链滴

Cookie、Session、Token、JWT 都是什么？今天我们彻底了解一下！

作者：ieras
原文链接：https://ld246.com/article/1597404908402
来源网站：链滴

1、什么是认证（Authentication）

- 通俗地讲就是 **验证当前用户的身份**，证明“你是你自己”（比如：你每天上下班打卡，都需要通指纹打卡，当你的指纹和系统里录入的指纹相匹配时，就打卡成功）
- 互联网中的认证：
  - 用户名密码登录
  - 邮箱发送登录链接
  - 手机号接收验证码
  - 只要你能收到邮箱/验证码，就默认你是账号的主人

2、什么是授权（Authorization）

- **用户授予第三方应用访问该用户某些资源的权限**
  - 你在安装手机应用的时候，APP 会询问是否允许授予权限（访问相册、地理位置等权限）
  - 你在访问微信小程序时，当登录时，小程序会询问是否允许授予权限（获取昵称、头像、地区性别等个人信息）
- 实现授权的方式有：cookie、session、token、OAuth

3、什么是凭证（Credentials）

- **实现认证和授权的前提**是需要一种**媒介（证书）** 来标记访问者的身份
  - 在战国时期，商鞅变法，发明了照身帖。照身帖由官府发放，是一块打磨光滑细密的竹板，上刻有持有人的头像和籍贯信息。国人必须持有，如若没有就被认为是黑户，或者间谍之类的。

● 在现实生活中，每个人都会有一张专属的居民身份证，是用于证明持有人身份的一种法定证件。通身份证，我们可以办理手机卡/银行卡/个人贷款/交通出行等等，这就是**认证的凭证。**

● 在互联网应用中，一般网站（如掘金）会有两种模式，游客模式和登录模式。游客模式下，可正常浏览网站上面的文章，一旦想要点赞/收藏/分享文章，就需要登录或者注册账号。当用户登录成后，服务器会给该用户使用的浏览器颁发一个令牌（token），这个令牌用来表明你的身份，每次浏器发送请求时会带上这个令牌，就可以使用游客模式下无法使用的功能。

4、什么是 Cookie

● **HTTP 是无状态的协议（对于事务处理没有记忆能力，每次客户端和服务端会话完成时，服务端不保存任何会话信息）**：每个请求都是完全独立的，服务端无法确认当前访问者的身份信息，无法分辨一次的请求发送者和这一次的发送者是不是同一个人。所以服务器与浏览器为了进行会话跟踪（知道谁在访问我），就必须主动的去维护一个状态，这个状态用于告知服务端前后两个请求是否来自同一览器。而这个状态需要通过 cookie 或者 session 去实现。

● **cookie 存储在客户端：** cookie 是服务器发送到用户浏览器并保存在本地的一小块数据，它会在浏器下次向同一服务器再发起请求时被携带并发送到服务器上。

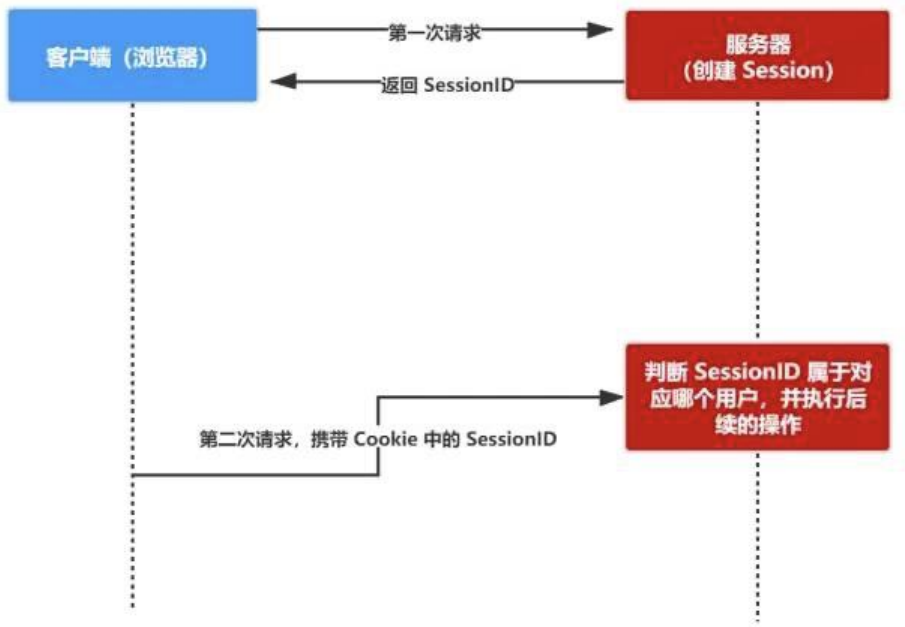
● **cookie 是不可跨域的：** 每个 cookie 都会绑定单一的域名，无法在别的域名下获取使用，**一级域和二级域名之间是允许共享使用的（靠的是 domain）**。

