前端常见登录实现方案 + 单点登录方案

> 链接：https://juejin.cn/post/6933115003327217671#comment

登录是每个网站中都会用到的一个必备功能，但是如何实现一个优秀的登录功能，如何根据自己的项目来选择一个适合自己的登录方案？今天我们就来介绍几种常用的登录方案。

- Cookie + Session 登录
- Token 登录
- SSO 单点登录
- OAuth 第三方登录

一、Cookie + Session 登录

HTTP 是一种无状态的协议，客户端每次发送请求时，首先要和服务器端建立一个连接，在请求完成后又会断开这个连接。这种方式可以节省传输时占用的连接资源，但同时也存在一个问题：每次请求都是独立的，服务器端无法判断本次请求和上一次请求是否来自同一个用户，进而也就无法判断用户的登录状态。

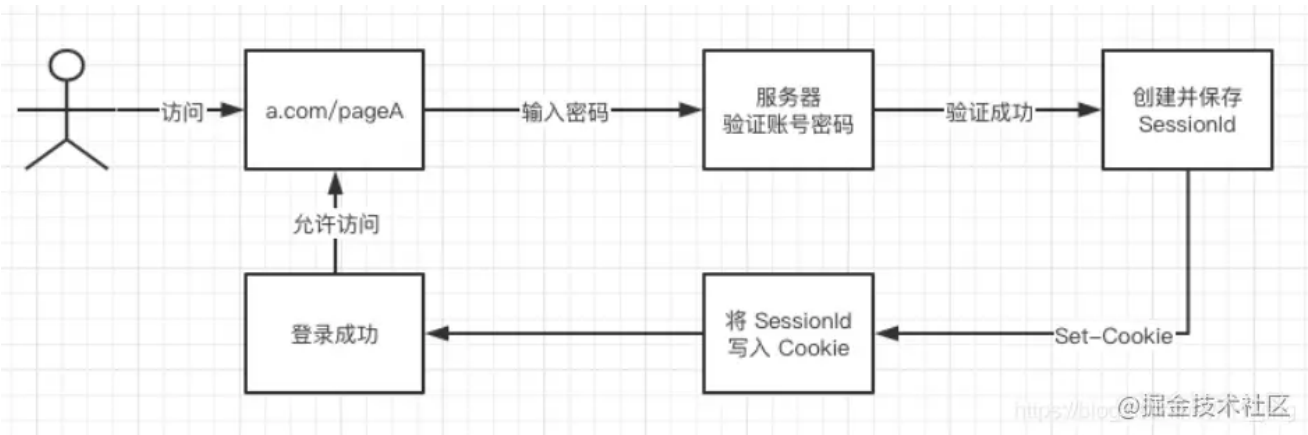
> 为了解决 HTTP 无状态的问题，Lou Montulli 在 1994 年的时候，推出了 Cookie。 Cookie 是服务器端发送给客户端的一段特殊信息，这些信息以文本的方式存放在客户端，客户端每次向服务器端发送请求时都会带上这些特殊信息。

在 B/S 系统中，登录功能通常都是基于 `Cookie` 来实现的。当用户登录成功后，一般会将登录状态记录到 `Session` 中。要实现服务端对客户端的登录信息进行验证都，需要在客户端保存一些信息（`SessionId`），并要求客户端在之后的每次请求中携带它们。在这样的场景下，使用 `Cookie` 无疑是最方便的，因此我们一般都会将 `Session` 的 `Id` 保存到 `Cookie` 中，当服务端收到请求后，通过验证 `Cookie` 中的信息来判断用户是否登录 。

