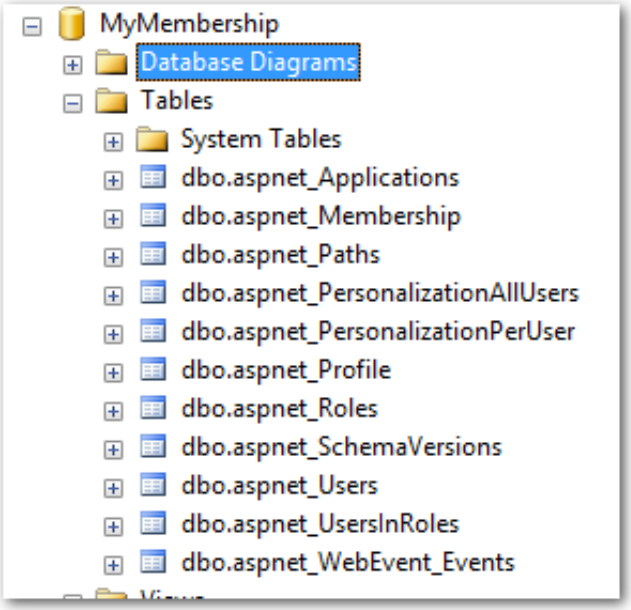
从Membership 到 .NET4.5 之 ASP.NET Identity - 文章 - 伯乐在线

> 我们前面已经讨论过了如何在一个网站中集成最基本的Membership功能，然后深入学习了Membership的架构设计。正所谓从实践从来，到实践从去，在我们把Membership的结构吃透之后，我们要完善它，改造它，这样我们才能真正学以致用。今天我们将以用户信息为主线，从SqlMembershipProvider出发，到ASP.NET Simple Membership最后再到MV5中引入的ASP.NET Identity，来看看微软是如何一步一步的改造这套框架的。

引入 - 用户信息是如何存在数据库中的

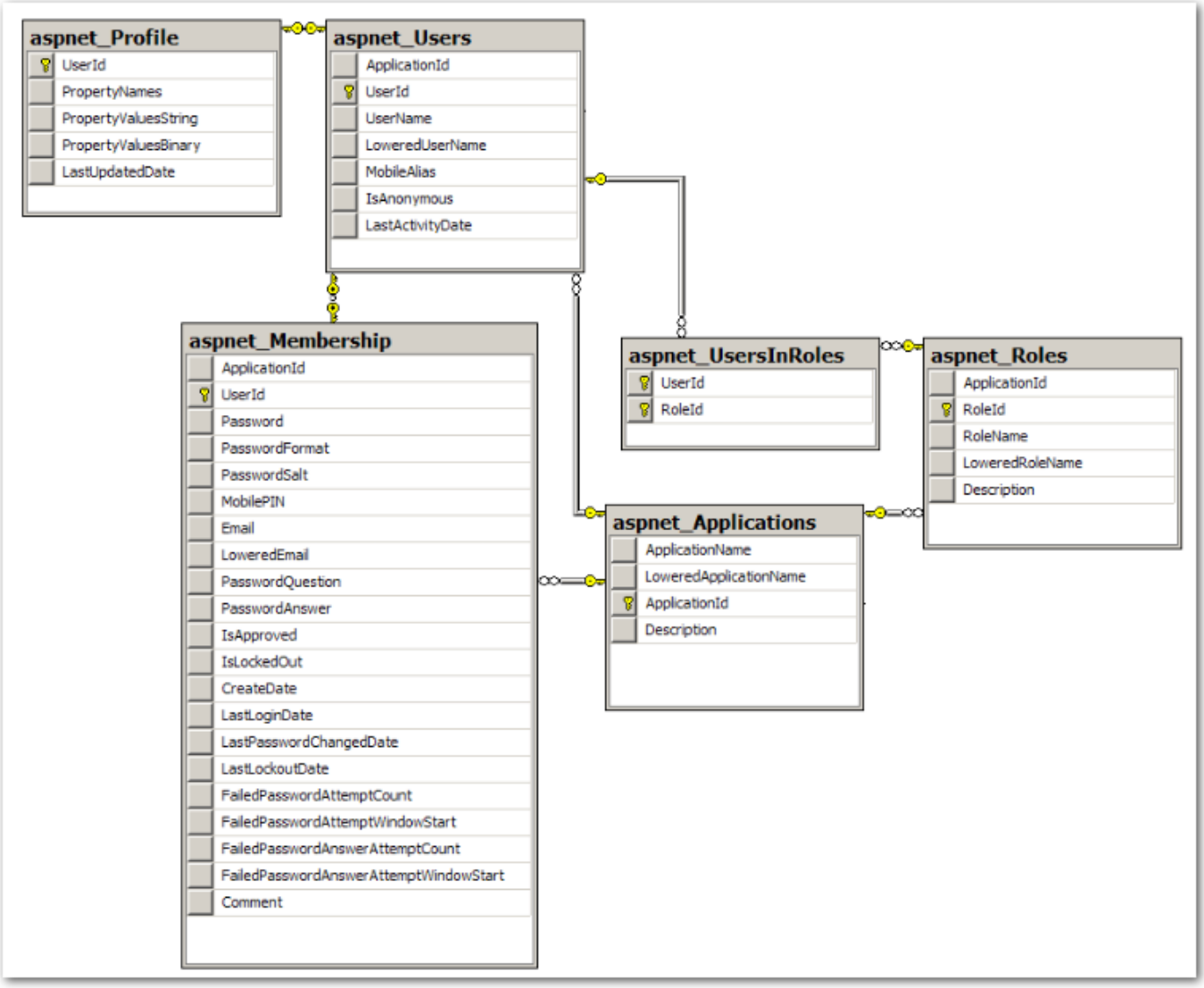
我们前两篇都只讲到了怎么用Membership注册，登录等，但是我们漏掉了一个很重要并且是基本上每个用Membership的人都想问的，我的用户信息怎么保存？我不可能只有用户名和密码，如果我要加其它的字段怎么办？我们首先来看一下，SqlMembershipProvider是如何做的，毕竟这个Provider是跟着Membership框架一起诞生出来的。

