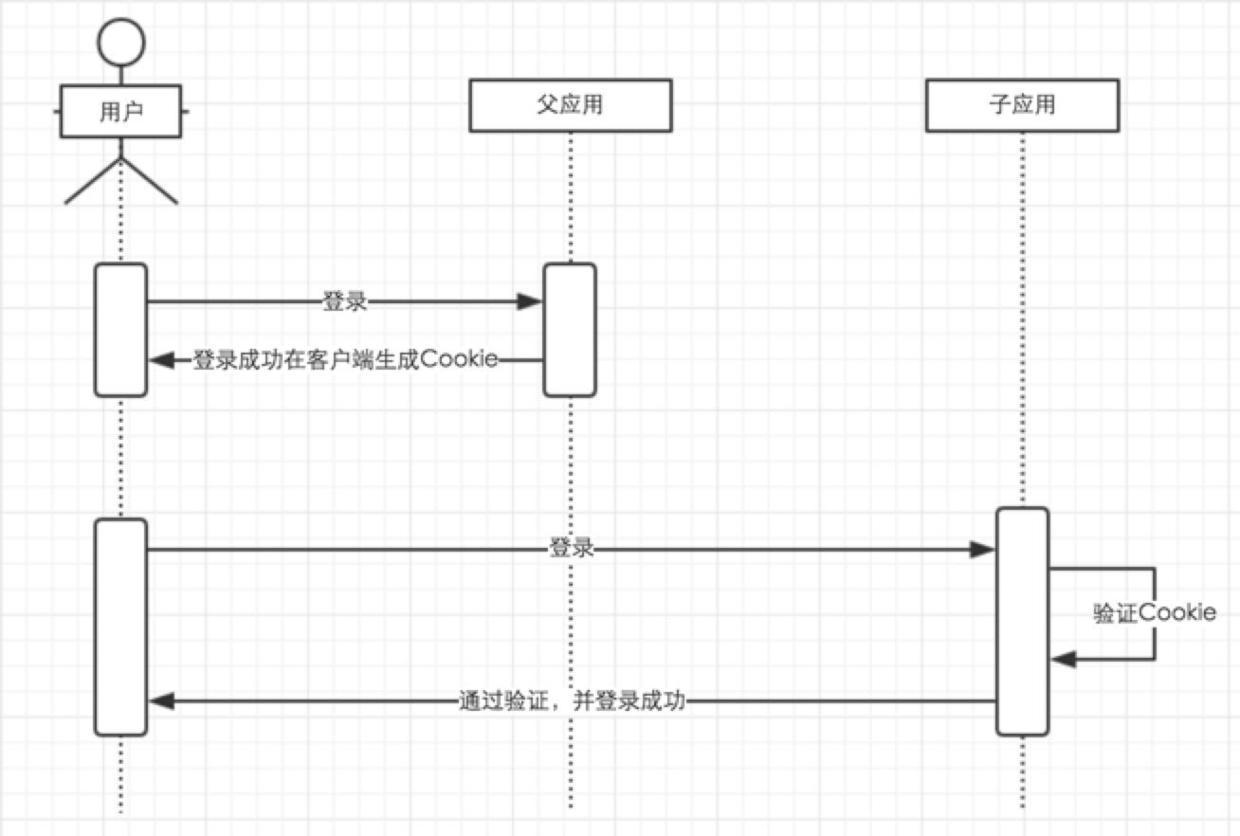
# 单点登录

单点登录英文全称Single Sign On，简称就是SSO。它的解释是：**在多个应用系统中，只需要登录一次，就可以访问其他相互信任的应用系统。**

**SSO只有登录模块**，没有其他的业务模块，当其他子系统需要登录时，将跳到SSO系统，SSO系统完成登录，其他的应用系统也就随之登录了。这完全符合我们对单点登录（SSO）的定义。

## 普通的登录认证机制



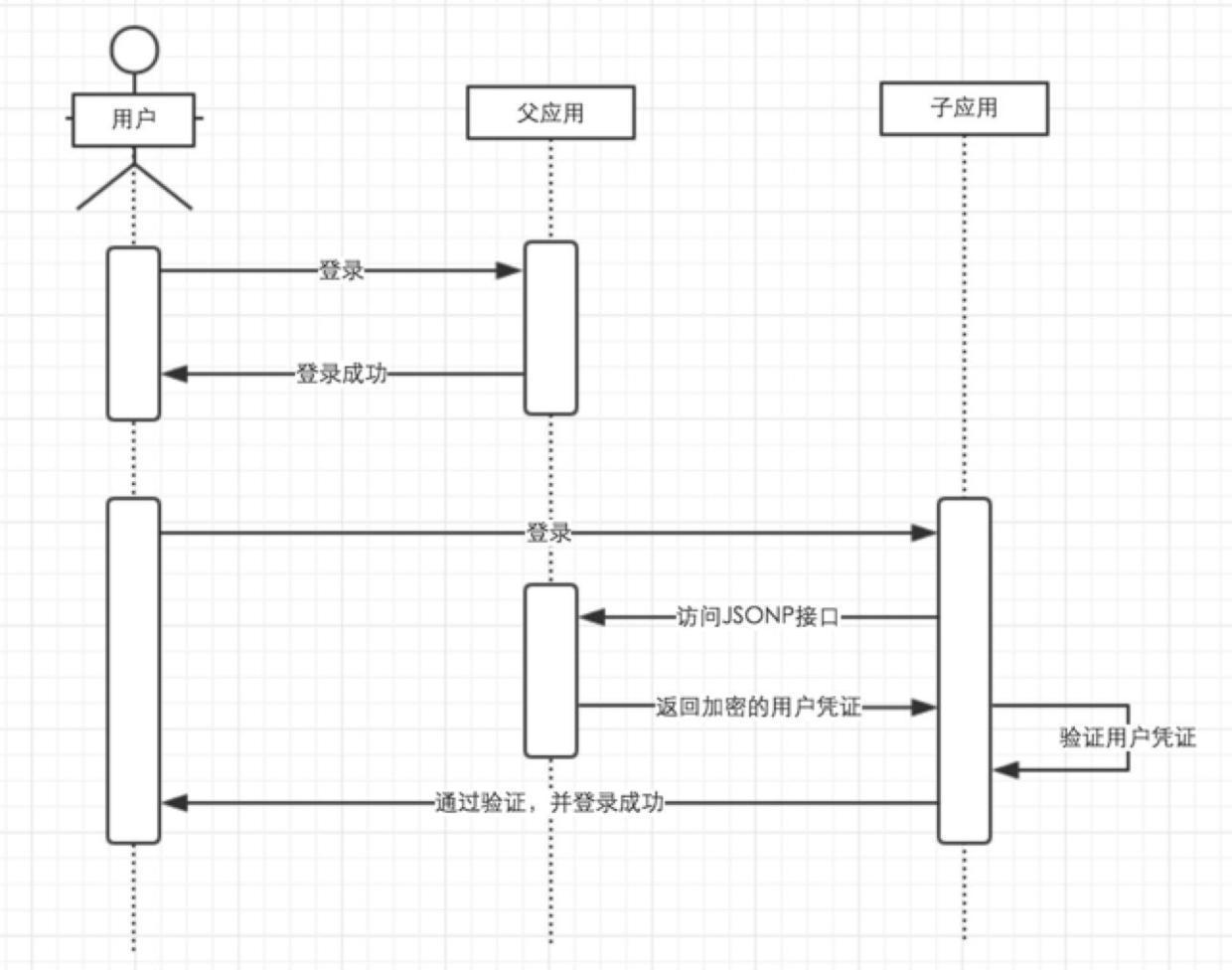
我们在浏览器（Browser）中访问一个应用，这个应用需要登录，我们填写完用户名和密码后，完成登录认证。这时，我们在这个用户的session中标记登录状态为yes（已登录），同时在浏览器（Browser）中写入Cookie，这个Cookie是这个用户的唯一标识。下次我们再访问这个应用的时候，请求中会带上这个Cookie，服务端会根据这个Cookie找到对应的session，通过session来判断这个用户是否登录。如果不做特殊配置，这个Cookie的名字叫做jsessionid，值在服务端（server）是唯一的。