cookie 重要的属性

属性	说明
name=value	键值对，设置 Cookie 的名称及相对应的值，都必须是**字符串类型** - 如果值为 Unicode 字符，需要为字符编码。 - 如果值为二进制数据，则需要使用 BASE64 编码。
domain	指定 cookie 所属域名，默认是当前域名
path	**指定 cookie 在哪个路径（路由）下生效，默认是 '/'。** 如果设置为 `/abc` ，则只有 `/abc` 下的路由可以访问到该 cookie，如： `/abc/read` 。
maxAge	cookie 失效的时间，单位秒。如果为整数，则该 cookie 在 maxAge 秒后失效。如果为负数，该 cookie 为临时 cookie ，关闭浏览器即失效，浏览器也不会以任何形式保存该 cookie 。如果为 0，表示删除该 cookie 。默认为 -1。 - **比 expires 好用。**
expires	过期时间，在设置的某个时间点后该 cookie 就会失效。 一般浏览器的 cookie 都是默认储存的，当关闭浏览器结束这个会话的时候，这个 cookie 也就会被删除
secure	该 cookie 是否仅被使用安全协议传输。安全协议有 HTTPS，SSL等，在网络上传输数据之前先将数据加密。默认为false。 当 secure 值为 true 时，cookie 在 HTTP 中是无效，在 HTTPS 中才有效。
httpOnly	**如果给某个 cookie 设置了 httpOnly 属性，则无法通过 JS 脚本 读取到该 cookie 的信息，但还是能通过 Application 中手动修改 cookie，所以只是在一定程度上可以防止 XSS 攻击，不是绝对的安全**

5、什么是 Session

- **session 是另一种记录服务器和客户端会话状态的机制**
- **session 是基于 cookie 实现的，session 存储在服务器端，sessionId 会被存储到客户端的cookie 中**



- **session 认证流程：**
  - 用户第一次请求服务器的时候，服务器根据用户提交的相关信息，创建对应的 Session
  - 请求返回时将此 Session 的唯一标识信息 SessionID 返回给浏览器
  - 浏览器接收到服务器返回的 SessionID 信息后，会将此信息存入到 Cookie 中，同时 Cookie 录此 SessionID 属于哪个域名
  - 当用户第二次访问服务器的时候，请求会自动判断此域名下是否存在 Cookie 信息，如果存在自将 Cookie 信息也发送给服务端，服务端会从 Cookie 中获取 SessionID，再根据 SessionID 查找对的 Session 信息，如果没有找到说明用户没有登录或者登录失效，如果找到 Session 证明用户已经登可执行后面操作。

根据以上流程可知，**SessionID 是连接 Cookie 和 Session 的一道桥梁**，大部分系统也是根据此原来验证用户登录状态。

Cookie 和 Session 的区别

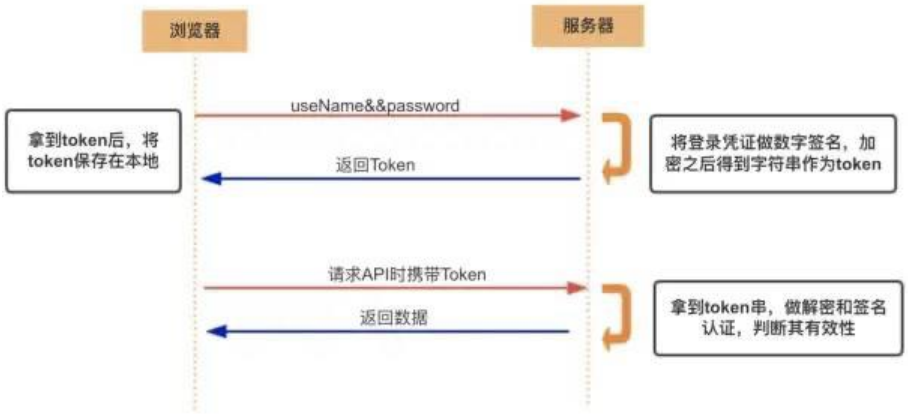
- **安全性：** Session 比 Cookie 安全，Session 是存储在服务器端的，Cookie 是存储在客户端的。
- **存取值的类型不同**：Cookie 只支持存字符串数据，想要设置其他类型的数据，需要将其转换成字串，Session 可以存任意数据类型。
- **有效期不同：** Cookie 可设置为长时间保持，比如我们经常使用的默认登录功能，Session 一般失时间较短，客户端关闭（默认情况下）或者 Session 超时都会失效。
- **存储大小不同：** 单个 Cookie 保存的数据不能超过 4K，Session 可存储数据远高于 Cookie，但是访问量过多，会占用过多的服务器资源。

