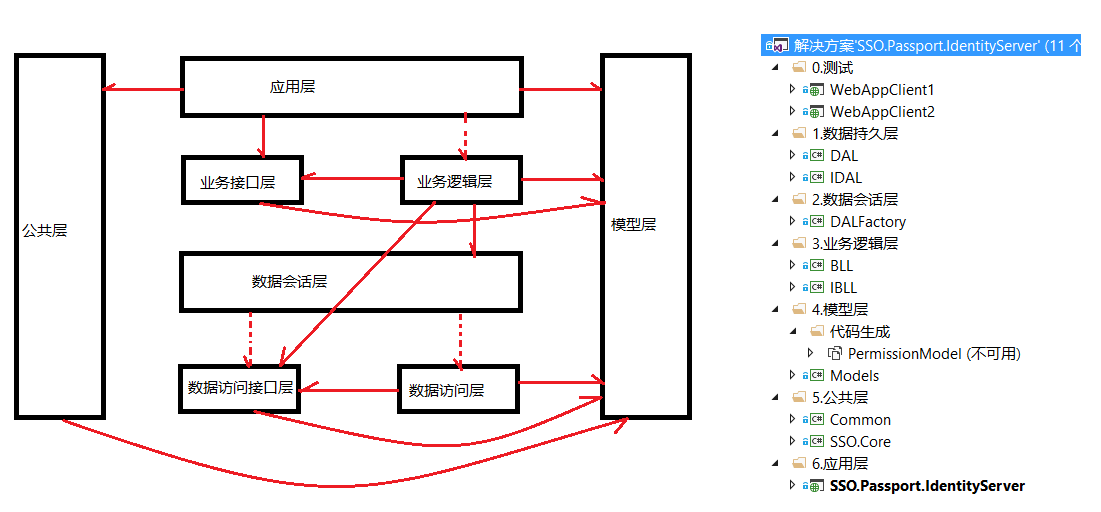
# SSO.Passport.IdentityServer项目介绍

## 项目架构介绍



实线代表实际引用，虚线代表通过反射的方式引用；

应用层的真实引用只需要引用Common和Model以及IBLL即可，从而达到项目间的解耦，但由于生成和发布的原因，导致需要反射引用的类库不能生成到应用层目录下，所以应用层也全部引用了运行时需要的所有项目；

反射引用需要在应用层的web.config中配置DalNameSpace和DalPath以及BllPath，若未配置，则取程序相应的预设值；

**如何将应用层的数据请求传到数据层的？**

应用程序启动时将通过反射的方式加载业务逻辑层BLL的程序集，并创建相应的接口实例存到autofac容器中，当应用层发起数据请求时，将通过IBLL的接口实例进行处理，依靠BLL调用数据会话层DALFactory来反射创建数据访问层DAL对应的接口实例IDAL，IDAL依靠DAL进行真实的数据库操作，最后将数据返回到应用层。

## 数据访问框架封装

数据库框架是基于EntityFramework6.1.3为基础进行数据访问层的封装，额外的实现了EntityFramework的一级缓存和二级缓存以及大批量操作优化，分别提供了基本操作、不跟踪操作、缓存操作、异步操作等，DAL层主要的是生成SQL语句和实体的上下文附加操作，BLL层均分别提供了统一保存方法以及增删改并保存的方法；

模型层使用CodeFirst为主，带数据库自动迁移、数据库自动创建和初始数据，由于启用了二级缓存，数据上下文对象重写了SaveChanges方法；并对DbContext做了一些额外的配置：

