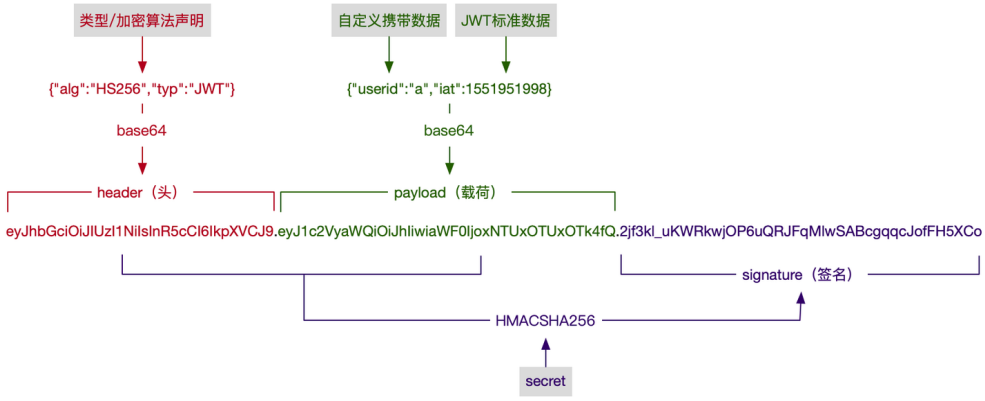


前端鉴权的兄弟们：cookie、session、token、jwt、单点登录

本文你将看到：

- 基于 HTTP 的前端鉴权背景
- cookie 为什么是最方便的存储方案，有哪些操作 cookie 的方式
- session 方案是如何实现的，存在哪些问题
- token 方案是如何实现的，如何进行编码和防篡改？jwt 是做什么的？refresh token 的实现和意义
- session 和 token 有什么异同和优缺点
- 单点登录是什么？实现思路和在浏览器下的处理

> 小广告：长期内推滴滴，直接私聊我

从状态说起

HTTP 无状态

我们知道，HTTP 是无状态的。也就是说，HTTP 请求方和响应方间无法维护状态，都是一次性的，它不知道前后的请求都发生了什么。

但有的场景下，我们需要维护状态。最典型的，一个用户登陆微博，发布、关注、评论，都应是在登录后的用户状态下的。

标记

那解决办法是什么呢？::标记::。

> 在学校或公司，入学入职那一天起，会录入你的身份、账户信息，然后给你发个卡，今后在园区内，你的门禁、打卡、消费都只需要刷这张卡。

前端存储

这就涉及到一发、一存、一带，发好办，登陆接口直接返回给前端，存储就需要前端想办法了。

> 前提是，你要把卡带在身上。

前端的存储方式有很多。

- 最矬的，挂到全局变量上，但这是个「体验卡」，一次刷新页面就没了
- 高端点的，存到 cookie、localStorage 等里，这属于「会员卡」，无论怎么刷新，只要浏览器没清掉或者过期，就一直拿着这个状态。

前端存储这里不展开了。

有地方存了，请求的时候就可以拼到参数里带给接口了。

基石：cookie

> 可是前端好麻烦啊，又要自己存，又要想办法带出去，有没有不用操心的？

有，cookie。

cookie 也是前端存储的一种，但相比于 localStorage 等其他方式，借助 HTTP 头、浏览器能力，cookie 可以做到前端无感知。

一般过程是这样的：

- 在提供标记的接口，通过 HTTP 返回头的 Set-Cookie 字段，直接「种」到浏览器上
- 浏览器发起请求时，会自动把 cookie 通过 HTTP 请求头的 Cookie 字段，带给接口

配置：Domain / Path

> 你不能拿清华的校园卡进北大。

cookie 是要限制::「空间范围」::的，通过 Domain（域）/ Path（路径）两级。

> Domain属性指定浏览器发出 HTTP 请求时，哪些域名要附带这个 Cookie。如果没有指定该属性，浏览器会默认将其设为当前 URL 的一级域名，比如 http://www.example.com 会设为 http://example.com，而且以后如果访问http://example.com的任何子域名，HTTP 请求也会带上这个 Cookie。如果服务器在Set-Cookie字段指定的域名，不属于当前域名，浏览器会拒绝这个 Cookie。
> Path属性指定浏览器发出 HTTP 请求时，哪些路径要附带这个 Cookie。只要浏览器发现，Path属性是 HTTP 请求路径的开头一部分，就会在头信息里面带上这个 Cookie。比如，PATH 属性是/，那么请求/docs路径也会包含该 Cookie。当然，前提是域名必须一致。
> —— Cookie — JavaScript 标准参考教程（alpha）

