheiticated就自动设置为true了，而对于ClaimsIdentity是在它有了第一个Claim的时候。在.NET4.5以后，我们就可以灵活控制了，默认ClaimsIdentity的IsAutheiticated是false，只有当我们构造函数中指定Authentication Type，它才为true。

最后结论，我们讲了ClaimsIdentity什么的，讲了这么多和今天的主题有嘛关系？我们上面说ASPNET Identity登录有三句话，第一句话可以略过，第二句话就是我们上面讲的。

```
1 var identity = await UserManager.CreateIdentityAsync(user, DefaultAuthenticationTypes.ApplicationCo
```

UserManager实际上只是为我们创建了一个ClaimsIdentity的对象，还是通过我们自己从数据库里面取出来的对象来创建的，它也就干了那么点事，一层小小的封装而已。不要被后面的DefaultAuthenticationTypes.ApplicationCookie吓到了，这里还没有和cookie扯上半点关系，这就是一个字符串常量，和我们上面自己定义的MyClaimsLogin是没有区别的。

到这里，我想算是把登录代码的第二句话讲完了，讲清楚了，那么我们来看看第三句话，也就是最后一句，其实它才是登录的核心，第二句只是创建了一个ClaimsIdentity的对象。

```
1 private IAuthenticationManager AuthenticationManager
```

```
{
get { return HttpContext.GetOwinContext().Authentication; }
}
AuthenticationManager.SignIn(new AuthenticationProperties() { IsPersistent = true }, identity);
```

通过F12查看，发现IAuthenticationManager 在 Microsoft.Owin.Security命名空间下，而这个接口是定义在Microsoft.OWin.dll中的。这又是个什么玩意儿？带着这个疑问，我开始了我的OWin学习之旅。

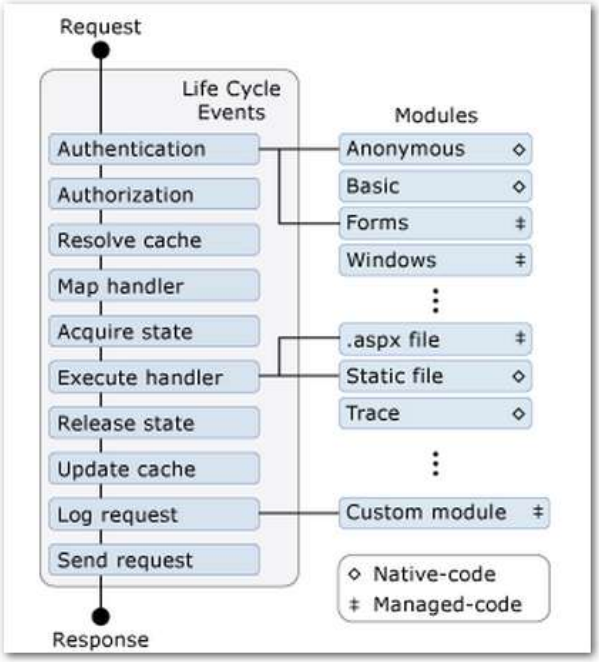
到底什么是OWIN

首先我们来简单介绍一下OWin，它是由微软ASP.NET小组成员组织成立的一个开源项目。目标是解耦服务器和应用，这里面的服务器主要是指web 服务器，比如说IIS等，全称是Open Web Interface for .Net。OWin可以说是一套定义，默认它是没有什么具体的实现的，那么在它的定义里面是如何实现服务器与应用程序的解耦的呢？ 我们又该如何理解服务器与应用程序的解耦呢？

下面是个人的理解，抛砖引玉，希望大家多探讨。

问题引入： 为什么要解耦服务器与应用程序 ？

既然是服务器和应用程序的解耦，那么这肯定是我们第一个应该考虑的问题。我们先来简单复习一下ASP.NET 或者是IIS 集成模式管道模型，也就是说一个http请求在进入IIS之后（我们这里指7.0及以后版本的集成模式），一直到返回response这中间所经历的步骤。



大家知道，我们可以开发自己的Http Module去注册这些事件，然后做相应的处理。比如说FormsAuthenticationModule就是注册了AuthenticateRequest事件，然后在这里面去检查用户的cookie信息来判断用户是否登录的，这里就是一个典型的**应用程序与服务器之间的交互**问题。而这些事件最后是被IIS触发的，我们是通过web.config把我们自定义的http module注册进了iis。回到我们的问题，如果**我们的网站不运行在iis了**，我们自己开发的这些Http module还能使用么？

