[**漫谈单点登录（SSO）**](http://www.cnblogs.com/EzrealLiu/p/5559255.html)

参考：<https://www.cnblogs.com/EzrealLiu/p/5559255.html>

**1. 摘要**

**（ 注意：请仔细看下摘要，留心此文是否是您的菜，若浪费宝贵时间，深感歉意！！！）**

SSO这一概念由来已久，网络上对应不同场景的成熟SSO解决方案比比皆是，从简单到复杂，各式各样应有尽有！开源的有OpenSSO、CAS ，微软的AD SSO，及基于kerberos 的SSO等等……这些优秀的解决方案尽显开发及使用者的逼格，当然需求所致无谓好坏高低，满足实际之需才是王道！

本文并不讨论上述提到的方案的整合使用、或者复杂场景如：安全、防火墙、N 多个系统层叠调用这种"巨型项目"里SSO的实现与使用，也并不涉及 C/S 、C/S+B/S 的SSO解决方案，仅关注B/S 上的SSO实现。虽是如此，然而万变不离其宗，这里我们将从一个简而小的登录场景去接触SSO的本质，描述如何原生态地自实现一个轻量、微核的SSO（本文不提供源码）。

文章将由浅入深地探讨SSO（单点登录），涉及SSO的定义、表现、原理、实现细节等方面的阐述，借助大家熟知的淘宝、天猫登录场景，通过对阿里登录的模仿实现，建立一个简单模型，然后不断由该模型进行迭代并对每一个迭代版本进行详细描述，最终得到一个支持跨域的SSO（ 力求条理清晰，层层递进，简单但有深度！！！！开始部分本着让即使从未听过SSO的同学也能够从抽象文字定义的概念印象过渡到具象的视觉认知这一宏（zhuang）伟（bi）理念入手，将会有很多浅显的描述，"老司机" 可以快速掠过）

**2. SSO简介**

**2.1 SSO定义**

SSO（ Single Sign-On ），中文意即单点登录，翻译得比较精简，个人觉得 Wiki 上的解释更细腻点—— SSO, is a property of access control of multiple related, but independent software systems. With this property a user logs with a single ID and password to gain access to connected system or systems without using different usernames or passwords, or in some configurations seamlessly sign on at each system.**( 单点登录是一种控制多个相关但彼此独立的系统的访问权限, 拥有这一权限的用户可以使用单一的ID和密码访问某个或多个系统从而避免使用不同的用户名或密码，或者通过某种配置无缝地登录每个系统 ).**注：系统，在本文特指WEB 应用或者WEB 服务；用户，下文也会称之为User；ID，用户标识；密码，本文也称其为口令，Password, Passcode 或者 Pin。

OK，从上面的定义中我们总结出 与 **SSO 交互的2个元素：1.  用户，2. 系统，它的特点是：一次登录，全部访问。**上面提到SSO是访问控制的一种，控制用户能否登录，即验证用户身份，而且是所有其它系统的身份验证都在它这里进行，那么我们是不是可以认为**SSO还是一个验证中心**。那么**从整个系统层面来看SSO，它的核心就是这3个元素了：1. 用户，2. 系统，3. 验证中心。**可能扯了那么多还是不足以形象地描述我们萌萌的SSO，呐，有图有真相：