配置：Expires / Max-Age

> 你毕业了卡就不好使了。

cookie 还可以限制::「时间范围」::，通过 Expires、Max-Age 中的一种。

> Expires属性指定一个具体的到期时间，到了指定时间以后，浏览器就不再保留这个 Cookie。它的值是 UTC 格式。如果不设置该属性，或者设为null，Cookie 只在当前会话（session）有效，浏览器窗口一旦关闭，当前 Session 结束，该 Cookie 就会被删除。另外，浏览器根据本地时间，决定 Cookie 是否过期，由于本地时间是不精确的，所以没有办法保证 Cookie 一定会在服务器指定的时间过期。
> Max-Age属性指定从现在开始 Cookie 存在的秒数，比如60 * 60 * 24 * 365（即一年）。过了这个时间以后，浏览器就不再保留这个 Cookie。
> 如果同时指定了Expires和Max-Age，那么Max-Age的值将优先生效。
> 如果Set-Cookie字段没有指定Expires或Max-Age属性，那么这个 Cookie 就是 Session Cookie，即它只在本次对话存在，一旦用户关闭浏览器，浏览器就不会再保留这个 Cookie。
> —— Cookie — JavaScript 标准参考教程（alpha）

配置：Secure / HttpOnly

> 有的学校规定，不带卡套不让刷（什么奇葩学校，假设）；有的学校不让自己给卡贴贴纸。

cookie 可以限制::「使用方式」::。

> Secure属性指定浏览器只有在加密协议 HTTPS 下，才能将这个 Cookie 发送到服务器。另一方面，如果当前协议是 HTTP，浏览器会自动忽略服务器发来的Secure属性。该属性只是一个开关，不需要指定值。如果通信是 HTTPS 协议，该开关自动打开。
> HttpOnly属性指定该 Cookie 无法通过 JavaScript 脚本拿到，主要是Document.cookie属性、XMLHttpRequest对象和 Request API 都拿不到该属性。这样就防止了该 Cookie 被脚本读到，只有浏览器发出 HTTP 请求时，才会带上该 Cookie。
> —— Cookie — JavaScript 标准参考教程（alpha）

HTTP 头对 cookie 的读写

回过头来，HTTP 是如何写入和传递 cookie 及其配置的呢?

HTTP 返回的一个 Set-Cookie 头用于向浏览器写入「一条（且只能是一条）」cookie，格式为 cookie 键值 + 配置键值。例如:

```
Set-Cookie: username=jimu; domain=jimu.com; path=/blog; Expires=Wed, 21 Oct 2015 07:28
```

那我想一次多 set 几个 cookie 怎么办？多给几个 Set-Cookie 头（一次 HTTP 请求中允许重复）

```
Set-Cookie: username=jimu; domain=jimu.com
Set-Cookie: height=180; domain=me.jimu.com
Set-Cookie: weight=80; domain=me.jimu.com
```

HTTP 请求的 Cookie 头用于浏览器把符合当前「空间、时间、使用方式」配置的所有 cookie 一并发给服务端。因为由浏览器做了筛选判断，就不需要归还配置内容了，只要发送键值就可以。

```
Cookie: username=jimu; height=180; weight=80
```

前端对 cookie 的读写

前端可以自己创建 cookie，如果服务端创建的 cookie 没加 `HttpOnly`，那恭喜你也可以修改他给的 cookie。

调用 `document.cookie` 可以创建、修改 cookie，和 HTTP 一样，一次 `document.cookie` 能且只能操作一个 cookie。

```
document.cookie = 'username=jimu; domain=jimu.com; path=/blog; Expires=Wed, 21 Oct 201
```

调用 `document.cookie` 也可以读到 cookie，也和 HTTP 一样，能读到所有的非 `HttpOnly` cookie。

```
console.log(document.cookie);
// username=jimu; height=180; weight=80
```

（就一个 cookie 属性，为什么读写行为不一样？get / set 了解下）

cookie 是维持 HTTP 请求状态的基石

了解了 cookie 后，我们知道 cookie 是最便捷的维持 HTTP 请求状态的方式，大多数前端鉴权问题都是靠 cookie 解决的。当然也可以选用别的存储方式（后面也会多多少少提到）。