Cookie + Session 实现流程

`Cookie` + `Session` 的登录方式是目前最经典的一种登录方式，现在仍然有大量的企业在使用。

用户初次登录时：

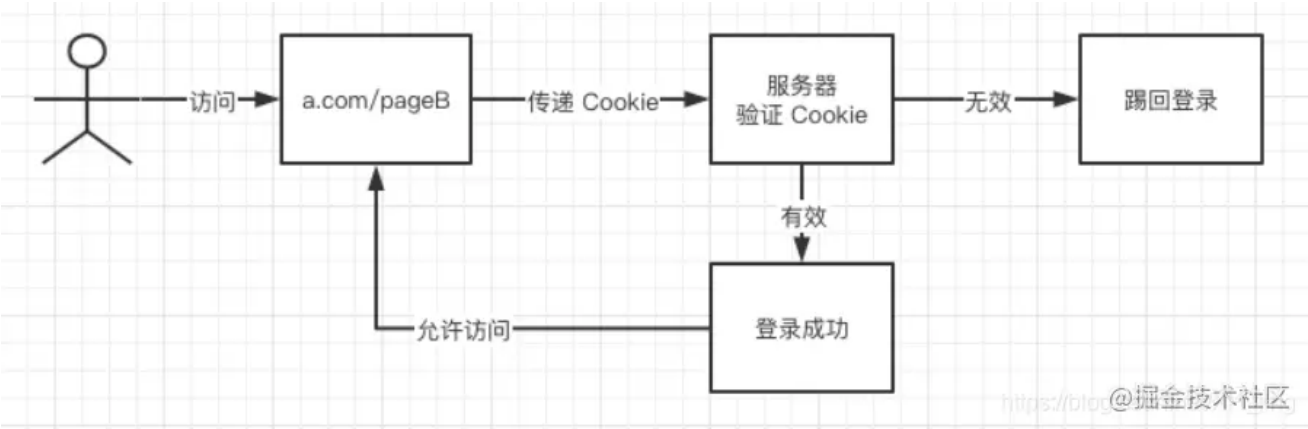


1. 用户访问 `a.com/pageA`，并输入密码登录。

2. 服务器验证密码无误后，会创建 `SessionId`，并将它保存起来。
3. 服务器端响应这个 HTTP 请求，并通过 `Set-Cookie` 头信息，将 `SessionId` 写入 `Cookie` 中。

> 服务器端的 `SessionId` 可能存放在很多地方，例如：内存、文件、数据库等。

第一次登录完成之后，后续的访问就可以直接使用 `Cookie` 进行身份验证了：



1. 用户访问 `a.com/pageB` 页面时，会自动带上第一次登录时写入的 `Cookie`。
2. 服务器端比对 `Cookie` 中的 `SessionId` 和保存在服务器端的 `SessionId` 是否一致。
3. 如果一致，则身份验证成功，访问页面；如果无效，则需要用户重新登录。

小结

虽然我们可以使用 `Cookie` + `Session` 的方式完成了登录验证，但仍然存在一些问题：

1. 由于服务器端需要对接大量的客户端，也就需要存放大量的 `SessionId`，这样会导致服务器压力过大。
2. 如果服务器端是一个集群，为了同步登录态，需要将 `SessionId` 同步到每一台机器上，无形中增加了服务器端维护成本。
3. 由于 `SessionId` 存放在 `Cookie` 中，所以无法避免 `CSRF` 攻击。