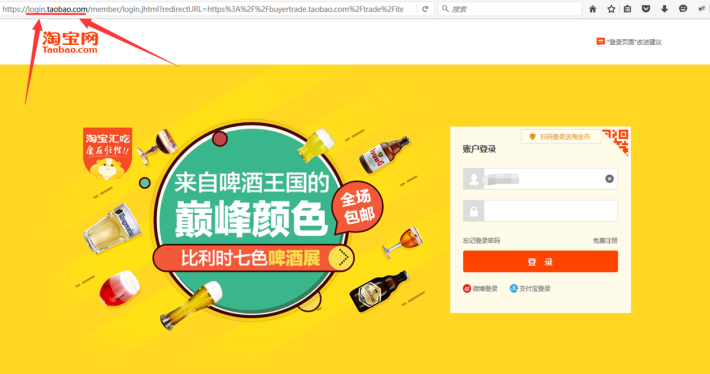


既然SSO这么棒，应该如何实现呢？

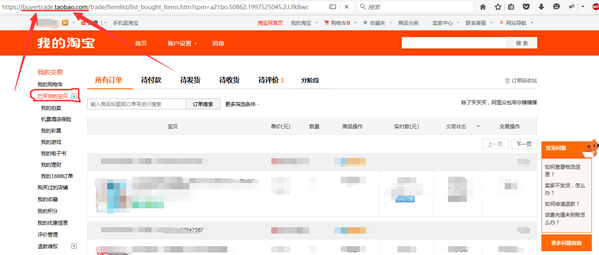
**2.2 SSO示例——淘宝、天猫的登录场景**

我们暂不考虑细节，先从SSO需要解决的问题入手：使用一个账户通过一次登录，即可在多个相关的系统之间来回访问，为了更加形象我们还是上图：**(多图预警)**

**登录页面，网址：login.taobao.com ….. 我将在 login.taobao.com 所指的系统进行登录**



**访问网站，第一张网址：buyertrade.taobao.com…. 访问 buyertrade.taobao.com所指的系统了；然后访问另一张网页网址为：favoriate.taobao.com， 访问favoriate.taobao.com 所指系统，两个系统的 Domain 是相同的，请注意这点；**





**接下来我再分别访问淘宝(**[**www.taobao.com**](http://www.taobao.com/)**)和天猫(**[**www.tmall.com**](http://www.tmall.com/)**)的首页 ，图中显示我仍旧是登录的( 注意：这里是不同的Domain下，系统之间的来回访问)**





可以看到，我除了在第一张网页图那里需要输入用户名（ID）和口令（password）进行登录，再访问其它相关系统时，从图2-5 中所有的访问操作，**无论域名相同还是不同我都不需要再登录了**，它们都知道我叫"望向明天"！对，没错，这就是SSO的作用：**一次登录，全部访问**，读者也可以尝试下看看是不是如此；

**3. SSO实现描述**

好，经过我上面一大段废话，基本上对SSO要解决什么问题有一个清晰的认识。现在我们自行脑（yi）补（yin）下SSO 的原理是什么样的。

1. 一个账户：嗯，规定所有系统统一使用相同账户，就能保证一个账户了；
2. 一次登录全部访问：通过SSO登录后，让其告知其它各个系统保存该用户的信息，用户就不用重复多次的登录了；

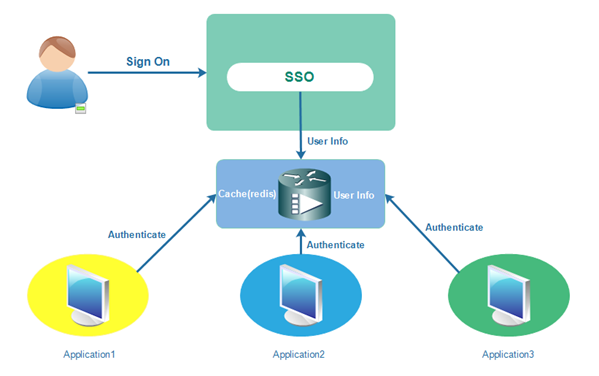
嗯，问题解决了，没错，就这样。

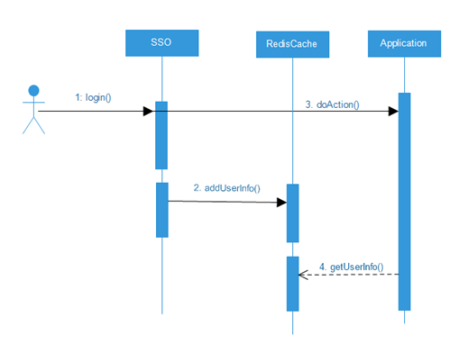
**3.1 方案1**

由上面的猜想可以得到第1个解决方案，记为方案1。这里对这个猜想做一点小小的优化，猜想中第2点 "各个系统保存" 好让人闹心，同一份数据保存多份，太浪费，这里我们把每个已登录的用户信息保存到公共缓存中。好，我们再来描述下这个方案：

1. User 发送登录请求给SSO，附上自己的 ID 和 password；
2. SSO验证成功将用户信息保存在公共缓存 Cache 中；
3. User每次发送请求给系统 Ai 时，将 ID 作为请求参数；
4. 系统 Aj 通过 请求中传过来的 User ID从公共缓存 Cache 中验证 User 是否登录，完成后续动作；