另外的问题就是，大家知道我们在ASP.NET 里面经常用到HttpContext，HttpApplicationt等对象，而ASP.NET所有的处理基本上都离不开这两个对象，因为我们的Request以及Response都是封装在HttpContext里面的，而这些信息是从IIS中来，最后也是交给IIS处理，因为微软给IIS写代码的时候直接集成了这一块，但是想一下，如果web服务器不是IIS，那么这些信息又从哪里获取呢？

为什么需要解耦，是因为他们彼此之间的依懒过大，从而导致我们不能够轻易的换掉其中任何一个。 即使现在，在web.config添加自己定义的http module 也不是一件能让人开心的事情，反正我一想到那个很长的类名以及程序集名就够蛋疼的。

显然，很多人已经开始意识到，在如今web飞快发展的年代，这种模式已经不能够满足灵活多变的需求。越是轻量级，组件化的东西，越能够快速适应变化，更何况.NET现在要吸引开源社区的注意，只有把这一块打通了，越来越多的强大的开源组件才能够出现在.NET的世界里，比如说写一个开源的ASP.NET web服务器。

OWin如何做到解耦

我们上面说Owin是一套定义，它通过将服务器与应用程序之间的交互归纳为一个方法签名，称之为“应用程序代理（application delegate）”

```
AppFunc = Func<IDictionary<string, object>, Task>;
```

在一个基于Owin的应用程序中的每一个组件都可以通过这样的一个代理来与服务器进行交互。 这们这里的交互其实是与服务器一起来处理http request，比如说ASP.NET管理模型中的那些事件，认证，授权，缓存等等，原先我们是通过自定义的http module，在里面拿到包含了request和response的HttpContext对象，进行处理。而现在我们能拿到的就是一个Dictionary。

可是别小看了这个Dictionary，我们所有的信息比如Application state, request state，server state等等这些信息全部存在这个数据结构中。这个dictionary会在Owin处理request的管道中进行传递，没错有了OWin之后，我们就不再是与ASP.NET 管道打交道了，而是OWin的管道，但是这个管道相对于ASP.NET 管道而言更灵活，更开放。

这个字典在OWin管道的各个组件中传输时，你可以任意的往里面添加或更改数据。 OWin默认为我们定义了以下的数据：

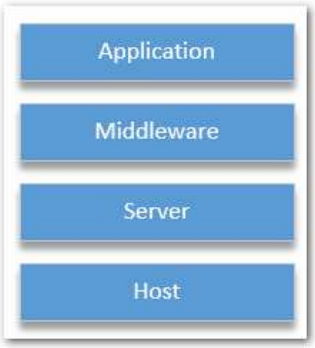
Key Name	类型	描述
"owin.RequestBody"	Stream	http 请求体 request body
"owin.RequestHeaders"	IDictionary<string, string[]>	http 请求头
"owin.RequestMethod"	String	请求方法，get, post 等
"owin.RequestPath"	String	URL 的路径，根后面的部分 /home/index
"owin.RequestPathBase"	String	URL 根，比如 www.baidu.com
"owin.RequestProtocol"	String	协议名称和版本 http/1.0, http/1.1
"owin.RequestQueryString"	String	URL 问号后面的参数
"owin.RequestScheme"	string	http 或者 https

有了这些数据以后，我们就不需要和.NET的那些对象打交道了，比如说ASP.NET MVC中的HttpContextBase, 以及WEB API 中的HttpRequestMessage和HttpResponseMessage。我们也不需要再考虑system.web 这个dll里的东西，我们只需要通过OWin就可以拿到我们想要的信息，做我们想做的事了。而OWin，它本身和web服务器或者IIS没有任何关系。

微软对OWin的开源实现Katana

我们上面讲到了OWin只是一套定义，它本身没有任何代码，我们可以把它看成是微软对外公开的一套标准。那么我们用到的Microsoft.OWin，这些dll又是从哪里来的呢？ 好消息是它是开源的，代码我们可以从CodePlex上下载，坏消息是它现在还没有比较全的文档，可能是我暂时还没有找到。

它包括下面4个组件：



- Host: 托管我们应用程序的进程，或者宿主，可以是IIS，可以我们自己写的程序等。主要是用来启动，加载OWin组件，以及合理的关闭他们
- Server: 这个Server就是用来暴露TCP端口，维护我们上面讲到的那个字典数据，然后通过OWin管理处理http请求
- Middleware : 这个中间件就是用来在OWin管道中处理请求的组件，你可以把它想象成一个自定义的httpModule，它会被注册到OWin管道中一起处理http request
- Application: 这个最好理解，就我们自己开发的那个应用程序或者说是网站