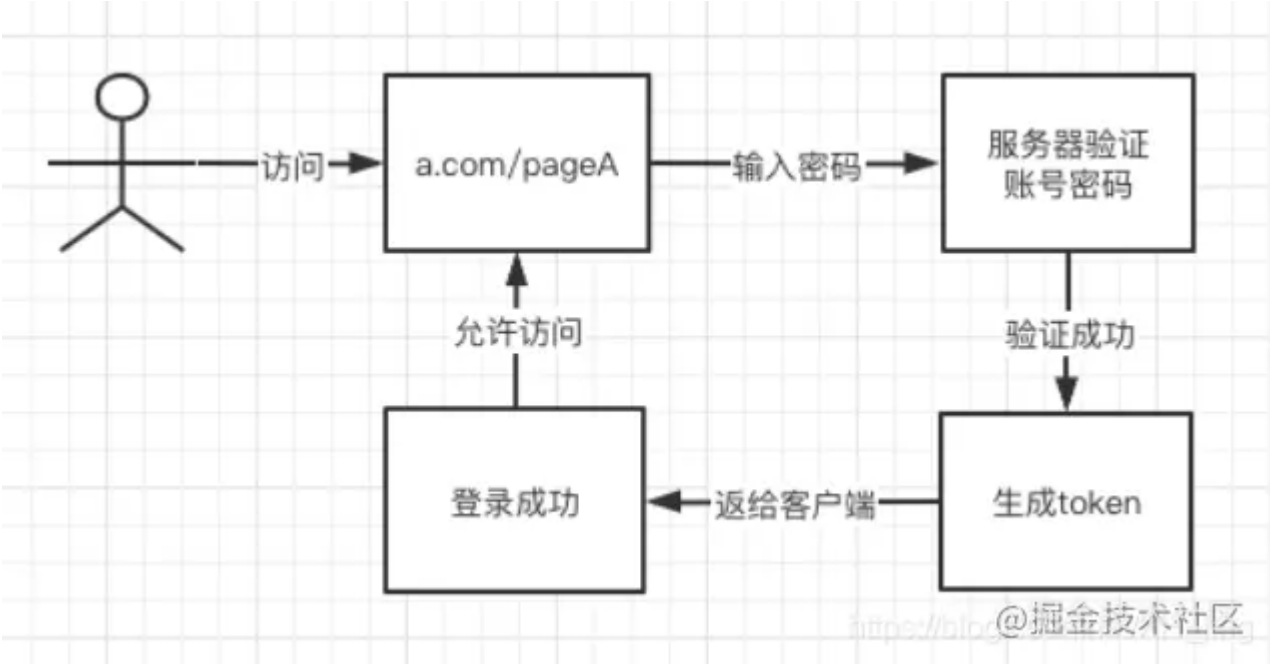
二、Token 登录

为了解决 `Cookie` + `Session` 机制暴露出的诸多问题，我们可以使用 `Token` 的登录方式。

> Token 是通过服务端生成的一串字符串，以作为客户端请求的一个令牌。当第一次登录后，服务器会生成一个 Token 并返回给客户端，客户端后续访问时，只需带上这个 Token 即可完成身份认证。

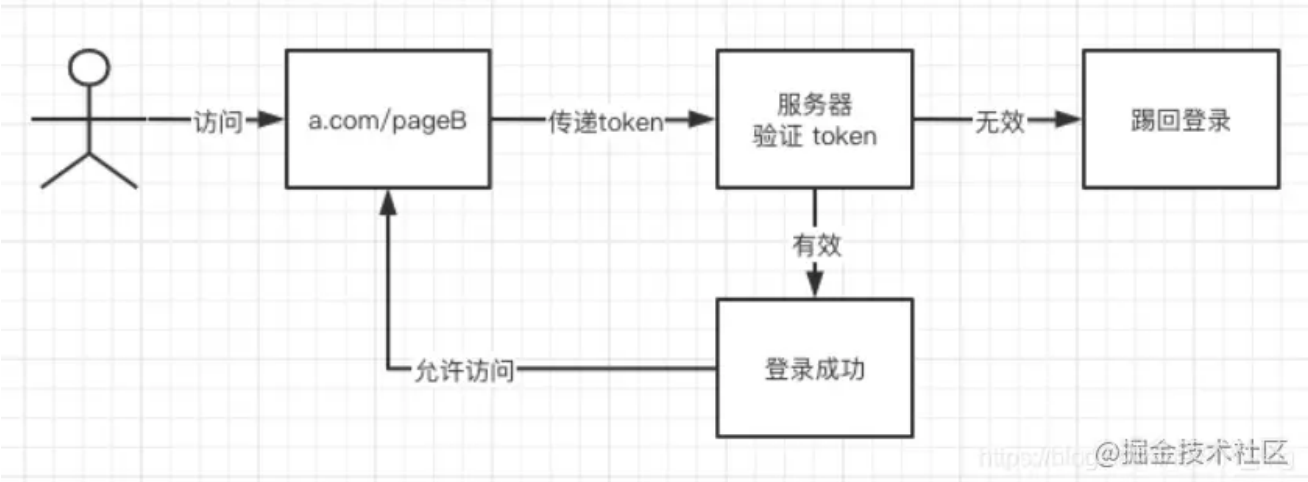
Token 机制实现流程

用户首次登录时：



1. 用户访问 `a.com/pageA`，输入账号密码，并点击登录。
2. 服务器端验证账号密码无误，创建 `Token`。
3. 服务器端将 `Token` 返回给客户端，由**客户端自由保存。**

后续页面访问时：



1. 用户访问 `a.com/pageB` 时，带上第一次登录时获取的 `Token`。
2. 服务器端验证该 `Token`，有效则身份验证成功，无效则踢回重新的登录。

Token 生成方式

最常见的 `Token` 生成方式是使用 `JWT`（`Json Web Token`），它是一种简洁的、自包含的方法，用于通信双方之间以 JSON 对象的形式安全的传递信息。

使用 `Token` 后，服务器端并不会存储 `Token`，那怎么判断客户端发过来的 `Token` 是合法有效的呢?