那有了存储工具，接下来怎么做呢？

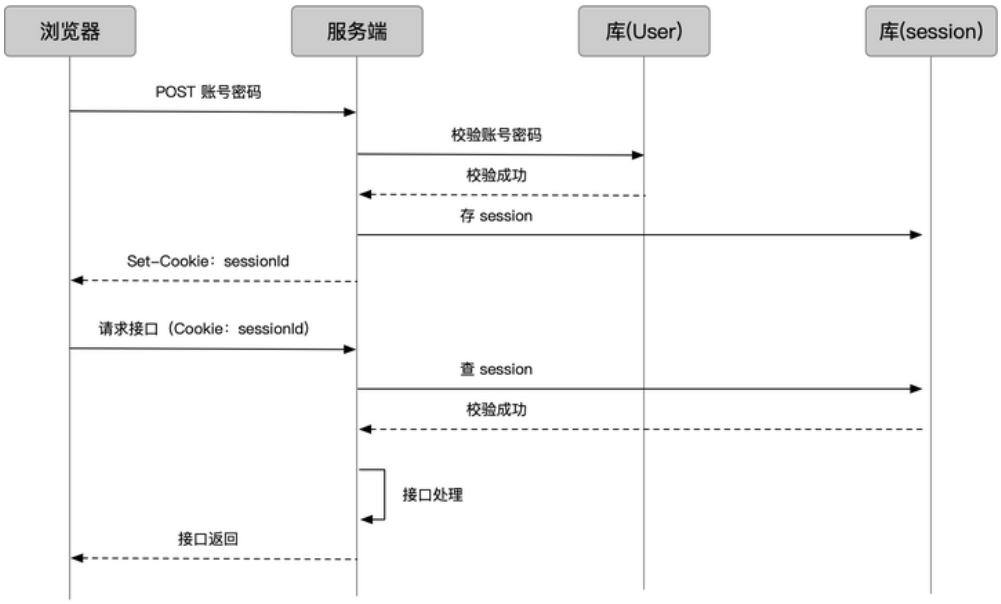
应用方案：服务端 session

现在回想下，你刷卡的时候发生了什么？

> 其实你的卡上只存了一个 id（可能是你的学号），刷的时候物业系统去查你的信息、账户，再决定「这个门你能不能进」「这个鸡腿去哪个账户扣钱」。

这种操作，在前后端鉴权系统中，叫 session。

典型的 session 登陆/验证流程：



- 浏览器登录发送账号密码，服务端查用户库，校验用户
- 服务端把用户登录状态存为 Session，生成一个 sessionId
- 通过登录接口返回，把 sessionId set 到 cookie 上
- 此后浏览器再请求业务接口，sessionId 随 cookie 带上
- 服务端查 sessionId 校验 session
- 成功后正常做业务处理，返回结果

Session 的存储方式

显然，服务端只是给 cookie 一个 sessionId，而 session 的具体内容（可能包含用户信息、session 状态等），要自己存一下。存储的方式有几种：

- Redis（推荐）：内存型数据库，redis中文官方网站。以 key-value 的形式存，正合 sessionId-sessionData 的场景；且访问快。
- 内存：直接放到变量里。一旦服务重启就没了
- 数据库：普通数据库。性能不高。

Session 的过期和销毁

很简单，只要把存储的 session 数据销毁就可以。

Session 的分布式问题

通常服务端是集群，而用户请求过来会走一次负载均衡，不一定打到哪台机器上。那一旦用户后续接口请求到的机器和他登录请求的机器不一致，或者登录请求的机器宕机了，session 不就失效了吗？

这个问题现在有几种解决方式。

- 一是从「存储」角度，把 session 集中存储。如果我们用独立的 Redis 或普通数据库，就可以把 session 都存到一个库里。
- 二是从「分布」角度，让相同 IP 的请求在负载均衡时都打到同一台机器上。以 nginx 为例，可以配置 ip_hash 来实现。

但通常还是采用第一种方式，因为第二种相当于阉割了负载均衡，且仍没有解决「用户请求的机器宕机」的问题。

node.js 下的 session 处理

前面的图很清楚了，服务端要实现对 cookie 和 session 的存取，实现起来要做的事还是很多的。在 `npm` 中，已经有封装好的中间件，比如 express-session - npm，用法就不贴了。

这是它种的 cookie：

Name	Value	Domain	Path	Expires	HTTP	Secure
connect.sid	s%3AuAjb_Sb0wuh5Nj31DnAMFAGPG-Up_HDZ.Xpt7h25YMRaYqTB%2FWrgk7R6yg%2FDozEg57UoT3ncNNTk	localhost	/	Never	true	false

express-session - npm 主要实现了：

- 封装了对cookie的读写操作，并提供配置项配置字段、加密方式、过期时间等。
- 封装了对session的存取操作，并提供配置项配置session存储方式（内存/redis）、存储规则等。
- 给req提供了session属性，控制属性的set/get并响应到cookie和session存取上，并给req.session提供了一些方法。