ASP.NET 2.0时代，我们需要借助一个VS提供的一个工具来帮助我们生成所需要的表。打开VS 开发者命令行工具，输入aspnet_regsql，后面简单的连接一下数据库就会帮我们生成以下的几张表：



我们这里简要关注以下几张表的结构就可以了。

aspnet_Membership	用户信息的关键表，一些通用必要的字段都在这个表里面
aspnet_Profile	用户资料表，主要用来给 Profile Provider 扩展用户信息字段
aspnet_Roles	角色表
aspnet_Users	用户信息表，除了 aspnet_Membership 的必要字段以外的一些额外信息。
aspnet_UsersInRoles	用户角色表



我想上面两张图应该可以说明很多问题，用户信息的一些基本字段比如用户名，密码以及一些其它登录的信息存储在哪里，角色存储在哪里，角色和用户之间是如何关联的等等，但是还有正如本节标题所说的一样，用户信息字段如何扩展呢？

ProfileProvider 来扩展用户信息

我们上面讲到有一张表aspnet_Profile是专门用来给ProfileProvider为扩展用户信息的。它和MebershipProvider, RoleProvider一起组成了用户信息，权限管理这样一套完整的框架。下面我们就来看看如何用ProfileProvider来扩展我们想要的用户信息。

- 我们先添加一个Model继承ProfileBase来为我们新的用户对象建模
- 在web.config配置ProfileProvider
- 在MVC站点中实现对我们的用户信息的管理

UserProfile的代码

```
public class UserProfile: ProfileBase
{
        [SettingsAllowAnonymous(false)]
        public string FirstName
        {
                get { return base["FirstName"] as string; }
                set { base["FirstName"] = value; }
        }
        [SettingsAllowAnonymous(false)]
        public string LastName
        {
                get { return base["LastName"] as string; }
                set { base["LastName"] = value; }
        }
        public static UserProfile GetUserProfile(string username)
        {
                return Create(username) as UserProfile;
        }
}
```

我们的UserProfile的所有字段都要从基类从获取，基类中以object类型存储着这些值。

web.config的配置

```
<profile defaultProvider="defaultProfileProvider"
         inherits="MyMembership.Models.UserProfile">
  <providers>
    <add name="defaultProfileProvider"
         type="System.Web.Profile.SqlProfileProvider,System.Web, Versio
         connectionStringName="DefaultConnection"/>
  </providers>
</profile>
```