6、什么是 Token（令牌）

Acesss Token

- **访问资源接口（API）时所需要的资源凭证**

- **简单 token 的组成：** uid(用户唯一的身份标识)、time(当前时间的时间戳)、sign（签名，token 前几位以哈希算法压缩成的一定长度的十六进制字符串）
- **特点：**
  - **服务端无状态化、可扩展性好**
  - **支持移动端设备**
  - 安全
  - 支持跨程序调用
- **token 的身份验证流程：**

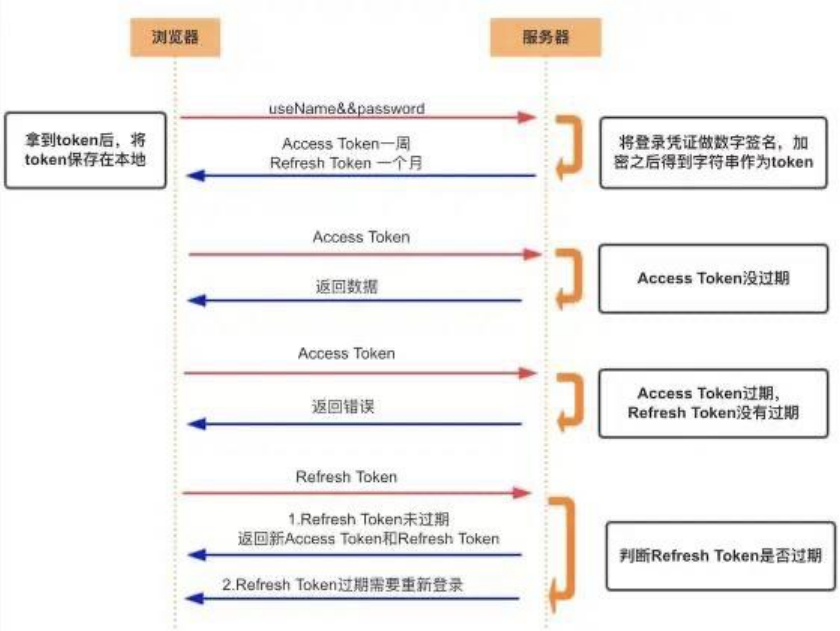


1. 客户端使用用户名跟密码请求登录
2. 服务端收到请求，去验证用户名与密码
3. 验证成功后，服务端会签发一个 token 并把这个 token 发送给客户端
4. 客户端收到 token 以后，会把它存储起来，比如放在 cookie 里或者 localStorage 里
5. 客户端每次向服务端请求资源的时候需要带着服务端签发的 token
6. 服务端收到请求，然后去验证客户端请求里面带着的 token ，如果验证成功，就向客户端返回请的数据

- **每一次请求都需要携带 token，需要把 token 放到 HTTP 的 Header 里**
- **基于 token 的用户认证是一种服务端无状态的认证方式，服务端不用存放 token 数据。用解析 to en 的计算时间换取 session 的存储空间，从而减轻服务器的压力，减少频繁的查询数据库**
- **token 完全由应用管理，所以它可以避开同源策略**

Refresh Token

- 另外一种 token——refresh token
- refresh token 是专用于刷新 access token 的 token。如果没有 refresh token，也可以刷新 acces token，但每次刷新都要用户输入登录用户名与密码，会很麻烦。有了 refresh token，可以减少这麻烦，客户端直接用 refresh token 去更新 access token，无需用户进行额外的操作。



● Access Token 的有效期比较短，当 Acesss Token 由于过期而失效时，使用 Refresh Token 就可获取到新的 Token，如果 Refresh Token 也失效了，用户就只能重新登录了。