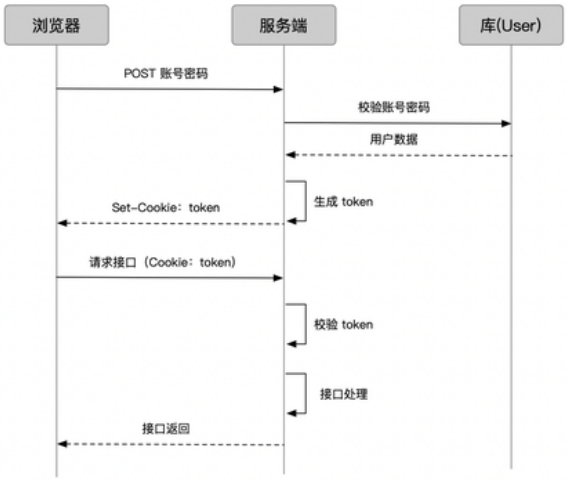
应用方案：token

session 的维护给服务端造成很大困扰，我们必须找地方存放它，又要考虑分布式的问题，甚至要单独为了它启用一套 Redis 集群。有没有更好的办法？

> 我又想到学校，在没有校园卡技术以前，我们都靠「学生证」。
> 门卫小哥直接对照我和学生证上的脸，确认学生证有效期、年级等信息，就可以放行了。

回过头来想想，一个登录场景，也不必往 session 存太多东西，那为什么不直接打包到 cookie 中呢？这样服务端不用存了，每次只要核验 cookie 带的「证件」有效性就可以了，也可以携带一些轻量的信息。

这种方式通常被叫做 token。



token 的流程是这样的：

- 用户登录，服务端校验账号密码，获得用户信息
- 把用户信息、token 配置编码成 token，通过 cookie set 到浏览器
- 此后用户请求业务接口，通过 cookie 携带 token
- 接口校验 token 有效性，进行正常业务接口处理

客户端 token 的存储方式

在前面 cookie 说过，cookie 并不是客户端存储凭证的唯一方式。token 因为它的「无状态性」，有效期、使用限制都包在 token 内容里，对 cookie 的管理能力依赖较小，客户端存起来就显得更自由。但 web 应用的主流方式仍是放在 cookie 里，毕竟少操心。

token 的过期

那我们如何控制 token 的有效期呢？很简单，把「过期时间」和数据一起塞进去，验证时判断就好。

token 的编码

编码的方式丰俭由人。

base64

比如 node 端的 cookie-session - npm 库

> 不要纠结名字，其实是个 token 库，但保持了和 express-session - npm 高度一致的用法，把要存的数据挂在 session 上

默认配置下，当我给他一个 userid，他会存成这样：

Name	Value	Domain	Path	Expires	HTTP	Secure
token	eyJ1c2VyaWQiOiJhIn0=	localhost	/	Never	true	false

这里的 `eyJ1c2VyaWQiOiJhIn0=`，就是 `{"userid":"abb”}` 的 base64 而已。

防篡改

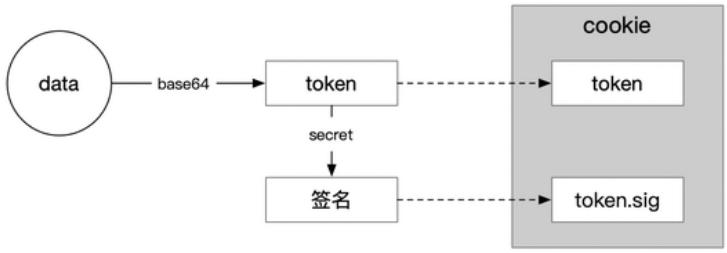
> 那问题来了，如果用户 cdd 拿 `{"userid":"abb”}` 转了个 base64，再手动修改了自己的 token 为 `eyJ1c2VyaWQiOiJhIn0=`，是不是就能直接访问到 abb 的数据了？

是的。所以看情况，如果 token 涉及到敏感权限，就要想办法避免 token 被篡改。

解决方案就是给 token 加签名，来识别 token 是否被篡改过。例如在 cookie-session - npm 库中，增加两项配置：

```
secret: 'iAmSecret',
signed: true,
```

这样会多种一个 .sig cookie，里面的值就是 `{"userid":"abb"}` 和 `iAmSecret` 通过加密算法计算出来的，常见的比如HMACSHA256 类 (System.Security.Cryptography) | Microsoft Docs。



好了，现在 cdd 虽然能伪造出 `eyJ1c2VyaWQiOiJhIn0=`，但伪造不出 sig 的内容，因为他不知道 secret。

JWT

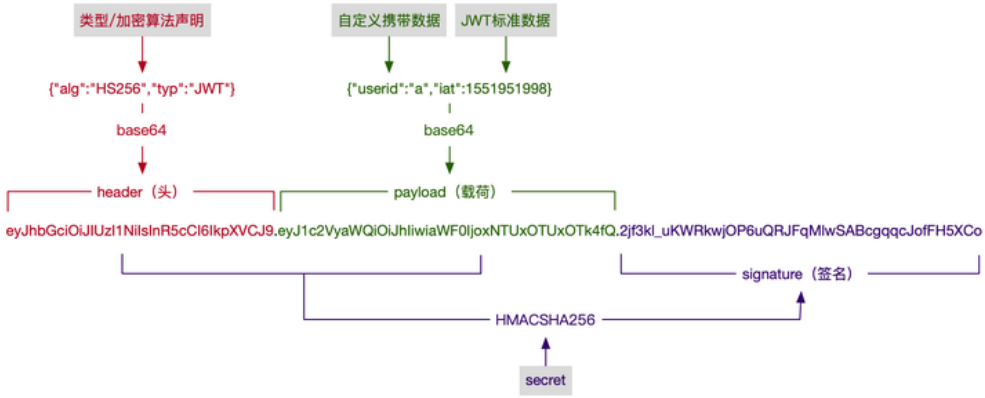
但上面的做法额外增加了 cookie 数量，数据本身也没有规范的格式，所以 JSON Web Token Introduction - jwt.io 横空出世了。