也就是说我们用到的Microsoft.OWin，那一系列的dll实现上是叫Katana(武士刀）象征着快，狠，准！下面来一些名词解释，是一些简单的概念有助于大家理解我们下面要讲的内容（ASP.NET Identity是如何借助 OWin来实现登录的）。

OWin Application(OWin 应用程序)

这个程序引入了OWin的dll，同时会使用OWin中的一些组件完成对request的一些处理，比如说我们下面要讲的OWin 认证。

OWin 组件

我们也可能管它叫中间件，它通过暴露一个应用程序代理，也就是接收一个IDictionary<string,object>，返回一个Task来参与到OWin对request和处理管道中。

Start up 类

每一个OWin的应用程序都需要有一个start up的类，用来声明我们要使用的OWin组件（即中间件)。Start up 类有以下几种声明方式：

1. 命名约定: Owin会扫描在程序集的根下名叫 startup的类作为默认启动配置类
2. OwinStartup 标签

```
[assembly: OwinStartup(typeof(StartupDemo.TestStartup))]
```

3. config 文件

```
<add key="owin:AutomaticAppStartup " value="false" />
```

OWin authentication

Owin的很大亮点之一就是它可以让我们的ASP.NET 网站摆脱IIS，但是毕竟大多数的ASP.NET 网站还是host在IIS上的，所以Katana项目还支持在IIS集成模式中运行Owin组件。 我们只需要在我们的项目中加上Microsoft.Owin.Host.SystemWeb这个包就可以了，其实默认MVC5程序已经为我们加上了。我们在VS2013中新建一个MVC5的站点，默认会为我们加上以下的dll:

- OWin.dll
- Microsoft.Owin.dll
- Microsoft.Owin.Host.SystemWeb
- Microsoft.Owin.Security
- Microsoft.Owin.Security.Cookie

他们对应nuget中的package:

```
<package id="Owin" version="1.0" targetFramework="net451" />
<package id="Microsoft.Owin" version="2.0.0" targetFramework="net451" />
<package id="Microsoft.Owin.Host.SystemWeb" version="2.0.0" targetFramework="net451" />
<package id="Microsoft.Owin.Security" version="2.0.0" targetFramework="net451" />
<package id="Microsoft.Owin.Security.Cookies" version="2.0.0" targetFramework="net451" />
```

这就是为什么我们可以拿到Microsoft.Owin.Security.IAuthenticationManager，然后再调用其 SignIn方法和SignOut方法。除了多了这些dll以外，VS还自动帮我们移除了FormsAuthenticationModule。

```
<system.webServer>
  <modules>
    <remove name="FormsAuthenticationModule" />
  </modules>
</system.webServer>
```

Forms 认证

我们来小小的复杂一下Forms认证，在Forms认证中我们检测完用户名和密码之后，只需要调用下面的代码就会为我们创建用户cookie。

```
FormsAuthentication.SetAuthCookie("Jesse", false);
```

然后FormsAuthenticateionModule会在ASP.NET 管道的 AuthenticateRequest 阶段去检查是否有这个cookie，并把它转换成我们需要的identity对象，这样的话我们就不需要每一次都让用户去输入用户名和密码了。所以登录的过程实现上是这样的。

1. 用户在没有登录的情况下访问了我们需要登录的页面
2. FormsAuthenticationModule检查不到用户身份的cookie，没有生成identity对象，HttpContext.User.IsAuthenticated = false
3. 在ASP.NET 管道 的Authroize 授权阶段，将用户跳转到登录页面

4. 用户输入用户名和密码点击提交
5. 我们检查用户名和密码，如果正确，就调用FormsAuthentication.SetAuthCookie方法生成登录cookie
6. 用户可以正常访问我们需要登录的页面了
7. 用户再次访问我们需要登录的页面
8. FormsAuthenticationModule检查到了用户身份的cookie，并生成identity对象，
   HttpContext.User.IsAuthenticated = true
9. ASP.NET 管道的 Authroize授权阶段，HttpContext.User.IsAuthenticated=true，可以正常浏览
10. 7,8,9 循环

Forms认证有以下几不足：

1. 用户名直接暴露在cookie中，需要额外的手段去将cookie加密
2. 不支持claims-based 认证
3. ....