大家可以看到profile里面的inherits结点我们设置了我们上一步建立的那个对象，这样我们就可以在代码将MVC里面的Profile对象转换成我们要的这些类型。

从Profile对象中获取当前登录用户的信息

1

2

3

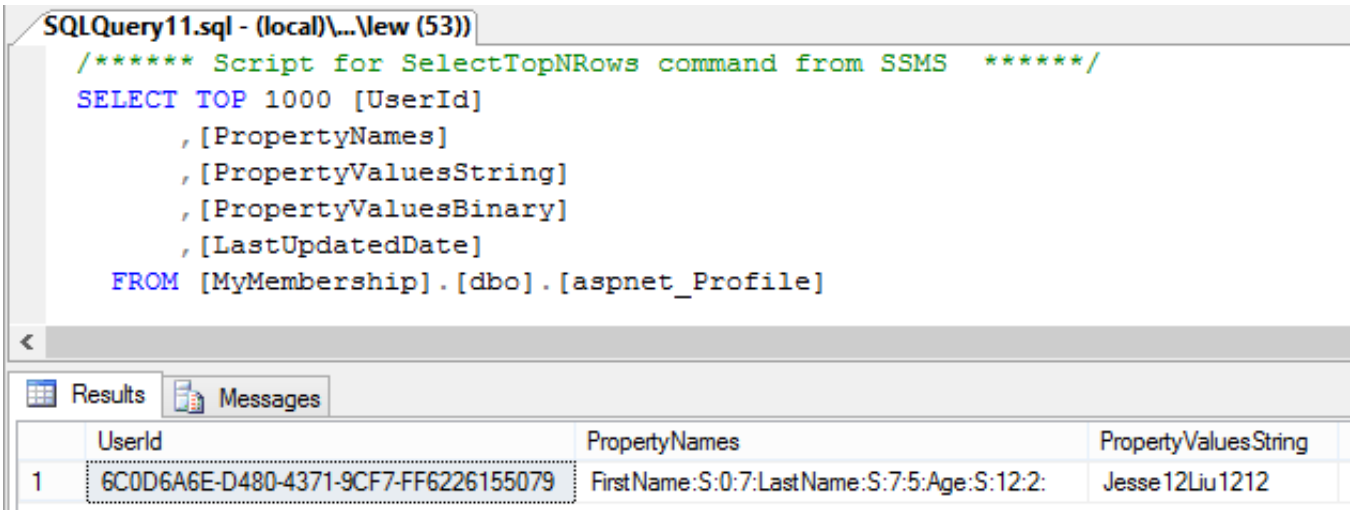
4

5

```
6
7
8
9
10
public ActionResult Manage()
{
        var profile = Profile as UserProfile;
        var model = new UserProfileViewModel
        {
                FirstName = profile.FirstName,
                LastName = profile.LastName
        };
        return View(model);
}
1
2
3
4
5
6
7
8
9
10
11
12
13
public ActionResult Manage(UserProfileViewModel model)
{
        if (ModelState.IsValid)
        {
                var myProfile = Profile as UserProfile;
                myProfile.FirstName = model.FirstName;
                myProfile.LastName = model.LastName;
                myProfile.Save();
                return RedirectToAction("Index", "Home");
        }
        return View(model);
}
```

怎么样？是不是不复杂？加上我们前面学到的MembershipProvider,RoleProvider那么我们很轻松就可以将这一系列登录、授权、认证以及用户模块相关的功能完成了。如果要使用ProfileProvider的话，最好

是在最开始的设计阶段就使用，因为要想把ProfileProvider直接集成到现有的老系统中，那是一件很难的事情，我们看一下Profile表的结构就知道了。



Profile要做到通用，那么这张表就要求能够存储任意类型的数据，所以微软就采用一种这样的设计，把所有的字段以string的格式放到了一列中，然后再解析出来。别的先不说，首先这种设计对于大型系统来说，肯定会有一个性能的瓶颈，并且如果我们想要把ProfileProvider集成到老的系统中，那会是一件很难的事情。那么微软后面做了哪些改进呢？