Configuration.AutoDetectChangesEnabled = false; //关闭自动跟踪对象的属性变化

Configuration.LazyLoadingEnabled = false; //关闭延迟加载

Configuration.ProxyCreationEnabled = false; //关闭代理类

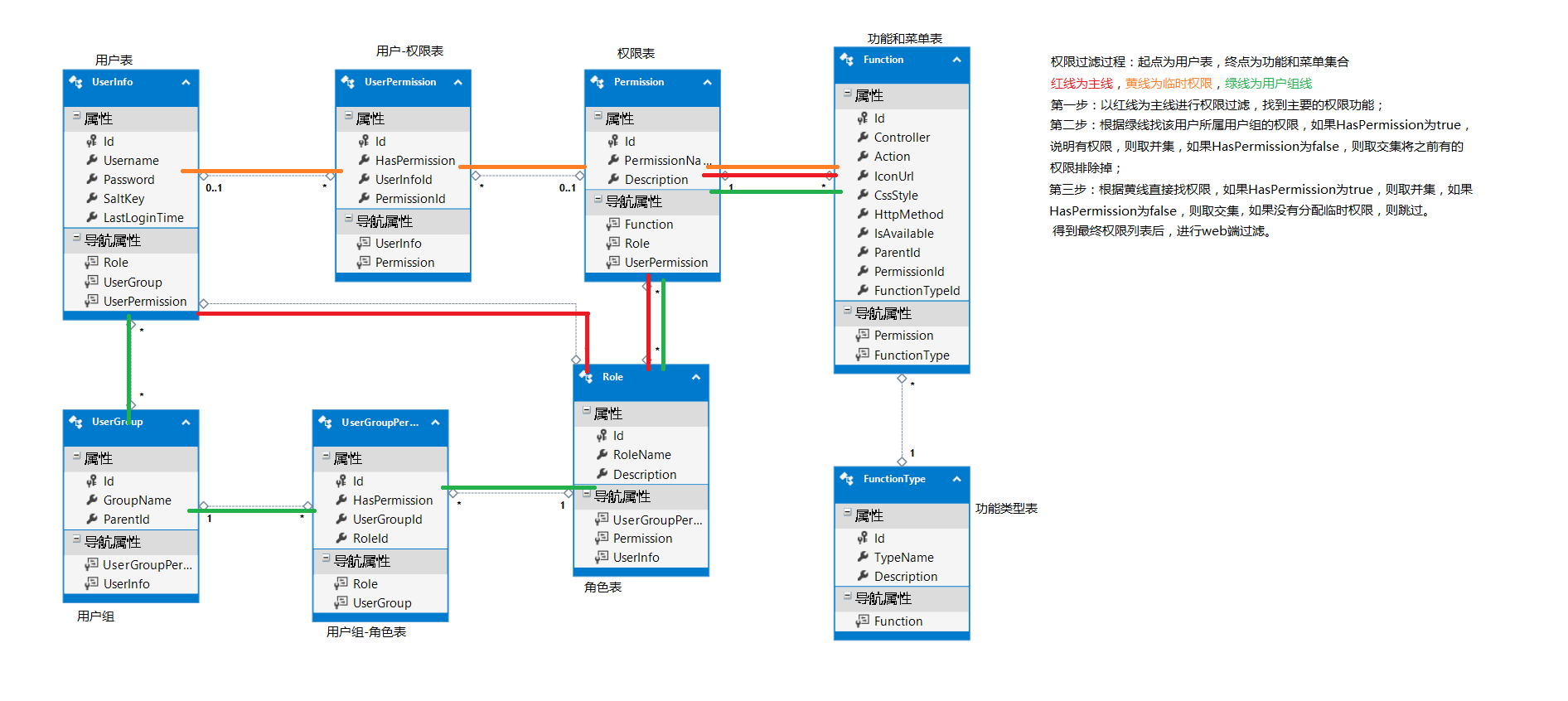
Configuration.ValidateOnSaveEnabled = false; //关闭保存时的实体验证

Configuration.UseDatabaseNullSemantics = true; //关闭数据库null比较行为

解决方案中的ModelFirst的项目仅用于代码生成，不作为项目运行时必须项目引用；

模型层分别提供了与每个实体对应的传入和传出的DTO对象以及ViewModel对象映射关系，映射配置位于Common.RegisterAutomapper.Excute()。