我们上面Forms的登录过程，对于OWin登录来说同样适用。我们在上面讲ASP.NET Identity登录第二句话的时候已经拿到了ClaimsIdentity，那么我们接下来要看的问题就是如何借助于IAuthenticationManager 去登录？FormsAuthenticationModuel没有了，**谁来负责检测cookie**？您请接着往下看！

MVC 5默认的start up配置类

VS除了为我们引用OWin相关dll，以及移除FormsAuthenticationModule以外，还为我们在App_Start文件夹里添加了一个Startup.Auth.cs的文件。

```
public partial class Startup
{
    public void ConfigureAuth(IAppBuilder app)
    {
        // 配置Middleware 組件
        app.UseCookieAuthentication(new CookieAuthenticationOptions
        {
            AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
            LoginPath = new PathString("/Account/Login"),
            CookieSecure = CookieSecureOption.Never,
        });
    }
}
```

UseCookieAuthentication是一IAppBuilder 的一个扩展方法，定义在Microsoft.Owin.Security.Cookies.dll中。

CookieAuthenticationExtensions.cs

```
public static IAppBuilder UseCookieAuthentication(this IAppBuilder app,
    CookieAuthenticationOptions options)
{
    if (app == null)
    {
        throw new ArgumentNullException("app");
    }
    app.Use(typeof(CookieAuthenticationMiddleware), app, options);
    app.UseStageMarker(PipelineStage.Authenticate);
    return app;
}
```


将Owin Middleware绑定到IIS 集成模式的管道

UseCookieAuthentication主要调用了两个方法:

- IAppBuilder.Use ： 添加OWin middleware 组件到 OWin 管道
- IAppBuilder.UseStageMarker ： 为前面添加的middleware指定在IIS 管道的哪个阶段执行。

PepelineStage这个枚举定义和我们IIS管道的那些顺序，也就是和我们Http Module里面可以绑定的那些事件是一样的。

```
public enum PipelineStage
{
    Authenticate = 0,
    PostAuthenticate = 1,
    Authorize = 2,
    PostAuthorize = 3,
    ResolveCache = 4,
    PostResolveCache = 5,
    MapHandler = 6,
    PostMapHandler = 7,
    AcquireState = 8,
    PostAcquireState = 9,
    PreHandlerExecute = 10,
}
```

我们可以回顾一样如何在http module中为Authenticate绑定事件。

```
public class MyModule : IHttpModule
{
    public void Init(HttpApplication context)
    {
        // 將ctx_AuthRequest 綁定的 AuthenticateRequest 事件
        context.AuthenticateRequest += ctx_AuthRequest;
    }
    void ctx_AuthRequest(object sender, EventArgs e)
    {
    }
}
```

Owin这里的Use，貌似是借用了Node.js的用法呀- -！ 不管怎么说，通过这样一种方式，我们就可以将Owin中间件注册进IIS 集成模式的管道了。也就是说我们上面注册的CookieAuthenticationMiddleware会在AuthenticaRequest 阶段执行。而它就是真正生成cookie以及读取cookie的那只背后的手。

CookieAuthenticationMiddelware 负责读取用户信息cookie

如果你看的还算认真的话，我们上面讲claims-based认证的时候有一个小例子。我们只需要在AuthenticateRequest阶段将ClaimsPrincipal赋给当前的User对象就可以实现登录了，而这也是IAuthenticationManager.SignIn所做的。但是我们上面讲Forms登录的过程一样，用户登录之后，我们需要生成cookie，这样用户下次访问的时候就不需要登录了，我们在Authenticate Request去检测有没有这个cookie就可以了，CookieAuthenticationMiddleware就负责做了这两件事情。

我们可以到Katana的站点去下载源码，然后找到CookieAuthenticationMiddleware这个类，然后找到最后生成cookie和读取cookie的类：CookieAuthenticationHandler。

```
internal class CookieAuthenticationMiddleware : AuthenticationMiddleware<CookieAuthenticationOptions>
{
    private readonly ILogger _logger;

    public CookieAuthenticationMiddleware(OwinMiddleware next, IAppBuilder app,
        CookieAuthenticationOptions options)
        : base(next, options)...

    protected override AuthenticationHandler<CookieAuthenticationOptions> CreateHandler()
    {
        return new CookieAuthenticationHandler(_logger);
    }
```