答案其实就在 `Token` 字符串中，其实 `Token` 并不是一串杂乱无章的字符串，而是通过多种算法拼接组合而成的字符串。

`JWT` 算法主要分为 3 个部分：`header`（头信息），`playload`（消息体），`signature`（签名）。

- `header` 部分指定了该 `JWT` 使用的签名算法；
- `playload` 部分表明了 `JWT` 的意图；
- `signature` 部分为 `JWT` 的签名，主要为了让 `JWT` 不能被随意篡改。

关于 `JWT`，这里简单说明一下，具体细节大家可以去看一下 JWT 官网。

小结

根据上面的案例，我们可以分析出 `Token` 的优缺点：

- 服务器端不需要存放 `Token`，所以不会对服务器端造成压力，即使是服务器集群，也不需要增加维护成本。
- `Token` 可以存放在前端任何地方，可以不用保存在 `Cookie` 中，提升了页面的安全性。
- `Token` 下发之后，只要在生效时间之内，就一直有效，但是如果服务器端想收回此 `Token` 的权限，并不容易。

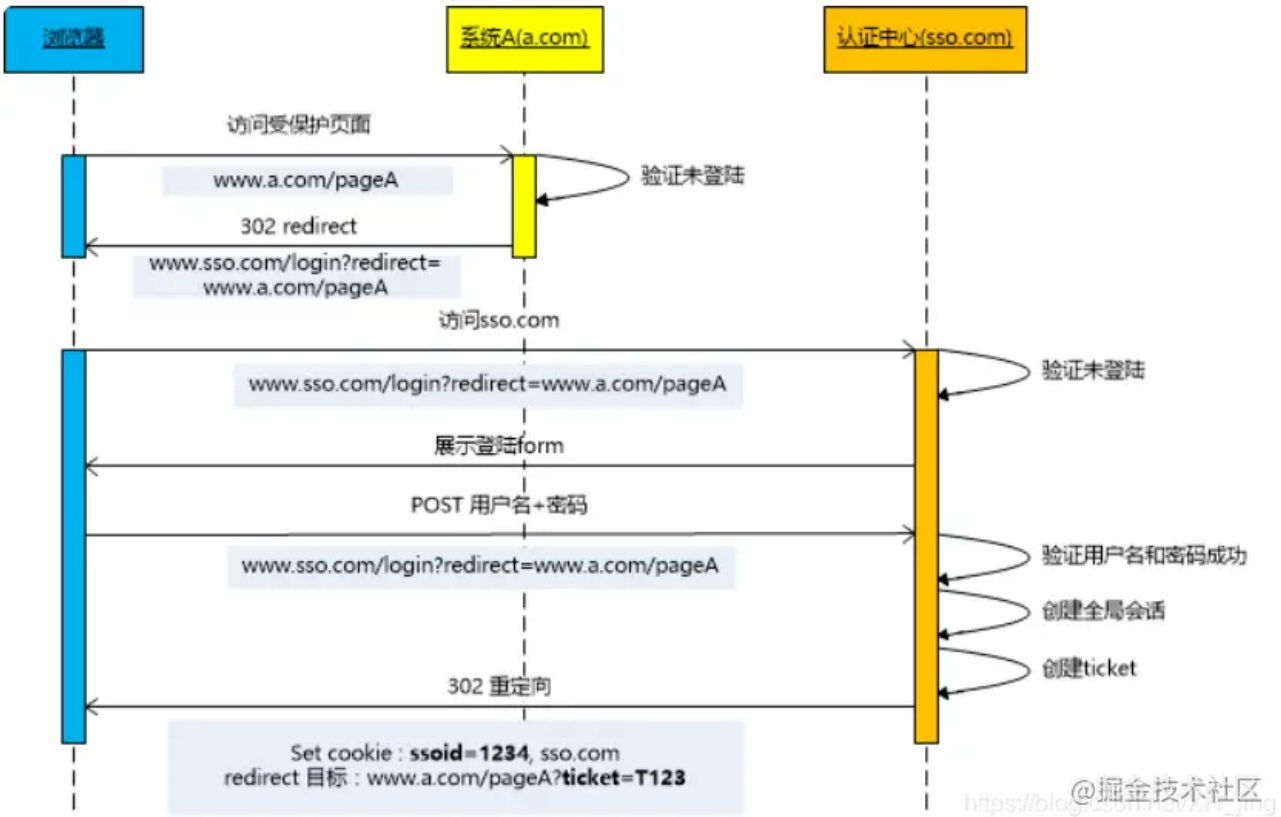
有了 `Token` 之后，登录方式已经变得非常高效。

三、SSO 单点登录

单点登录是指在同一帐号平台下的多个应用系统中，用户只需登录一次，即可访问所有相互信任的应用系统。本质就是在多个应用系统中共享登录状态。举例来说，百度贴吧和百度地图是百度公司旗下的两个不同的应用系统，如果用户在百度贴吧登录过之后，当他访问百度地图时无需再次登录，那么就说明百度贴吧和百度地图之间实现了单点登录。