不难发现以上方式把信任存储在客户端的Cookie中，这种方式很容易令人质疑：

* Cookie不安全
* **不能跨域实现免登**

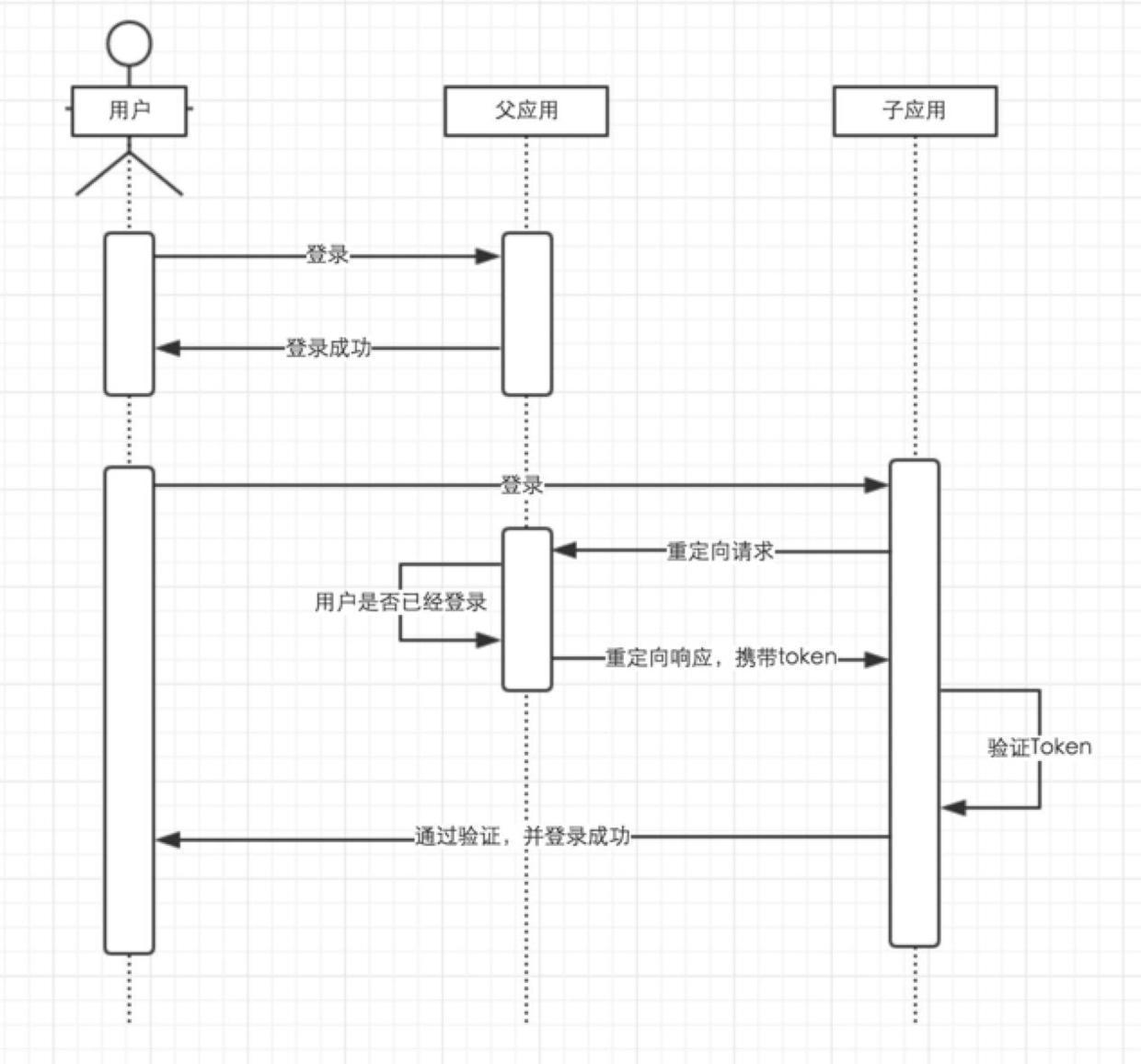
对于第一个问题，通过加密Cookie可以保证安全性，当然这是在源代码不泄露的前提下。如果Cookie的加密算法泄露，攻击者通过伪造Cookie则可以伪造特定用户身份，这是很危险的。   
对于第二个问题，更是硬伤。

