### 用户实体

User实体表示**应用程序**的**用户**。

public class User : AbpUser<Tenant, User>

{

//add your own user properties here

}

用户存储在数据库的**AbpUsers**表中。您可以向User类添加自定义属性（并为更改创建数据库迁移）。

AbpUser类定义了一些基本属性。一些属性是：

* **UserName**：**用户的**登录名。对于[租户来说](https://aspnetboilerplate.com/Pages/Documents/Zero/Tenant-Management)应该是**独一无二**的 。
* **EmailAddress**：用户的电子邮件地址。 对于[租户来说](https://aspnetboilerplate.com/Pages/Documents/Zero/Tenant-Management)应该是**独一无二**的。
* **Password**：用户的哈希密码。
* **IsActive**：是的，如果此用户可以登录到该应用程序。
* 用户的**姓名(Name)**和**姓氏(Surname)**。

用户的**ID**定义为**long**。

### 用户管理器

**UserManager**是为用户执行**域逻辑**的服务：

public class UserManager : AbpUserManager<Tenant, Role, User>

{

//...

}

您可以[注入](https://aspnetboilerplate.com/Pages/Documents/Dependency-Injection)并使用UserManager来创建，删除，更新用户，授予权限，更改用户角色等等。

#### 用户登录

Module Zero定义了LoginManager，它具有用于登录应用程序的**LoginAsync**方法。

#### 关于IdentityResults

UserManager的某些方法返回IdentityResult而不是抛出异常。

IdentityResult定义了**CheckErrors**扩展方法，该方法自动检查错误并在需要时抛出异常

(await UserManager.CreateAsync(user)).CheckErrors();

### 外部认证

Module Zero的Login方法从 数据库中的**AbpUsers**表对用户进行身份验证。某些应用程序可能要求您从某些外部源（如活动目录，其他数据库的表，甚至远程服务）对用户进行身份验证。

对于这种情况，UserManager定义了一个名为“外部认证源”的扩展点。我们可以创建一个派生自**IExternalAuthenticationSource**的类， 并将其注册到配置中。有一个**DefaultExternalAuthenticationSource**类来简化IExternalAuthenticationSource的实现。我们来看一个例子：

public class MyExternalAuthSource : DefaultExternalAuthenticationSource<Tenant, User>, ITransientDependency

{

public override string Name

{

get { return "MyCustomSource"; }

}

public override Task<bool> TryAuthenticateAsync(string userNameOrEmailAddress, string plainPassword, Tenant tenant)

{

//TODO: authenticate user and return true or false

}

}

要注册我们的验证源，我们可以在模块的[PreInitialize](https://aspnetboilerplate.com/Pages/Documents/Module-System)方法中使用这样的代码 ：

Configuration.Modules.Zero().UserManagement.ExternalAuthenticationSources.Add<MyExternalAuthSource>();