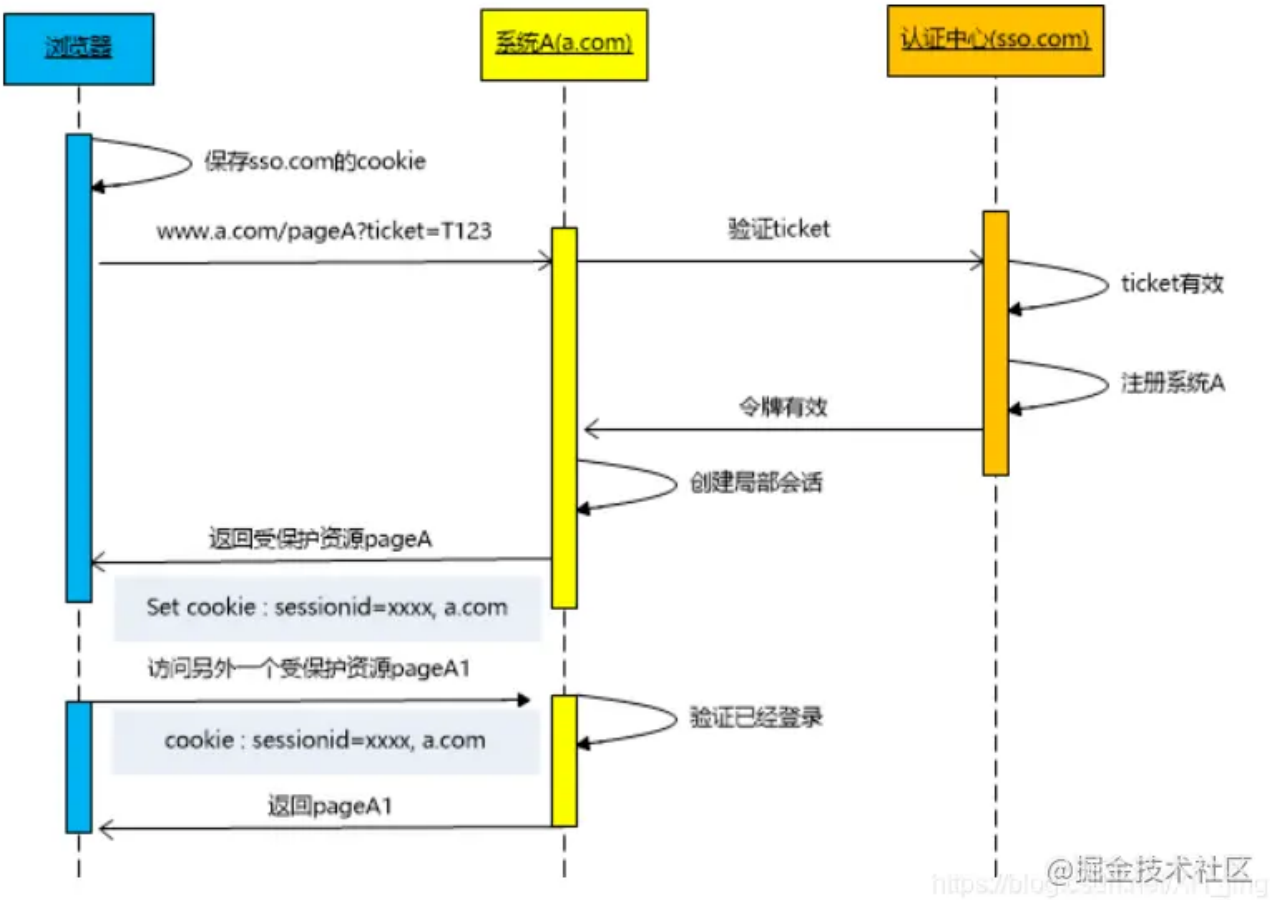
SSO 机制实现流程

用户首次访问时，需要在认证中心登录：



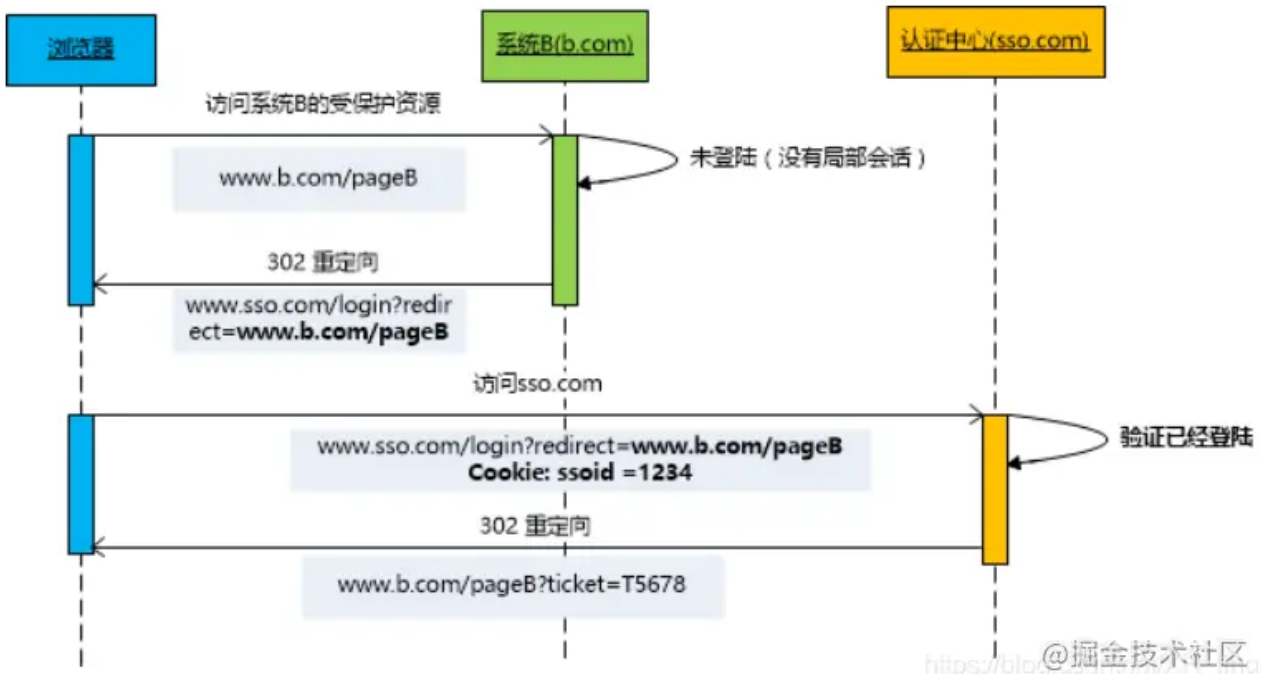
1. 用户访问网站 `a.com` 下的 `pageA` 页面。
2. 由于没有登录，则会重定向到认证中心，并带上回调地址 `www.sso.com?return_uri=a.com/pageA`，以便登录后直接进入对应页面。
3. 用户在认证中心输入账号密码，提交登录。
4. 认证中心验证账号密码有效，然后重定向 `a.com?ticket=123` 带上授权码 `ticket`，并将认证中心 `sso.com` 的登录态写入 `Cookie`。
5. 在 `a.com` 服务器中，拿着 `ticket` 向认证中心确认，授权码 `ticket` 真实有效。
6. 验证成功后，服务器将登录信息写入 `Cookie`（此时客户端有 2 个 `Cookie` 分别存有 `a.com` 和 `sso.com` 的登录态）。

认证中心登录完成之后，继续访问 `a.com` 下的其他页面：



这个时候，由于 `a.com` 存在已登录的 `Cookie` 信息，所以服务器端直接认证成功。

如果认证中心登录完成之后，访问 `b.com` 下的页面：



这个时候，由于认证中心存在之前登录过的 `Cookie`，所以也不用再次输入账号密码，直接返回第 4 步，下发 `ticket` 给 `b.com` 即可。

SSO 机制实现方式

单点登录主要有三种实现方式：

1. 父域 Cookie
2. 认证中心
3. LocalStorage 跨域

一般情况下，用户的登录状态是记录在 `Session` 中的，要实现共享登录状态，就要先共享 `Session`，但是由于不同的应用系统有着不同的域名，尽管 `Session` 共享了，但是由于 `SessionId` 是往往保存在浏览器 `Cookie` 中的，因此存在作用域的限制，无法跨域名传递，也就是说当用户在 `a.com` 中登录后，`Session Id` 仅在浏览器访问 `a.com` 时才会自动在请求头中携带，而当浏览器访问 `b.com` 时，`Session Id` 是不会被带过去的。实现单点登录的关键在于，如何让 `Session Id`（或 Token）在多个域中共享。