**通过JSONP实现**   
对于跨域问题，可以使用**JSONP实现**。   
用户在父应用中登录后，跟Session匹配的Cookie会存到客户端中，当用户需要登录子应用的时候，授权应用访问父应用提供的JSONP接口，并在请求中带上父应用域名下的Cookie，父应用接收到请求，验证用户的登录状态，返回加密的信息，子应用通过解析返回来的加密信息来验证用户，如果通过验证则登录用户。



这种方式虽然能解决跨域问题，但是安全性其实跟把信任存储到Cookie是差不多的。如果一旦加密算法泄露了，攻击者可以在本地建立一个实现了登录接口的假冒父应用，通过绑定Host来把子应用发起的请求指向本地的假冒父应用，并作出回应。   
因为攻击者完全可以按照加密算法来伪造响应请求，子应用接收到这个响应之后一样可以通过验证，并且登录特定用户。

**通过页面重定向的方式**   
最后一种介绍的方式，是通过父应用和子应用来回重定向中进行通信，实现信息的安全传递。   
父应用提供一个GET方式的登录接口，用户通过子应用重定向连接的方式访问这个接口，如果用户还没有登录，则返回一个的登录页面，用户输入账号密码进行登录。如果用户已经登录了，则生成加密的Token，并且重定向到子应用提供的验证Token的接口，通过解密和校验之后，子应用登录当前用户。



这种方式较前面两种方式，接解决了上面两种方法暴露出来的安全性问题和跨域的问题，但是并没有前面两种方式方便。   
安全与方便，本来就是一对矛盾。

## 同域下不同的子域的单点登录

**Cookie性质：**

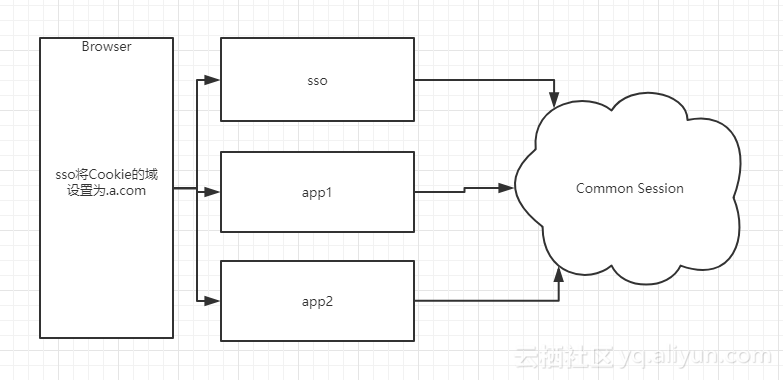
[www.onmpw.com/site1](http://www.onmpw.com/site1)  www.onmpw.com/site2

按照HTTP协议规定，两个站点是可以共享Cookie的。前提是这两个站点是在同一个域名下面（或者是二级域名也可）。这种情况是属于同域下的Cookie。浏览器会将Cookie以及该Cookie所属的域存在本地。当你对该域下的任何子站点进行访问的时候，浏览器都会将这些Cookie发送给站点系统。

一个企业一般情况下只有一个域名，通过二级域名区分不同的系统。比如我们有个域名叫做：a.com，同时有两个业务系统分别为：app1.a.com和app2.a.com。我们要做单点登录（SSO），需要一个登录系统，叫做：sso.a.com。

我们只要在sso.a.com登录，app1.a.com和app2.a.com就也登录了。通过上面的登陆认证机制，我们可以知道，在sso.a.com中登录了，其实是在**sso.a.com的服务端的session中记录了登录状态**，同时在浏览器端（Browser）的sso.a.com下写入了Cookie。那么我们怎么才能让app1.a.com和app2.a.com登录呢？这里有两个问题：

