单点登录将多个子系统的认证体系打通，实现了一个入口多处使用，而在架构单点登录时，也会遇到一些小问题，在不同的应用环境中可以采用不同的单点登录实现方案来满足需求。

#### 1、共享Session

共享Session可为是实现单点登录最直接、最简单的方式。将用户认证信息保存于Session中，即以Session内存储的值为用户凭证，这在单个站点内使用是很正常也很容易实现的，而在用户验证、用户信息管理与业务应用分离的场景下即会遇到单点登录的问题，在应用体系简单，子系统很少的情况下，可以考虑采用Session共享的方法来处理这个问题。



这个架构使用了基于Redis的Session共享方案。将Session存储于Redis上，然后将整个系统的全局Cookie Domain设置于顶级域名上，这样SessionId就能在各个子系统间共享。

这个方案存在着严重的扩展性问题，首先，ASP.NET的Session存储必须为SessionStateItemController对象，而存储的结构是经过序列化后经过加密存储的。

并且当用户访问应用时，他首先做的就是将存储容器里的所有内容全部取出，并且反序列化为SessionStateItemController对象。这就决定了他具有以下约束：

* Session中所涉及的类型必须是子系统中共同拥有的（即程序集、类型都需要一致），这导致Session的使用受到诸多限制；
* 跨顶级域名的情况完全无法处理；

#### 2、基于OpenId的单点登录

这种单点登录将用户的身份标识信息简化为OpenId存放于客户端，当用户登录某个子系统时，将OpenId传送到服务端，服务端根据OpenId构造用户验证信息，多用于C/S与B/S相结合的系统，流程如下：



由上图可以看到，这套单点登录依赖于OpenId的传递，其验证的基础在于OpenId的存储以及发送。

1. 当用户第一次登录时，将用户名和密码发送给验证服务；
2. 验证服务将用户标识OpenId返回到客户端；
3. 客户端进行存储；
4. 访问子系统时，将OpenId发送到子系统；
5. 子系统将OpenId转发到验证服务；
6. 验证服务将用户认证信息返回给子系统；
7. 子系统构建用户验证信息后将授权后的内容返回给客户端。

这套单点登录验证机制的主要问题在于他基于C/S架构下降用户的OpenId存储于客户端，在子系统之间发送OpenId，而B/S模式下要做到这一点就显得较为困难。为了处理这个问题我们将引出下一种方式，这种方式将解决B/S模式下的OpenId的存储、传递问题。

#### 3、基于Cookie的OpenId存储方案

Cookie的作用在于充当一个信息载体在Server端和Browser端进行信息传递，而Cookie一般是以域名为分隔的，例如a.xxx.com与b.xxx.com是可以访问xxx.com下的Cookie的，于是就能将顶级域名的Cookie作为OpenId的载体。



验证步骤和上第二个方法非常相似：

* 在提供验证服务的站点里登录；
* 将OpenId写入顶级域名Cookie里；
* 访问子系统（Cookie里带有OpenId）
* 子系统取出OpenId通过并向验证服务发送OpenId
* 返回用户认证信息
* 返回授权后的内容

在以上两种方法中我们都可以看到通过OpenId解耦了Session共享方案中的类型等问题，并且构造用户验证信息将更灵活，子系统间的验证是相互独立的，但是在第三种方案里，我们基于所有子系统都是同一个顶级域名的假设，而在实际生产环境里有多个域名是很正常的事情，那么就不得不考虑跨域问题究竟如何解决。

#### 4、B/S多域名环境下的单点登录处理

在多个顶级域名情况下，我们将无法让各个子系统的OpenId共享。处理B/S环境下的跨域问题，我们首先就应该想到JSONP的方案。



验证步骤如下：

* 用户通过登录子系统进行用户登录；
* 用户登录子系统记录了用户的登录状态、OpenId等信息；
* 用户使用业务子系统；
* 若用户未登录业务子系统则将用户跳转至用户登录子系统；
* 用户子系统通过JSONP接口将用户OpenId传给业务子系统；
* 业务子系统通过OpenId调用验证服务；
* 验证服务返回认证信息、业务子系统构造用户登录凭证；（此时用户客户端已经与子业务系统的验证信息已经一一对应）
* 将用户登录结果返回用户登录子系统，若成功登录则将用户跳转回业务子系统；
* 将授权后的内容返回客户端；

#### 5、安全问题

经过以上步骤，跨域情况下的单点登录问题已经可以得到解决。而在整个开发过程初期，我们采用用户表中记录一个OpenId字段来保存用户OpenId，而这个机制下很明显存在一些安全性、扩展性问题。这个扩展性问题主要体现在一个方面：OpenId的安全性和用户体验的矛盾。

整个单点登录的机制决定了OpenId是会出现在客户端的，所以OpenId需要有过期机制，假如用户在一个终端登录的话可以选择在用户每次登录或者每次退出时刷新OpenId，而在多终端登录的情况下就会出现矛盾：当一个终端刷新了OpenId之后其他终端将无法正常授权。

而最终，我采用了单用户多OpenId的解决方案。每次用户通过用户名/密码登录时，产生一个OpenId保存在Redis里，并且设定过期时间，这样多个终端登录就会有多个OpenId与之对应，不再会存在一个OpenId失效所有终端验证都失效的情况。