文字完了，接下来看看方案1的架构图和时序图：





**嗯，图文并茂的样子，难道就这么大功告成了？ 我们先把方案1中完成的第一版 SSO 记为SSO\_V1，接下来我们来好好地捋一捋。**

**3.2 方案2**

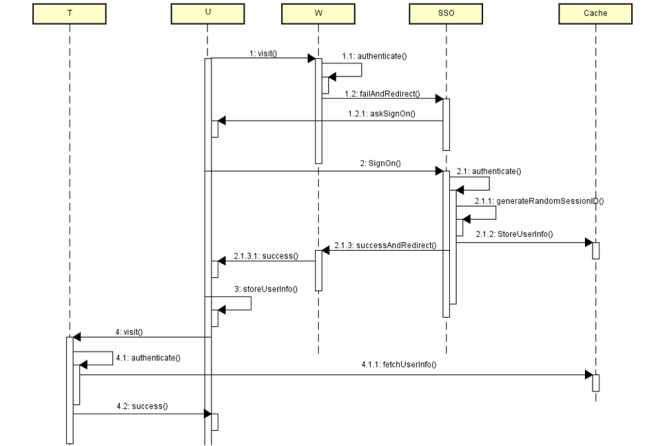
**SSO\_V1 貌似解决了问题，但是深入思考，细思极恐！因为这个设计有Bug**：每次传 ID 给服务Ai，但是这个ID 每次怎么获取来呢？登录SSO的时候，这倒没有问题，可以让用户填！但第2次请求是发给Ai中的某一个 Aj 时，ID 要怎么来（ 假设百度和新浪是相关但彼此独立的系统，登录百度后，再访问新浪时怎么让新浪取到与登录百度时一样的ID吧）？总不至于每次发请求时都要求用户填一遍ID 吧？

其实我们把 猜想 中最值得思考的问题之一忽略掉了：

**如何让SSO"告知"系统Ai，当前登录的User 的ID和password？**

这问题可以这样来描述：假设有W ( [www.weidai.com](http://www.weidai.com/) )和 T( trade.weidai.com ) 两个系统，W和T 都通过S (login.weidai.com) 系统登录，当由U访问W再转向S 完成登录后，怎样做才能使 U 访问T 时不需要再一次通过 S 进行登录验证？

对，如果你是WEB 开发的老司机，很自然你会想到用**cookie** ，即把用户信息（ 本文也会称之为UserInfo ）保存在**cookie** 当中，因为 无论W 、T 或者 S 它们的Domain是一样的——都是 weidai.com ——同一Domain，这有何用？用处就在于 W 、T 以及 S 可以共享此路径下的 **cookie**。这里，让我们优化的心再一次燃烧起来——直接保存用户的 ID 和 口令 对于我们这么有逼格，有追求的猿来说有点太不讲究——为什么呢？不太安全，**cookie** 中 最好保存一个 **公共Session ID（ 请和WEB 自己生成的Seesion ID进行区分 ）**，而我们的公共缓存 Cache 中保存的 UserInfo 是一个由 **公共Session ID为Key** ，**以包含用户标识和口令的数据结构为Value的Map。**最后附上这一流程的时序图及简要说明：



1. U访问W ，W进行验证，验证失败，跳转至SSO，要求U登录；
2. U通过SSO登录，SSO进行验证，成功并生成**SessionID**，随后将UserInfo（ **SessionID**、ID和口令）存储到公共缓存C 中，跳转至W（携带SessionID），并允许U访问W；
3. U保存UserInfo （ **SessionID** ） 至 **cookie** ;**（这里请将 U 看成一个浏览器，当下文有提到 U 保存XXX至Cookie时，读者请自行切换）**
4. U 再访问 T ( 并携带 在3 中保存至**cookie** 中的 UserInfo ) ，T从公共缓存中拉取UserInfo 进行验证，成功则允许访问；