Simple Membership Provider

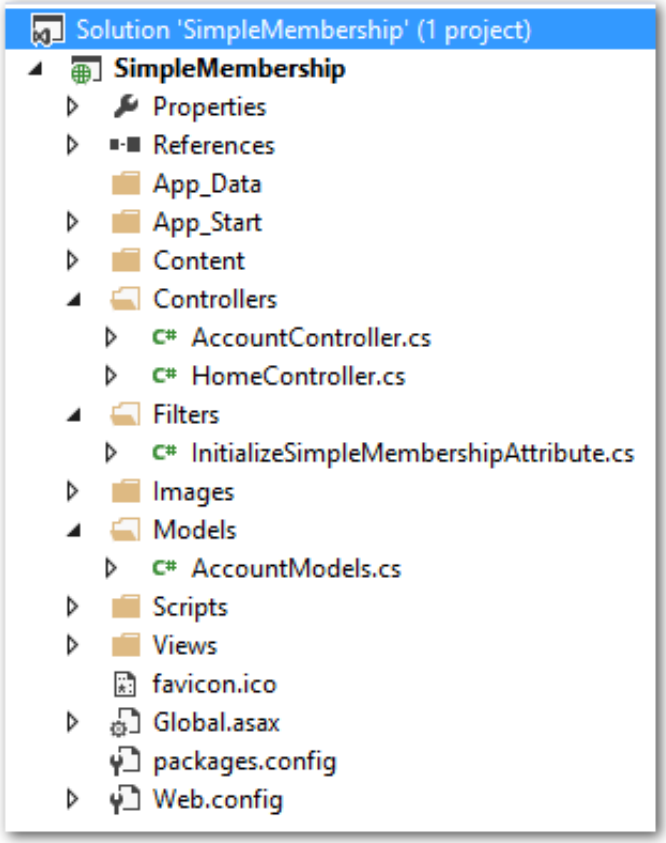
假想一下，你使用了SQL Membership Provider，你想抱怨哪些问题呢？

1. 最先抱怨的肯定是没有办法自定义用户信息，必须要通过ProfileProvider，那玩意儿真心不好用！
2. 其实与现有或其它系统集成简直是太麻烦了！！
3. 数据表都被你定义好了，但是很抱歉，那都不是我想要的啊！！！
4. 等等。。。

好吧，这些问题确实是导致Membership一直不温不火的原因之一。 所有这就是为什么后来，我们有了Simple Mebership Provider，借助于它：

1. 我们不必再依懒于Profile Provider去扩展用户信息。
2. 可以完全让Membership 根据我们自己定义的表结构来运行。
3. 与Entity Framework集成，好吧（微软这是捆绑销售么？ 惯用伎俩）
4. 另外，在VS2012或2013中创建一个MVC4.0的Internet程序，就会为你自动添加所有代码！

最后一招够狠，我们来试一下。在VS2012中创建一个4.0 的MVC站点，就可以在Controllers和Models中发现相关代码，在AccountController中已经有了登录注册相关的代码。



在AccountModel中，我们可以找到一个UserProfile的类就是一个Entity Framework 的实体类。

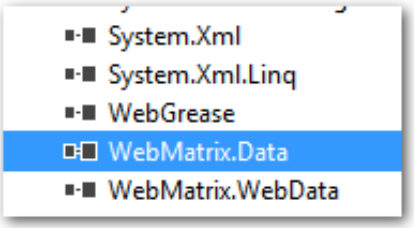
```
1
2
3
4
5
6
7
8
[Table("UserProfile")]
public class UserProfile
{
        [Key]
        [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
        public int UserId { get; set; }
        public string UserName { get; set; }
}
```