1. 父域 Cookie

`Cookie` 的作用域由 `domain` 属性和 `path` 属性共同决定。`domain` 属性的有效值为当前域或其父域的域名/IP地址，在 Tomcat 中，`domain` 属性默认为当前域的域名/IP地址。`path` 属性的有效值是以“`/`”开头的路径，在 Tomcat 中，`path` 属性默认为当前 Web 应用的上下文路径。

如果将 `Cookie` 的 `domain` 属性设置为当前域的父域，那么就认为它是父域 `Cookie`。`Cookie` 有一个特点，即父域中的 `Cookie` 被子域所共享，也就是说，子域会自动继承父域中的 `Cookie`。

利用 `Cookie` 的这个特点，可以将 `Session Id`（或 `Token`）保存到父域中就可以了。我们只需要将 `Cookie` 的 `domain` 属性设置为父域的域名（主域名），同时将 `Cookie` 的 `path` 属性设置为根路径，这样所有的子域应用就都可以访问到这个 `Cookie` 了。不过这要求应用系统的域名需建立在一个共同的主域名之下，如 tieba.baidu.com 和 map.baidu.com，它们都建立在 baidu.com 这个主域名之下，那么它们就可以通过这种方式来实现单点登录。

总结：此种实现方式比较简单，但不支持跨主域名。

2. 认证中心

我们可以部署一个认证中心，认证中心就是一个专门负责处理登录请求的独立的 Web 服务。

用户统一在认证中心进行登录，登录成功后，认证中心记录用户的登录状态，并将 `Token` 写入 `Cookie`。（注意这个 `Cookie` 是认证中心的，应用系统是访问不到的）

应用系统检查当前请求有没有 `Token`，如果没有，说明用户在当前系统中尚未登录，那么就将页面跳转至认证中心进行登录。由于这个操作会将认证中心的 `Cookie` 自动带过去，因此，认证中心能够根据 `Cookie` 知道用户是否已经登录过了。如果认证中心发现用户尚未登录，则返回登录页面，等待用户登录，如果发现用户已经登录过了，就不会让用户再次登录了，而是会跳转回目标 URL，并在跳转前生成一个 `Token`，拼接在目标 URL 的后面，回传给目标应用系统。

应用系统拿到 `Token` 之后，还需要向认证中心确认下 `Token` 的合法性，防止用户伪造。确认无误后，应用系统记录用户的登录状态，并将 `Token` 写入 `Cookie`，然后给本次访问放行。（这个 `Cookie` 是当前应用系统的，其他应用系统是访问不到的）当用户再次访问当前应用系统时，就会自动带上这个 `Token`，应用系统验证 `Token` 发现用户已登录，于是就不会有认证中心什么事了。

总结：此种实现方式相对复杂，支持跨域，扩展性好，是单点登录的标准做法。

3. LocalStorage 跨域

单点登录的关键在于，如何让 `Session Id` （或 `Token`）在多个域中共享。但是 `Cookie` 是不支持跨主域名的，而且浏览器对 `Cookie` 的跨域限制越来越严格。

在前后端分离的情况下，完全可以不使用 `Cookie`，我们可以选择将 `Session Id` （或 `Token` ）保存到浏览器的 `LocalStorage` 中，让前端在每次向后端发送请求时，主动将 `LocalStorage` 的数据传递给服务端。这些都是由前端来控制的，后端需要做的仅仅是在用户登录成功后，将 `Session Id` （或 `Token` ）放在响应体中传递给前端。

在这样的场景下，单点登录完全可以在前端实现。前端拿到 `Session Id` （或 `Token` ）后，除了将它写入自己的 `LocalStorage` 中之外，还可以通过特殊手段将它写入多个其他域下的 `LocalStorage` 中。

总结：此种实现方式完全由前端控制，几乎不需要后端参与，同样支持跨域。

SSO 单点登录退出

目前我们已经完成了单点登录，在同一套认证中心的管理下，多个产品可以共享登录态。现在我们需要考虑退出了，即：在一个产品中退出了登录，怎么让其他的产品也都退出登录?