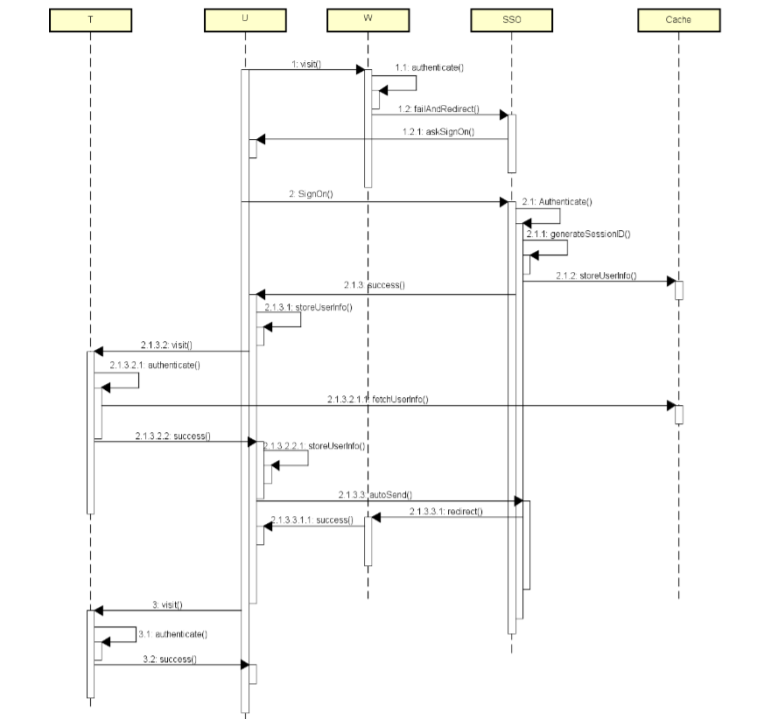
**嗯，又是图文并茂的样子，难道再一次大功告成？ 我们暂时把刚才的方案记为方案2，并把方案2中完成的升级版SSO记为SSO\_V2，接下来我们再来好好地捋一捋！**

**3.3 方案3**

**SSO\_V2 能够在 Domain 相同的情况下"完美"解决问题，但是在Domain不同的情况下怎么做到免登呢？**如上面图示淘宝（ [www.taobao.com](http://www.taobao.com/) ）和天猫（ [www.tmall.com](http://www.tmall.com/) ）若采用SSO\_V2 肯定无法做到免登的，因为我们知道当访问天猫时（Domain 为tmall.com ），淘宝（ Domain 为 taobao.com ）下的 cookie 是无法随访问请求一并传给与天猫相关的系统的。**所以问题变成，怎么让不同Domain下的系统也"知晓"用户已经登录的实事？**

在我们提出SSO\_V3前，**我们先看看SSO 本质是什么？通过这么多的文字描述、样图解释，我们可以看到，要让用户"一次登录，全部访问"无非就是让所有的系统共享"一份"（相同）已验证的、安全可靠的验证信息。**所以问题就可以转化为：不同Domain下的系统如何共享一份的验证信息？既然Domain无法做到交叉访问，那我们可以让不同Domain下的WEB应用持有相同的验证信息，这在效果上不就是一份吗！所以最终要解决的问题就是：**SSO系统如何使不同的 Domain 拥有一份相同的cookie？ —— 让SSO在用户进行登录时再去访问其它域下的系统，并让各个系统保存一样的验证信息，这样不同域下就会有同一份cookie。**

**以下是SSO\_V3的时序图和文字说明，**这里我们假设 SSO 的Domain 为 SD，T 的 Domain 为 TD：



1. U第一次访问W，W验证失败，跳转至SSO要求U进行登录验证；
2. 登录并使各不同Domain下：
   1. U 给SSO发送登录请求，SSO验证成功，生成SessionID 并保存UserInfo；
   2. 返回给U的Response 将 UserInfo 存放至cookie中，Domain为SD；
   3. 将 2 中 cookie 内容作为query parameter 重定向至T，T验证后成功返回给U，也在Response 中设置 cookie；Domain为TD；
   4. U自动访问SSO，SSO将请求重定向至W，完成U对W 的访问；
3. U 再访问 T，验证成功并允许U进行访问；