在CookieAuthenticationMiddleware中有两个方法：

- AuthenticateCoreAsync：从request中读取cookie值，附给到identity对象，没有什么内幕，就是读读cookie进行解密，转成identity对象。
- ApplyResponseGrantAsync ： 往response中写入cookie值，同样没有什么内幕，有兴趣的同学可以下载katana源码瞅瞅。

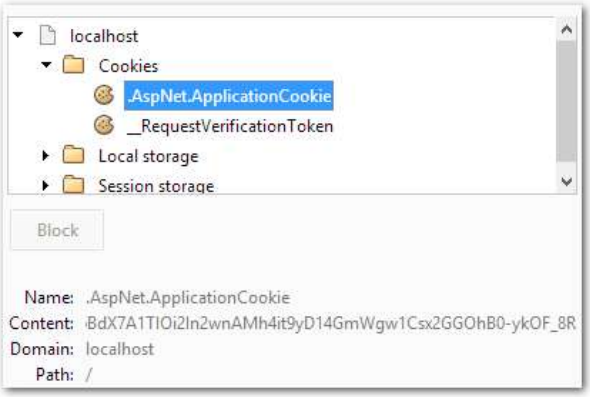

```
protected override async Task<AuthenticationTicket> AuthenticateCoreAsync()...

protected override async Task ApplyResponseGrantAsync()...
```

CookieAuthenticationMiddelware 对cookie的加密方式

在我们上篇文章中对ASP.NET Identity登录的例子中，如果你登录了，那么你会发现我们的cookie是经过加密的。而cookie的名称是以.AspNet.为前缀加上我们

Startup中配置的AuthenticationType。




```
// 配置Middleware 组件
app.UseCookieAuthentication(new CookieAuthenticationOptions
{
    AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
    LoginPath = new PathString("/Account/Login"),
    CookieSecure = CookieSecureOption.Never,
});
```

```
public static class DefaultAuthenticationTypes
{
    public const string ApplicationCookie = "ApplicationCookie";
    public const string ExternalBearer = "ExternalBearer";
    public const string ExternalCookie = "ExternalCookie";
}
```

那么接下来，我们就来看一下CookieAuthenticationMiddleware是以什么样的加密方式将我们的identity信息加密的，我们能不能将它解回来呢？

欲知后事如何，请听下回分解~

参考&小结

这一篇文章涉及到的新东西比较多，也花了我不少的时间。但是总的来说收获还是蛮大的，把Claims-based总结性的概括了一下，然后又开始了Owin的学习之旅。越来越发现.NET的强大，在开源社区的不断贡献下.NET也逐渐开始绽放出新的生命力。后面还会继续Owin的学习，有兴趣的朋友可以继续关注！还是我一直强调的，虽然ASP.NET Identity登录只有三行代码，但是背后却隐藏的如此之深，如果你不怀着一颗好奇以及好学的心，你永远不知道背后有多么美丽的故事。如果只是习惯了使用框架，而不去了解它，那就会迷失在学习新技术的路上。但是如果你知道它的设计原理，你就会发现其实都差不多的，思想就是那么一套，但是外观却可能随时改变。 另外的话我觉得这篇文章写的不错，值得点赞，你觉得呢？

我是Jesse Liu，关注我，跟我一起探寻.NET 那些新鲜，好玩，以及背后的故事吧 :)

参考资料：

- http://owin.org/
- http://brockallen.com/2013/10/24/a-primer-on-owin-cookie-authentication-middleware-for-the-asp-net-developer/
- http://www.asp.net/aspnet/overview/owin-and-katana/an-overview-of-project-katana
- http://www.asp.net/aspnet/overview/owin-and-katana/owin-middleware-in-the-iis-integrated-pipeline
- http://www.asp.net/aspnet/overview/owin-and-katana/owin-startup-class-detection
- http://msdn.microsoft.com/en-us/library/ff359101.aspx

分类: ASP.NET 安全系列 , C# 揭密 , Web , Owin系列

标签: Owin , Membership

腾飞（Jesse)
关注 - 15
粉丝 - 3719
荣誉：推荐博客
+加关注

« 上一篇：从Membership 到 .NET4.5 之 ASP.NET Identity
» 下一篇：一不小心写了个WEB服务器

posted @ 2014-04-01 08:27 腾飞（Jesse) 阅读(45147) 评论(101) 编辑 收藏

< Prev 1 2 3

评论列表

#101楼 2017-02-14 22:22 春華秋實

马克一个

支持(0) 反对(0)

< Prev 1 2 3

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。