> JSON Web Token (JWT) 是一个开放标准，定义了一种传递 JSON 信息的方式。这些信息通过数字签名确保可信。

它是一种成熟的 token 字符串生成方案，包含了我们前面提到的数据、签名。不如直接看一下一个 JWT token 长什么样：

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyaWQiOiJhIiwiaWF0IjoxNTUxOTUxOTk4fQ.2jf3kl
```

这串东西是怎么生成的呢？看图：



类型、加密算法的选项，以及 JWT 标准数据字段，可以参考 RFC 7519 - JSON Web Token (JWT)

node 上同样有相关的库实现：express-jwt - npm koa-jwt - npm

refresh token

token，作为权限守护者，最重要的就是「安全」。

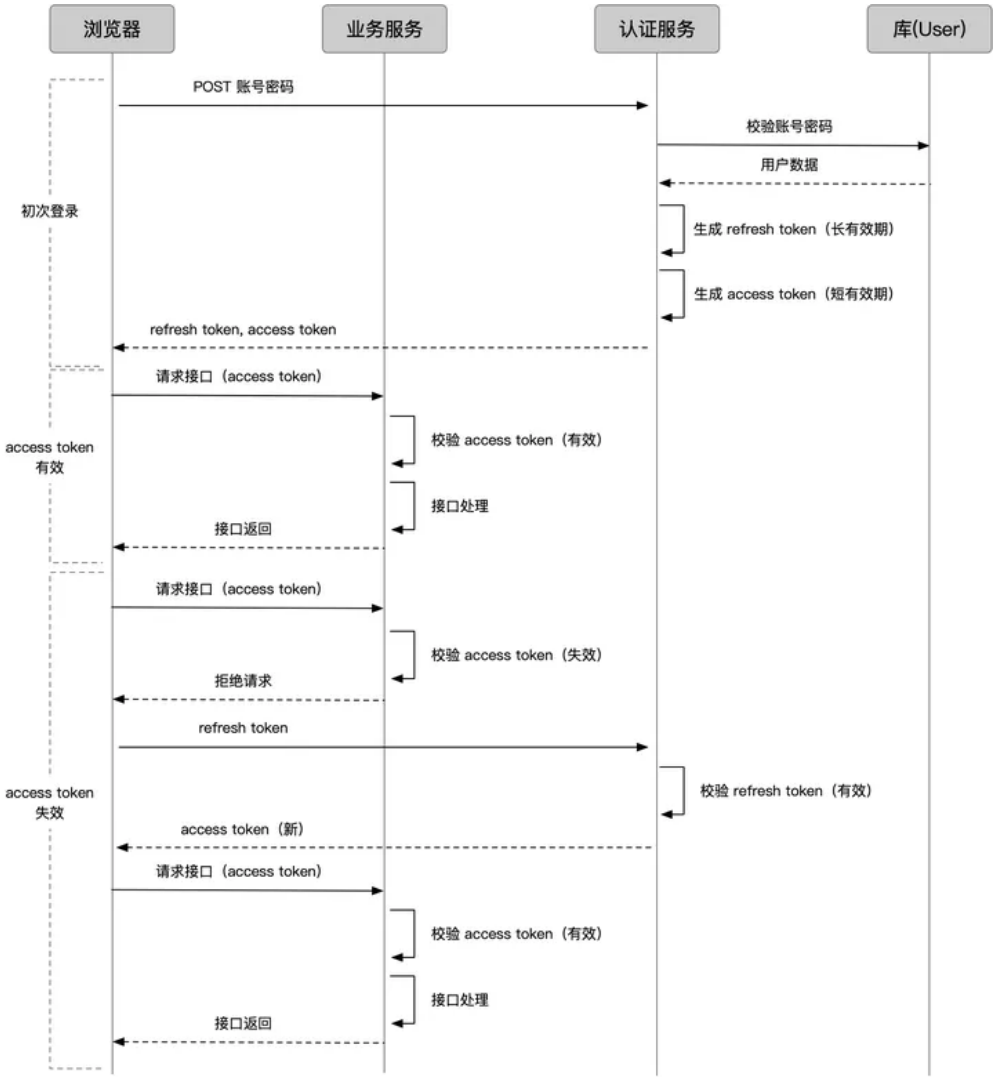
业务接口用来鉴权的 token，我们称之为 access token。越是权限敏感的业务，我们越希望 access token 有效期足够短，以避免被盗用。但过短的有效期会造成 access token 经常过期，过期后怎么办呢?

一种办法是，让用户重新登录获取新 token，显然不够友好，要知道有的 access token 过期时间可能只有几分钟。

另外一种办法是，再来一个 token，一个专门生成 access token 的 token，我们称为 refresh token。

- access token 用来访问业务接口，由于有效期足够短，盗用风险小，也可以使请求方式更宽松灵活
- refresh token 用来获取 access token，有效期可以长一些，通过独立服务和严格的请求方式增加安全性；由于不常验证，也可以如前面的 session 一样处理