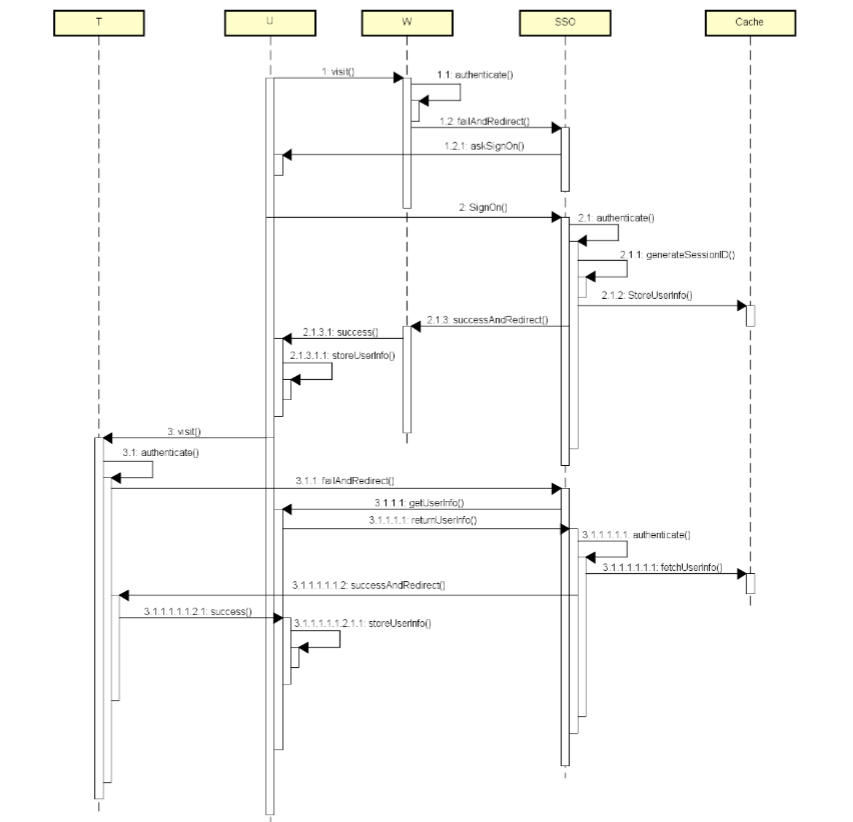
**嗯，还是图文并茂的样子，这下是不是可以完事了呢？我们还是把刚才的方案记为方案3，并把方案3中完成的升级版SSO记为SSO\_V3，然后还是来好好地捋一捋！**

**3.4 方案4**

再细细的考虑下SSO\_V3的实现方式，有没有感觉它哪里有点不对劲（ 思维一直跟着我来走，是不是被绕晕了，想发现不对劲，怎么可能）？ SSO\_V3 使不同 Domain 获取相同的cookie 拷贝时，表面是在U处主动发出向Ｔ的请求（其实是被动）， 但实际上是 SSO 返回给 U 的页面自动完成的（通过 JS、通过页面自动跳转、iframe都可以实现）。**所以方案SSO\_V3要求SSO 预先知道有哪些系统是跨域的！！！而且它还有一个很严重的问题：假如与SSO相关但相互独立的系统中，有 20+ 需要跨域才能访问，而SSO要在用户登录时完成20+跳转……现在你是不是要呵呵了？**貌似完美解决跨域的SSO\_V3 竟然如此有问题，有没有心好塞！

 SSO\_V3 解决的核心问题是：针对跨域的系统，各系统间如何保证获取到的 **验证信息**是一致的，解决方法即是**在用户第一次登录时把验证信息复制给所有跨域的系统。**这种方案在跨域系统少的情况下倒是不需要有太多担心，但是当跨域系统多、且验证步骤比较复杂时用户将会卡在登录界面，最后不得不怒关页面！所以当理清这些逻辑，很自然就会想到接下来要如何对SSO\_V3进行优化。**核心思想就是：既然一次性解决会有问题，那就分多次解决！**简单的描述下我们将要看到的SSO\_V4，用户登录后，当第一次访问跨域系统W 时，跳到SSO，复制一份至W的cookie中，过程结束；当访问T时，重复该处理动作。