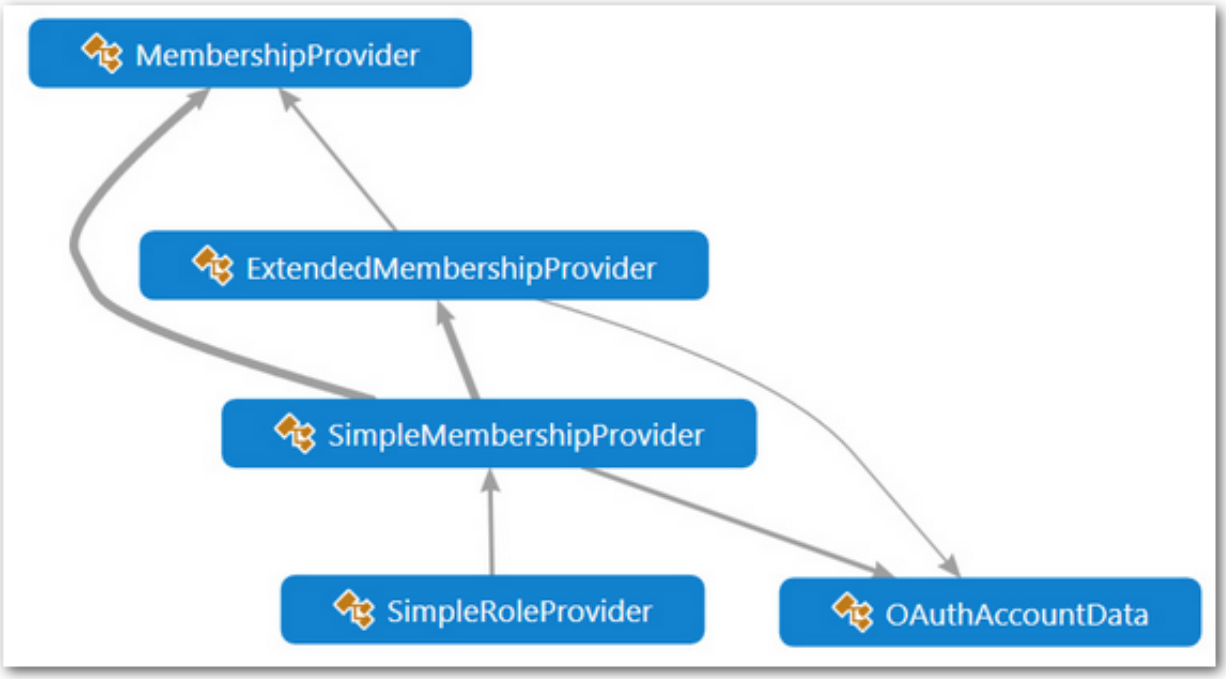
那么我们就可以像这样查询用户的信息了。

```
1
2
3
4
var context = new UsersContext();
```

```
var username = User.Identity.Name;
var user = context.UserProfiles.SingleOrDefault(u => u.UserName == username);
var birthday = user.Birthday;
```

有人可能会问，那这个我直接用EF来整个用户实体类做登录模块有啥区别？ 我也怀疑区别就是可以在创建membership用户记录的时候，可以一起把我们的额外信息带进去，其余的还真没有发现什么区别。SimpleMembershipProvider所有的操作都是通过WebSecurity这个类来完成的，这个类所完成的功能与Membershipo类是一样的，主要是对Provider的功能进行一个封装，而这个类是包含在WebMatrix.WebData.dll中的。打开网站的引用目录发现引用了WebMatrix.Data和WebMatrix.WebData这两个dll。这两个dll主要是给web page用的， 而SimpleMembershipProvider的相关代码就包含在这两个dll当中。





里面怎么实现的我想就不用详述了，无非就是继承MembershipProvider然后覆盖其中的一些方法而已。我们Membership系列第二篇已经详述过了，有兴趣的同学请移步。在后来微软还推出来Universal Providers，用来帮助Membership转移到Windows Azure的以及对SQL Compact的支持。

ASP.NET Identity

基础示例

ASP.NET Identity是在.NET Framework4.5中引入的，从Membership发布以来，我想微软已经从开发者以及企业客户那里面得到了足够的反馈信息来帮助他们打造这样一套新的框架。他所拥有的特点大多也是前面所不能满足的，至少我们看到的是进步，不是么？

1. 一套ASP.NET Identity，可以用于ASP.NET下的web form, MVC, web pages, web API等
2. 和Simple Membership Provider，可以灵活订制用户信息，同样采用EF Code First来完成数据操作
3. 完全自定义数据结构
4. 单元测试的支持
5. 与Role Provider集成
6. 支持面向Clamis的认证
7. 支持社交账号的登录
8. OWIN 集成
9. 通过NuGet发布来实现快速迭代