有了 refresh token 后，几种情况的请求流程变成这样：



如果 refresh token 也过期了，就只能重新登录了。

session 和 token

session 和 token 都是边界很模糊的概念，就像前面说的，refresh token 也可能以 session 的形式组织维护。

狭义上，我们通常认为 session 是「种在 cookie 上、数据存在服务端」的认证方案，token 是「客户端存哪都行、数据存在 token 里」的认证方案。对 session 和 token 的对比本质上是「客户端存 cookie / 存别地儿」、「服务端存数据 / 不存数据」的对比。

客户端存 cookie / 存别地儿

存 cookie 固然方便不操心，但问题也很明显：

- 在浏览器端，可以用 cookie（实际上 token 就常用 cookie），但出了浏览器端，没有 cookie 怎么办？
- cookie 是浏览器在域下自动携带的，这就容易引发 CSRF 攻击（前端安全系列（二）：如何防止CSRF攻击？ - 美团技术团队）

存别的地方，可以解决没有 cookie 的场景；通过参数等方式手动带，可以避免 CSRF 攻击。

服务端存数据 / 不存数据

- 存数据：请求只需携带 id，可以大幅缩短认证字符串长度，减小请求体积
- 不存数据：不需要服务端整套的解决方案和分布式处理，降低硬件成本；避免查库带来的验证延迟

单点登录

前面我们已经知道了，在同域下的客户端/服务端认证系统中，通过客户端携带凭证，维持一段时间内的登录状态。

但当我们业务线越来越多，就会有更多业务系统分散到不同域名下，就需要「一次登录，全线通用」的能力，叫做「单点登录」。

“虚假”的单点登录（一级域名相同）

简单的，如果业务系统都在同一一级域名下，比如 `wenku.baidu.com` `tieba.baidu.com`，就好办了。可以直接把 cookie domain 设置为一级域名 `baidu.com`，百度也就是这么干的。