以下为SSO\_V4的时序图及简要说明：

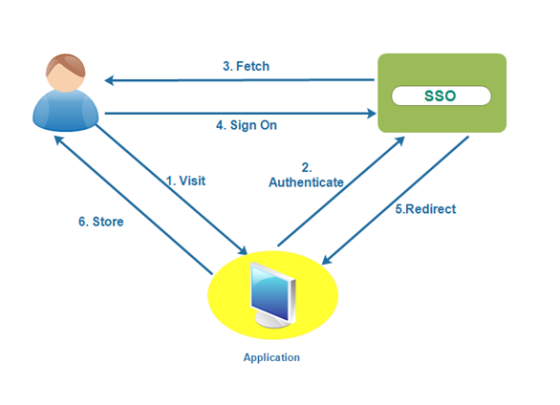


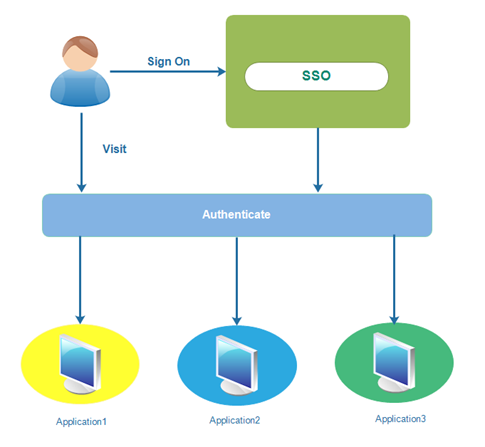
1. 用户U访问W ，W进行验证，验证失败，跳转至SSO，要求U登录；
2. U通过SSO登录，SSO进行验证，成功并生成**SessionID**，随后将UserInfo（ **SessionID**、ID和口令）存储到公共缓存C 中，跳转至W（携带SessionID），并允许U访问W；U保存UserInfo （ **SessionID** ） 至 **cookie；**
3. U访问T，T 进行验证，失败跳转至SSO，SSO将触发U请求SSO将验证信息随请求一并发给SSO，经SSO验证成功跳转至Ｔ，允许U对T 的访问；使U保存UserInfo（ SessionID）至cookie；

**3.5 小结**

其实我们通过上面的实用版（SSO\_V2，SSO\_V3，SSO\_V4）SSO，可以看到除了用户的第一次登录某个应用相对来说比较特殊，其它处理都是一致的。所以当我们抛去细节之后，不仿这样联想SSO的实现：**完成登录逻辑并使各系统共享验证信息和验证逻辑**，从这个层次去看SSO，我们发现它其实只负责用户登录和身份验证这2、3个点。

下面是用户第一次登录及SSO与其它系统交互的简图：





**4. 设计与实现**

**4.1 验证信息的安全考虑**

第3部分中的身份验证和验证信息方面都做得比较简单，在实际项目中不可能如此使用！在此提出一个方案以供参考（这也是比较流行的一种）。

1. 使用 HTTPS 进行用户登录；
2. 为每个用户生成一个对称密钥Ku；
3. 验证信息由"ID"+ "password"+ SessionID 组成，当然你可以按需设置，比如再加个IP 地址……
4. 存储在cookie 中的验证信息，ID 和口令部分经由用户密钥Ku和SSO公钥处理后在存放至"客户端"；

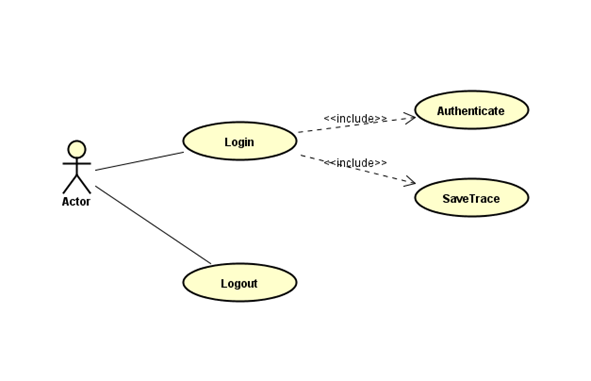
这样处理后相信能够满足大部分应用的需求了！

**4.2 SSO的概要设计**

**4.2.1 整体思路**

SSO这一理念到目前为止已经非常成熟，关于它的各种设计、设置都可以定制一套标准了。**然而由于SSO与用户有强关联，所以很多设计在最初时往往会把SSO设计成一个用户管理系统，而使得SSO与业务耦合，随着业务的不断变化和演进，底层数据结构、接口不断的复杂化，又反过来使得上层服务的架构设计变得尴尬。**