* Cookie是不能跨域的，我们Cookie的domain属性是sso.a.com，在给app1.a.com和app2.a.com发送请求是带不上的。
* sso、app1和app2是不同的应用，它们的session存在自己的应用内，是不共享的。



那么我们如何解决这两个问题呢？

（1）sso登录以后，可以将Cookie的域设置为顶域，即**.a.com**，这样所有子域的系统都可以访问到顶域的Cookie。我们在设置Cookie时，只能设置顶域和自己的域，不能设置其他的域。比如：我们不能在自己的系统中给baidu.com的域设置Cookie。

**setcookie**(‘username’,’ username’,null,’ .a.com’);

setcookie(‘password’,’pwd’,null,’ .a.com’);

浏览器会将cookie中的信息username和password附带在请求中一块儿发送到app2.a.com系统。这时该系统会先检查**session是否登录**，如果没有登录则验证**cookie中的username和password从而实现自动登录**。 登录成功以后再写session信息。以后的验证就用自己的session信息验证就可以了。

但是，这里存在一个问题就是app1.a.com系统退出以后，除了可以清除自身的session信息和所属域为.a.com的cookie的信息。它并不能清除sub2系统的session信息。那sub2仍然是登录状态。也就是说，**这种方式虽说可以实现单点登录，但是不能实现同时退出。**原因是，app1和app2虽说通过setcookie函数的设置可以共享cookie，**但是二者的sessionId是不同的，而且这个sessionId在浏览器中也是以cookie的形式存储的，不过它所属的域并不是.a.com。**也就是说二者的sessionId是不同的。

session.cookie\_domain = '**.a.com** '

经过以上设置，sub1和sub2系统就会使用相同的session信息了。这样既可以实现单点登录，也可以实现同时退出。

（2）Cookie的问题解决了，我们再来看看session的问题。我们在sso系统登录了，这时再访问app1，Cookie也带到了app1的服务端（Server），app1的服务端怎么找到这个Cookie对应的Session呢？这里就要把3个系统的Session共享，如图所示。共享Session的解决方案有很多，例如：Spring-Session。这样第2个问题也解决了。

同域下的单点登录就实现了，**但这还不是真正的单点登录。**

## 不同域之间如何实现单点登录

**总结**

起初，浏览器没有所属域为SSOsite的cookie信息。因此无论是点击任何站点的需要验证的界面都会跳转到登录页（这个过程是由程序内部重定向到SSOsite来检查是否存在cookie的）。一旦用户登录成功，所属域为SSOsite的，并且带有登录用户信息的cookie会被浏览器存储在本地。

然后，当用户再次访问需要验证的页面的时候，同样请求会在被重定向到SSOsite，并且浏览器会带上先前已经保存的cookie信息。SSOsite检索cookie，从中提取出用户token，并带着这个token重定向到最初请求的站点页面。然后该站点会通过web/WCF服务去验证token的合法性。然后将相应的页面发送给客户端。

一旦用户通过该单点登录模型登录到站点上，请求任何需要验证的页面都会内部重定向到SSOsite验证cookie和提取用户token，然后将请求的页面发送给浏览器输出。

**有的同学问我，SSO系统登录后，跳回原业务系统时，带了个参数ST，业务系统还要拿ST再次访问SSO进行验证，觉得这个步骤有点多余。他想SSO登录认证通过后，通过回调地址将用户信息返回给原业务系统，原业务系统直接设置登录状态，这样流程简单，也完成了登录，不是很好吗？**

**其实这样问题时很严重的，如果我在SSO没有登录，而是直接在浏览器中敲入回调的地址，并带上伪造的用户信息，是不是业务系统也认为登录了呢？这是很可怕的。单点登录，资源都在各个业务系统这边，不在SSO那一方。**

**用户在给SSO服务器提供了用户名密码后，作为业务系统并不知道这件事。 SSO随便给业务系统一个ST，那么业务系统是不能确定这个ST是用户伪造的，还是真的有效，所以要拿着这个ST去SSO服务器再问一下，这个用户给我的ST是否有效，是有效的我才能让这个用户访问。**