## 权限控制介绍



## SSO单点登录介绍

为了方便说明先说明几个名词的含义：

　　P站：统一登录授权验证中心，项目中的SSO.Passport.IdentityServer

　　A站：处于不同域名下的客户端网站A，项目中的WebAppClient1

　　B站：处于不同域名下的客户端网站B，项目中的WebAppClient2

　　Token：用户访问P站的秘钥

Ticket：用来保存用户信息的加密字符串；

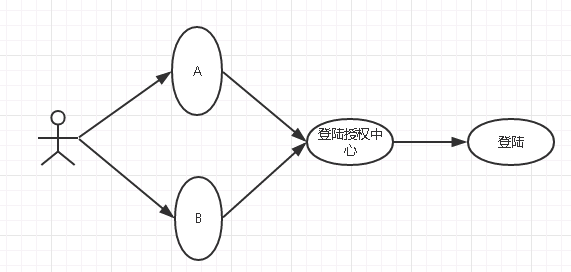
**什么是SSO？**

访问A站需要登陆的就跳转P站中进行登陆，P站登陆之后跳转回至A站，用户再次访问B站需要登陆的页面，用户不需要进行登陆操作就可以正常访问。

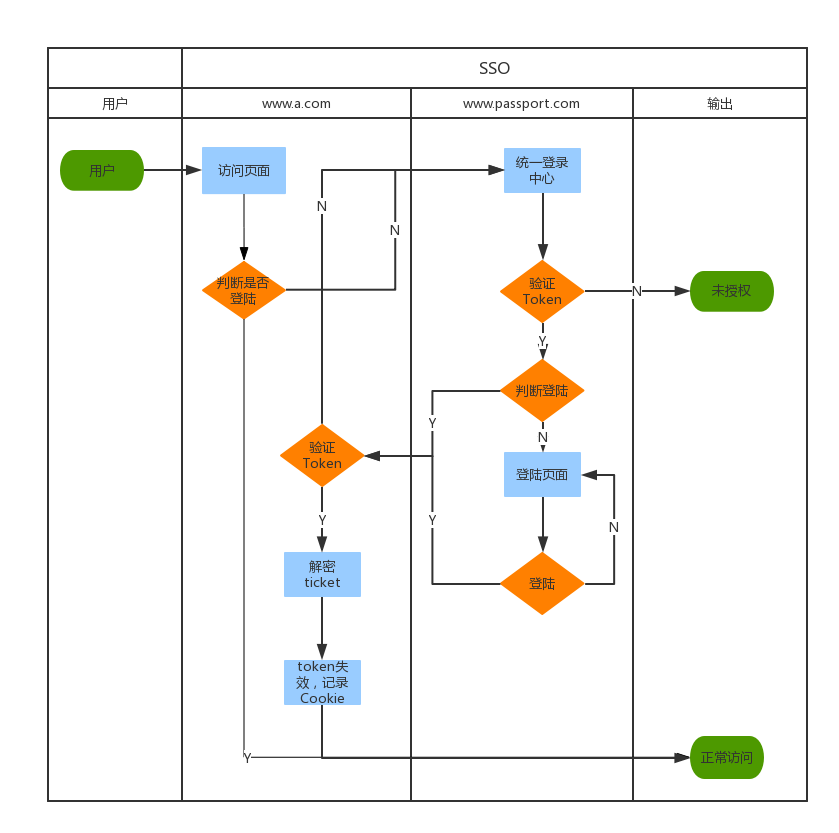
**实现思路**

未登录用户访问A站，首先会重定向跳转至P站授权中心，P站首先通过检测Cookie来判断当前不是处于登陆状态，就跳转至登陆页面进行登陆操作，登陆成功之后把用户信息加密ticket附在A的请求地址上返回，A站通过解密ticket来获取用户信息，解密成功并存进Session中（这样用户在A中就处于登陆状态了），访问通过；当用户再次访问B站的时候，对于B站来说，用户是处于未登录状态，则同样会重定向跳转至P站授权中心，P站检测Cookie，判断当前用户处于登陆状态，就把当前用户信息加密成ticket附在B的请求地址上返回,后面的操作就和A站处理一样；这样都登陆之后再次访问A或者B，A和B中Session中都存储了用户信息，就不会再次请求P站了。

**简单关系图**



**泳道流程图**



**如何使用：**

客户端网站引用SSO.Core项目，并在web.config中配置<add key="PassportUrl" value="http://localhost:56075" />，然后在需要登录授权的控制器或方法上打上标签[Authority(Code = AuthCodeEnum.**Login**)]即可；

**主要流程说明：**

A站主要逻辑：

用户首先访问A站，A站中会生成Token，并存入Cache中。Token是A访问P的钥匙，P在回调给A的时候需要携带这个Token。A请求P，P验证Token，P回调A，A检测Token是否是发送出去的Token，验证之后Token即失效，防止Token被再次使用。

Token的生成是通过取时间戳的不同字段进行MD5加密生成，当然这里可以再加个盐进行防伪，对应的方法是SSO.Core.Client.AuthernUtil.CreateToken(DateTime timestamp)。

P回调A的时候进行，A中对Token进行校验，校验不成功则请求P站统一授权验证，即Authority标签。

P站主要逻辑：

P站收到授权请求，触发Passport/PassportVertify控制器方法，首先通过Coookie来判断是否登陆，未登录则跳转至登陆页面进行登陆操作。

已登陆则调用SSO.Core.Server.PassportService.AuthernVertify(string token, DateTime timestamp)验证Token。

未登录则跳转到登陆页面进行统一授权，在登陆页面加载时也判断Session和Cookie是否存在，若存在，则进行自动登录；

当其他客户端B网站也需要授权时，跳转到P后，若已经登陆，则自动为B网站分配用户密钥；

注销逻辑：

当客户端网站注销时，不会影响到其他客户端网站的登录状态，当P站被注销时，则所有客户端登录状态会丢失。

## 授权方式介绍

授权方式分为SSO授权、hash时效性校验以及不进行校验，对应的Authority授权码分别为：

SSO：Authority(Code = AuthCodeEnum.**Login**)

Hash：Authority(Code = AuthCodeEnum.**HashCheck**)

Public：Authority(Code = AuthCodeEnum.Public)或无

特别的，hash时效性校验必需参数：

time：当前时间到1970-01-01 00:00:00的秒数，

hash：time和加密盐拼接后的MD5一次加密结果，加密盐需要在config文件中AppSettings节点下键为encryptSalt，若未配置，则取“masuit”的DES默认加密结果；

该校验方式一般用于客户端的RestAPI访问方式以加密url进行防篡改校验。