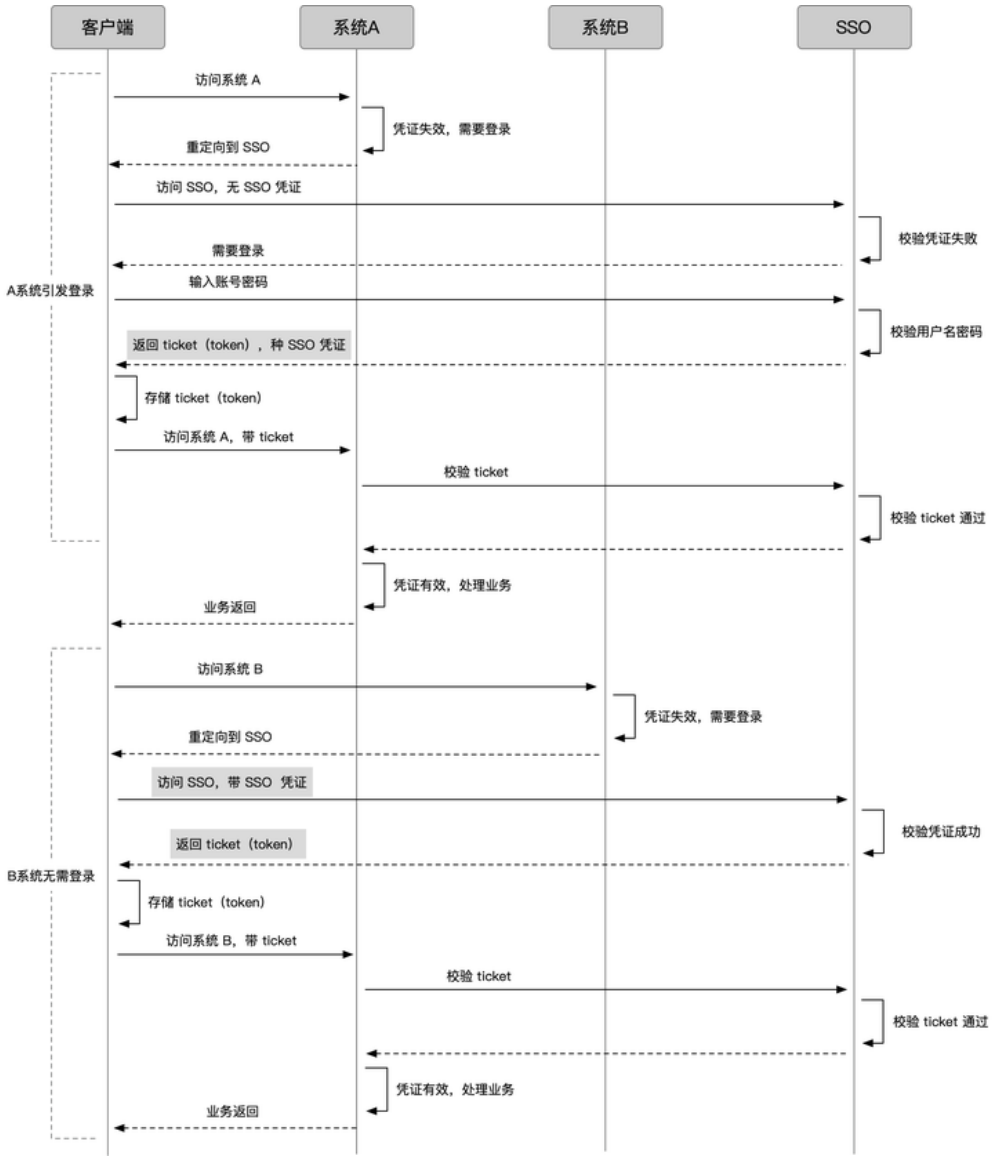
“真实”的单点登录（一级域名不同）

比如滴滴这么潮的公司，同时拥有 `didichuxing.com` `xiaojukeji.com` `didiglobal.com` 等域名，种 cookie 是完全绕不开的。

这要能实现「一次登录，全线通用」，才是真正的单点登录。

这种场景下，我们需要独立的认证服务，通常被称为 SSO。

一次「从 A 系统引发登录，到 B 系统不用登录」的完整流程

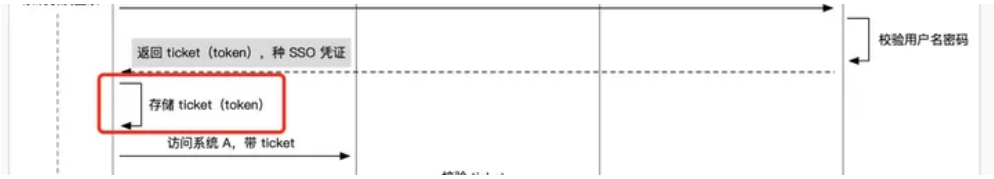


- 用户进入 A 系统，没有登录凭证（ticket），A 系统给他跳到 SSO
- SSO 没登录过，也就没有 sso 系统下没有凭证（注意这个和前面 A ticket 是两回事），输入账号密码登录
- SSO 账号密码验证成功，通过接口返回做两件事：一是种下 sso 系统下凭证（记录用户在 SSO 登录状态）；二是下发一个 ticket
- 客户端拿到 ticket，保存起来，带着请求系统 A 接口
- 系统 A 校验 ticket，成功后正常处理业务请求
- 此时用户第一次进入系统 B，没有登录凭证（ticket），B 系统给他跳到 SSO
- SSO 登录过，系统下有凭证，不用再次登录，只需要下发 ticket
- 客户端拿到 ticket，保存起来，带着请求系统 B 接口

完整版本：考虑浏览器的场景

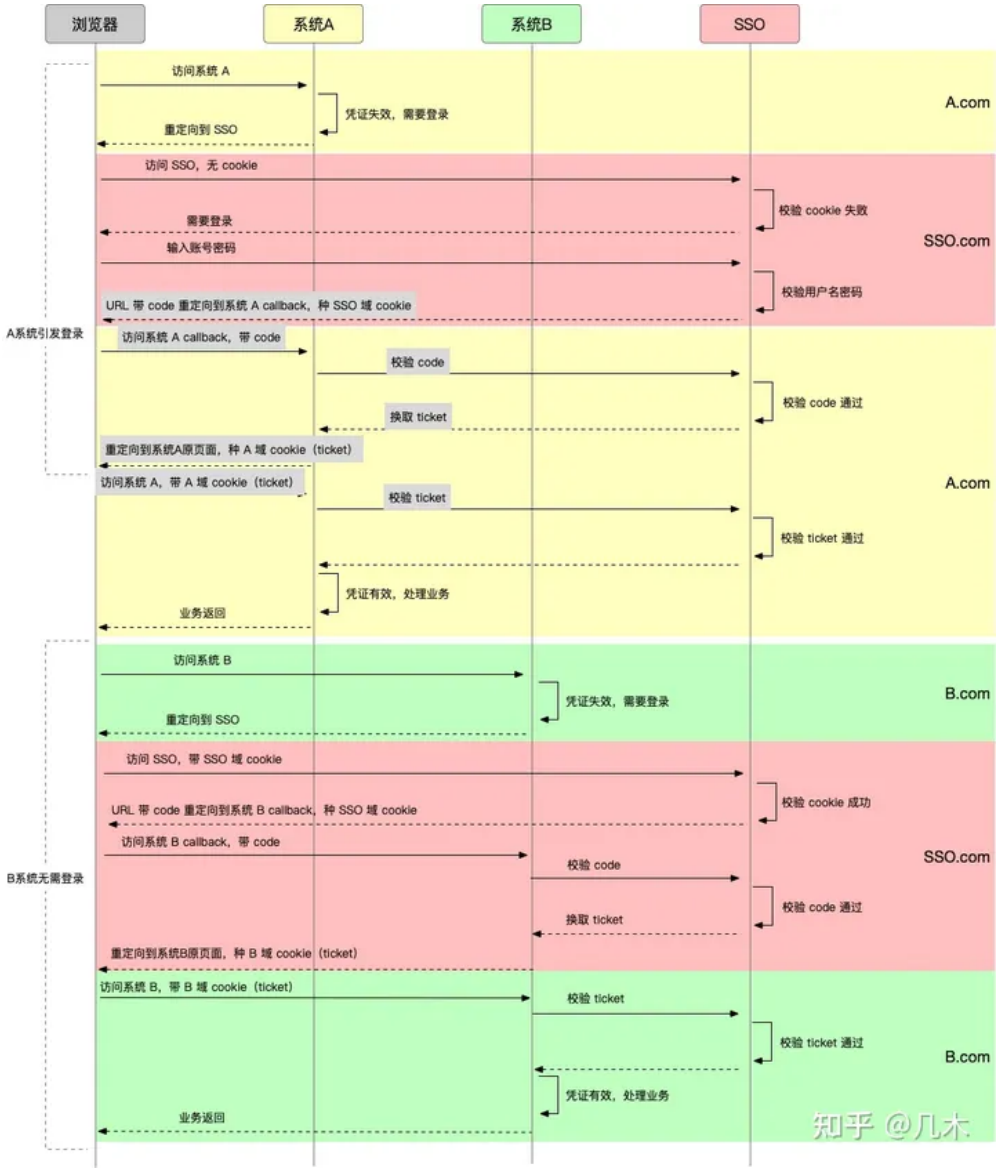
上面的过程看起来没问题，实际上很多 APP 等端上这样就够了。但在浏览器下不见得好用。

看这里：



对浏览器来说，SSO 域下返回的数据要怎么存，才能在访问 A 的时候带上？浏览器对跨域有严格限制，cookie、localStorage 等方式都是有域限制的。