瞟一眼好处还真不少，但是至少对于开发者来说，好用，能满足需求，灵活才是王道，那我们下面就来看看如何使用ASP.NET Identity来完成我们的用户授权和认证模块。其实我们已经不用写任何示例代码，因为我们只要使用VS创建一个.NET Framework 4.5 的 MVC站点，所有的代码都已经包括了。

默认创建的IdentityModels.cs

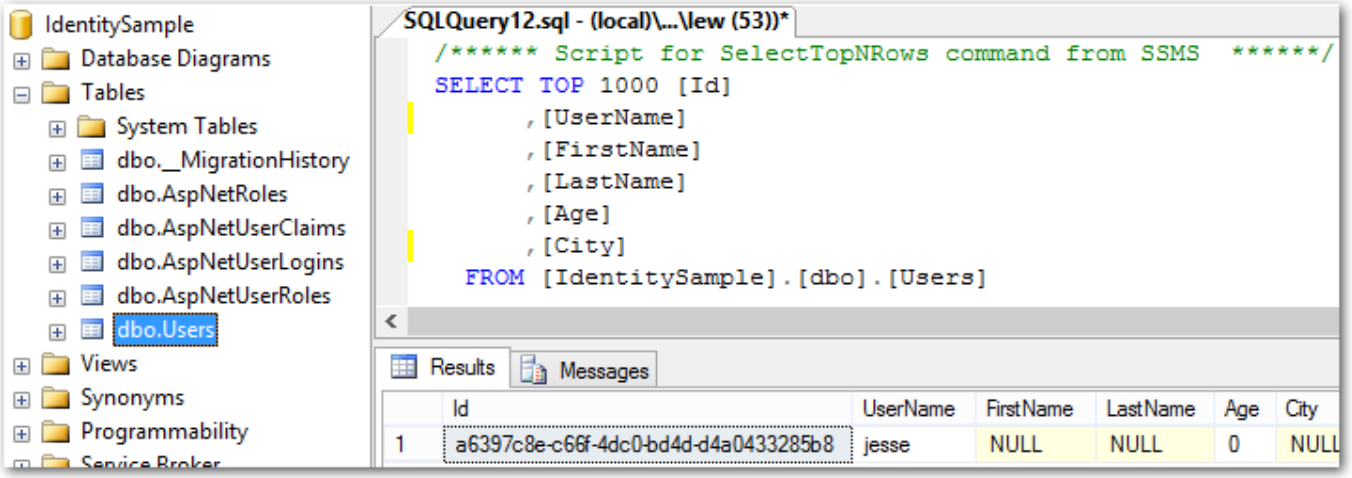
```
public class ApplicationUser : IdentityUser
{       }
public class ApplicationDbContext : IdentityDbContext
{
        public ApplicationDbContext()
                : base("DefaultConnection")
        {
        }
}
```

我们需要在ApplicaitonUser实体中添加我们的用户字段就可以了，同时我们还可以很简单的更改表名。

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

```
18
19
20
21
22
public class ApplicationUser : IdentityUser
{
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public int Age { get; set; }
        public string City { get; set; }
}
public class ApplicationDbContext : IdentityDbContext
{
        public ApplicationDbContext()
                : base("DefaultConnection"){}
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
                base.OnModelCreating(modelBuilder);
                // 默认表名是AspNetUsers,我们可以把它改成任意我们想要的
                modelBuilder.Entity()
                        .ToTable("Users");
                modelBuilder.Entity()
                        .ToTable("Users");
        }
}
```

接下来，你就可以run一下你的网站，来体验一把ASP.NET Identity了，别忘了先把web.config里面的连接字符串改一下，方便我们自己去查看数据库，只要设置一下数据库就可以了，创建工作就交给EF吧。



我们可以在AccountController中找到所有的相关代码。

初始化UserManager对象

```
var user = new ApplicationUser() { UserName = model.UserName };
var result = await UserManager.CreateAsync(user, model.Password);
```