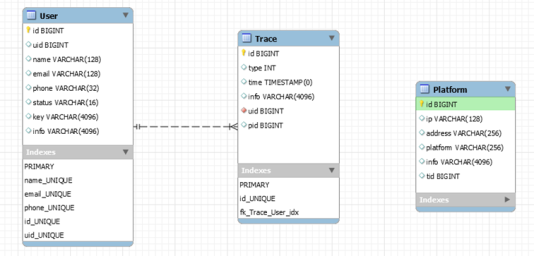
**若做更进一层的抽象和划分，SSO只需负责登录这单一功能即可，设计上满足单一职责原则**[**[1]**](https://en.wikipedia.org/wiki/Single_responsibility_principle)**，加上几乎所有网站的登录都大同小异(可能登录界面会变幻无常)且不与业务有过多牵连，这又使得SSO与业务完全分离，无论将来业务怎样演进，产品如何迭代，SSO作为底层应用可以以不变应万变。Really? All problem in computer science can be solved with another level of indirection，except of course for the problem of too many indirections.**[**[2]**](https://en.wikipedia.org/wiki/David_Wheeler_%28British_computer_scientist%29)如何在设计中做到复杂与简洁的平衡，需要根据实际情境深度地考量，这可以扯出长篇大论了（按下不表），我们的SSO姑且就搞这几个功能：登录、记录轨迹、登出，以下是用例图：



第3节第5部分有提到"登录交由SSO完成，各系统共享一套验证逻辑"，很自然的验证这一逻辑对SSO也是必须的，在此就由SSO来完成，其它系统只需将其配置到各自系统里即可。再加上SSO是用户"做案的第一现场"，所以记录用户登录信息的事也很自然的就让SSO给干起来了，而且这一功能不仅能够让用户感受到我们对客户的用心，同时也为后期数据分析业务提供数据源！

**4.2.2 数据表设计**

经过上面的讨论，我们着手思考SSO的数据结构——数据表设计（个人认为面向对象编程中数据结构的优劣基本决定整个应用的质量）。从SSO 功能简单及其微服务的定位，SSO的表应该简洁、单一，上层服务若需要对其进行扩展，只需要对基本表进行外键引用即可！这里我们暂时只用3张表，分别为User、Trace（用户轨迹表）和使用平台表，图示与描述如下：



用户表：User

1. uid 用户唯一标识，（ varchar 是否有更好）
2. name ：账号，可以唯一标识用户，email，phone等都唯一标识用户；
3. status：用户状态；（冻结，已删除……）；
4. key ：用户密钥；
5. info：扩展字段，用以应变需求；

用户轨迹表：Trace

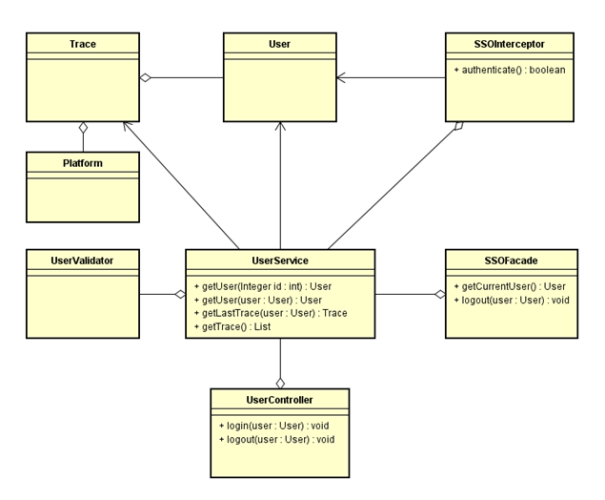
1. type ：轨迹类型，（删除，登录，登出，修改……）；
2. time ：操作时间；
3. info同上，uid 用户表外键，pid 为Platform的外键；

使用平台表：Platform

1. ip：用户登录ip
2. address：用户登录地址，可由IP 解析得到，（手机端可以使用GPS）；
3. platform：使用平台的信息，将在请求的head上得到；
4. info同上，tid 表示Trace 表的外键；

**4.2.3 简要类设计**

通过上面的整体思路及数据结构的定型，我们可以继续铺开将SSO要涉及到的一些主体类及主要方法定义好，仍旧上图：



写到这里，对于这个图示就不再做过多解释，大家基本可以开始做各种各样的脑补了！额，仅说小小的一个点：验证由Interceptor实现，这样验证逻辑则可以以插件形式配置到其它系统，实现所有系统共享一套验证逻辑，当然你也可以根据具体情况做成Filter，看个人爱好; 访问这方面交给第三方处理，比如由Shiro、Spring Security等来完成……酱紫，结束！