这就需要也只能由 A 提供 A 域下存储凭证的能力。一般我们是这么做的：



图中我们通过颜色把浏览器当前所处的域名标记出来。注意图中灰底文字说明部分的变化。

- 在 SSO 域下，SSO 不是通过接口把 ticket 直接返回，而是通过一个带 code 的 URL 重定向到系统 A 的接口上，这个接口通常在 A 向 SSO 注册时约定
- 浏览器被重定向到 A 域下，带着 code 访问了 A 的 callback 接口，callback 接口通过 code 换取 ticket
- 这个 code 不同于 ticket，code 是一次性的，暴露在 URL 中，只为了传一下换 ticket，换完就失效
- callback 接口拿到 ticket 后，在自己的域下 set cookie 成功
- 在后续请求中，只需要把 cookie 中的 ticket 解析出来，去 SSO 验证就好
- 访问 B 系统也是一样

总结

- HTTP 是无状态的，为了维持前后请求，需要前端存储标记
- cookie 是一种完善的标记方式，通过 HTTP 头或 js 操作，有对应的安全策略，是大多数状态管理方案的基石

- session 是一种状态管理方案，前端通过 cookie 存储 id，后端存储数据，但后端要处理分布式问题
- token 是另一种状态管理方案，相比于 session 不需要后端存储，数据全部存在前端，解放后端，释放灵活性
- token 的编码技术，通常基于 base64，或增加加密算法防篡改，jwt 是一种成熟的编码方案
- 在复杂系统中，token 可通过 service token、refresh token 的分权，同时满足安全性和用户体验
- session 和 token 的对比就是「用不用cookie」和「后端存不存」的对比
- 单点登录要求不同域下的系统「一次登录，全线通用」，通常由独立的 SSO 系统记录登录状态、下发 ticket，各业务系统配合存储和认证 ticket