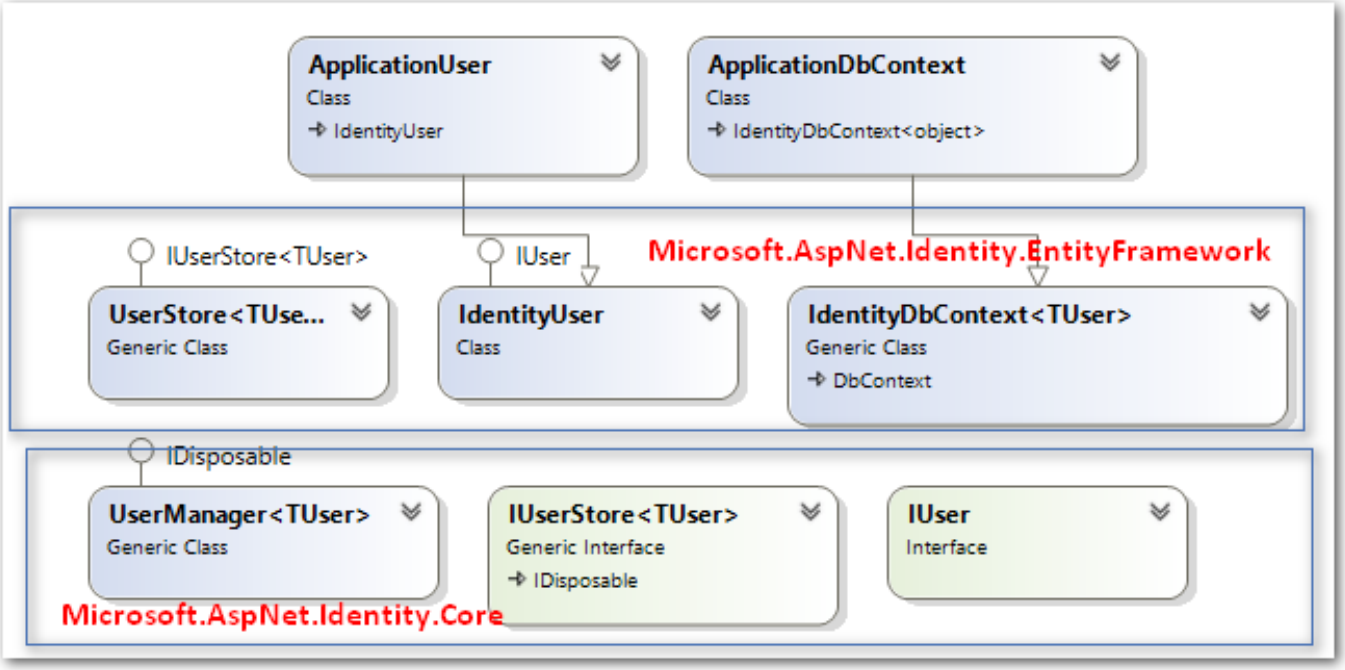
框架设计

我们上面是直接利用VS帮助我们创建好了一些初始代码，我们也可以创建一个空白的站点，然后再把ASP.NET Identity引用进来。所需要的类库可以直接从Nuget上下载就可以了。

主要包括ASP.NET Identity 的EF 部分的实现，有了EF的帮助我们就可以完全自定义数据结构，当然我们也只需要定义一个实体类就可以了。

名字就已经告诉大家了，这是ASP.NET Identity的核心了，所以主要的功能在这里面。上面那个包是ASP.NET Identity EF的实现，那么我们可以在这个核心包的基础上扩展出基于No SQL, Azure Storage 的 ASP.NET Identity实现。

ASP.NET Identity对OWIN 认证的支持。



最上面两个就是我们自己创建的代码，分别继承自己Microsoft.AspNet.Identity.EntityFramework的IdentityUser和IdentityDbContext。但是最后别忘了，我们与用户相关的操作实际上是通过Microsoft.AspNet.Identity.Core的 UserManager类来完成的。通过这样一种设计，可以把具体定义和实现交给上层，但是最后的核心却完全由自己掌控，实现松耦合，高内聚（一不小心我竟然说出了这么专业的解释，小心脏砰砰跳呀！）。