原理其实不难，可以在每一个产品在向认证中心验证 `ticket(token)` 时，其实可以顺带将自己的退出登录 `api` 发送到认证中心。

当某个产品 `c.com` 退出登录时：

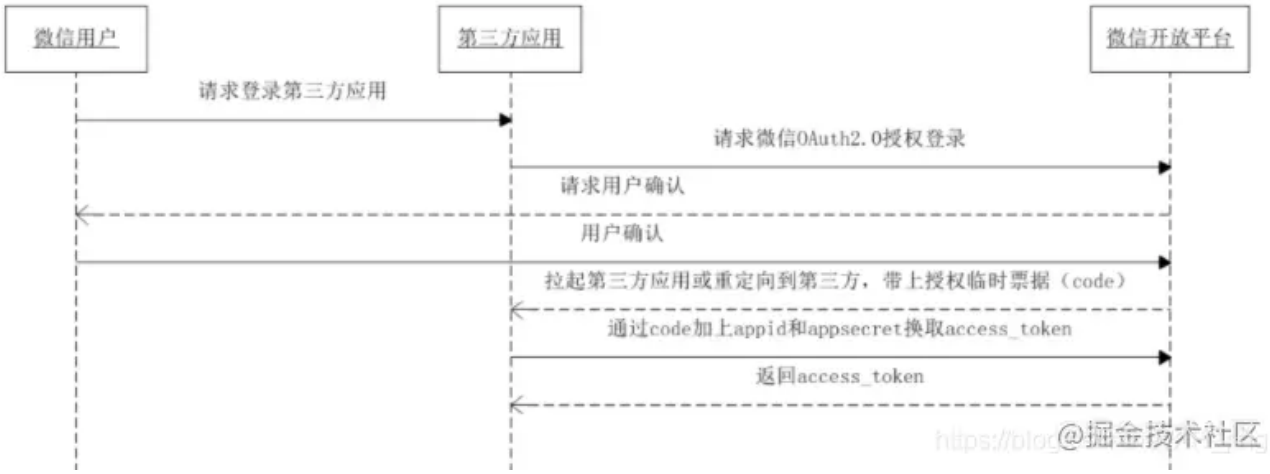
1. 清空 `c.com` 中的登录态 `Cookie`。
2. 请求认证中心 `sso.com` 中的退出 `api`。
3. 认证中心遍历下发过 `ticket(token)` 的所有产品，并调用对应的退出 `api`，完成退出。

四、OAuth 第三方登录

OAuth 第三方登录方式通常使用以下三种方式：

OAuth 机制实现流程

这里以微信开放平台的接入流程为例：



1. 首先，`a.com` 的运营者需要在微信开放平台注册账号，并向微信申请使用微信登录功能。
2. 申请成功后，得到申请的 `appid`、`appsecret`。
3. 用户在 `a.com` 上选择使用微信登录。
4. 这时会跳转微信的 OAuth 授权登录，并带上 `a.com` 的回调地址。
5. 用户输入微信账号和密码，登录成功后，需要选择具体的授权范围，如：授权用户的头像、昵称等。
6. 授权之后，微信会根据拉起 `a.com?code=123`，这时带上了一个临时票据 `code`。
7. 获取 `code` 之后，`a.com` 会拿着 `code`、`appid`、`appsecret`，向微信服务器申请 `token`，验证成功后，微信会下发一个 `token`。
8. 有了 `token` 之后，`a.com` 就可以凭借 `token` 拿到对应的微信用户头像，用户昵称等信息了。
9. `a.com` 提示用户登录成功，并将登录状态写入 `Cookie`，以作为后续访问的凭证。

其他平台的接入方式可以去对应得官方文档查看，流程基本类似。

总结

上面四种登录实现方案，基本囊括了现有的登录验证方案，原理以及实现流程基本都了解。

- `Cookie + Session` 历史悠久，适合于简单的后端架构，需开发人员自己处理好安全问题。
- `Token` 方案对后端压力小，适合大型分布式的后端架构，但已分发出去的 `token`，如果想收回权限，就不是很方便了。
- SSO 单点登录，适用于中大型企业，想要统一内部所有产品的登录方式的情况。
- OAuth 第三方登录，简单易用，对用户和开发者都友好，但第三方平台很多，需要选择合适自己的第三方登录平台。

> 参考文章： cnblogs.com/yonghengzh/... mp.weixin.qq.com/s/lcryd6TPX...