框架实现剖析

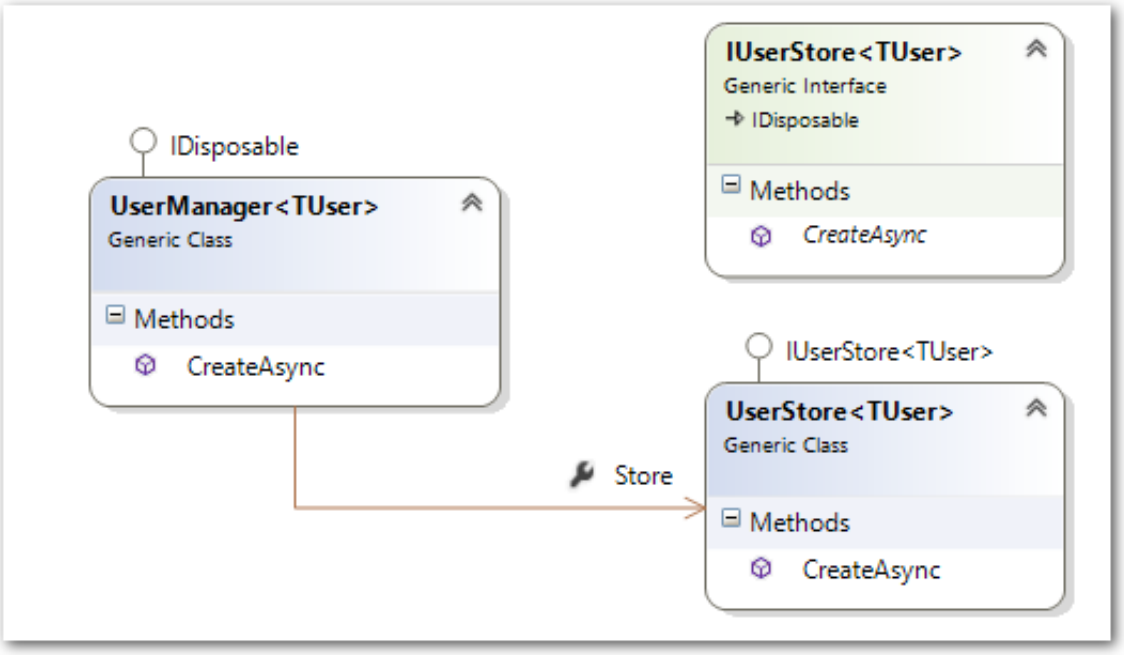
上面只是一张粗略的类图，下面我们就来看一下这些类之间是如何关联起来协作的。我们通过上面基础示例的代码可以发现，用用户相关的功能是通过调用UserManager的方法来完成的。 我们可以在AccountController中找到UserManager的初始代码：

```
1
new UserManager(new UserStore(new ApplicationDbContext()));
```

虽然所说有的方法通过UserManager来调用，但是最后实现的还是UserStore，并且如果我们找到UserManager的定义，会发现实际上它所接收的正是在Microsoft.AspNet.Identity.Core中定义的IUserStore接口。

```
1
2
3
4
public UserManager(IUserStore store)
{
        this.Store = store;
}
```

我们现在使用的是ASP.NET Identity EF的实现，所以在UserStore中，直接调用传进来的DbContext的Save操作就可以了。



有没有发现这张图和我们第二篇中讲的Provider模式有那么点点的神似？ 在Membership中，我们所有的操作通过调用Membership来过多成，但是Membership本身只是一个包装类，内部的操作实际上是通过Provider的实际类来完成的，这就是策略模式的典型案例。只不过Membership的Provider通过web.config配置完成，而UserManager通过构造函数注入完成。

扩展ASP.NET Identity – 将用户信息写入文件

为了熟悉AspNet.Identity的结构，我们来扩展实现一个将用户信息写入文件的组件，然后实现登录注册功能，我们就给它命名AspNet.Identity.File吧。

1. 创建一个自己的用户类（UserIdentity）实现Microsoft.AspNet.Identity.IUser接口
2. 创建一个自己的UserStore类实现Microsoft.AspNet.Identity.IUserStore接口

3. 作为演示，我们的用户类就尽量简单，只有id，用户名，和密码三个属性
4. 我们的UserStore，也只重写了Get和Create几个基本的方法，没有重写Update。

UserIdentity.cs 代码