● Refresh Token 及过期时间是存储在服务器的数据库中，只有在申请新的 Acesss Token 时才会验，不会对业务接口响应时间造成影响，也不需要向 Session 一样一直保持在内存中以应对大量的请求。

Token 和 Session 的区别

● Session 是一种 **记录服务器和客户端会话状态的机制，使服务端有状态化，可以记录会话信息。**而 Token 是**令牌，访问资源接口（API）时所需要的资源凭证。**Token **使服务端无状态化，不会存储会信息。**

● Session 和 Token 并不矛盾，作为身份认证 Token 安全性比 Session 好，因为每一个请求都有签还能防止监听以及重放攻击，而 Session 就必须依赖链路层来保障通讯安全了。**如果你需要实现有状的会话，仍然可以增加 Session 来在服务器端保存一些状态。**

● 所谓 Session 认证只是简单的把 User 信息存储到 Session 里，因为 SessionID 的不可预测性，暂认为是安全的。而 Token ，如果指的是 OAuth Token 或类似的机制的话，提供的是 认证 和 授权 认证是针对用户，授权是针对 App 。其目的是让某 App 有权利访问某用户的信息。这里的 Token 唯一的。不可以转移到其它 App上，也不可以转到其它用户上。Session 只提供一种简单的认证，即要有此 SessionID ，即认为有此 User 的全部权利。是需要严格保密的，这个数据应该只保存在站方不应该共享给其它网站或者第三方 App。所以简单来说：**如果你的用户数据可能需要和第三方共享，者允许第三方调用 API 接口，用 Token 。如果永远只是自己的网站，自己的 App，用什么就无所了。**

7、什么是 JWT

● JSON Web Token（简称 JWT）是目前最流行的 **跨域认证**解决方案。

● 是一种 **认证授权机制。**

● JWT 是为了在网络应用环境间 **传递声明**而执行的一种基于 JSON 的开放标准（RFC 7519）。JWT 的声明一般被用来在身份提供者和服务提供者间传递被认证的用户身份信息，以便于从资源服务器获资源。比如用在用户登录上。

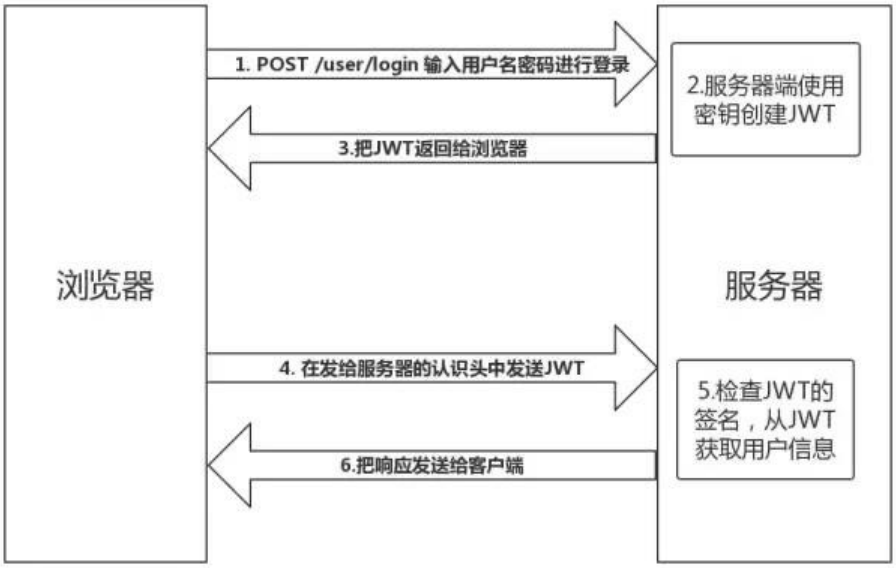
● 可以使用 HMAC 算法或者是 RSA 的公/私秘钥对 JWT 进行签名。因为数字签名的存在，这些传递信息是可信的。

- **阮一峰老师的 JSON Web Token 入门教程 讲的非常通俗易懂，这里就不再班门弄斧了**

生成 JWT

- https://jwt.io/
- https://www.jsonwebtoken.io/

JWT 的原理



- **JWT 认证流程：**
  - 用户输入用户名/密码登录，服务端认证成功后，会返回给客户端一个 JWT
  - 客户端将 token 保存到本地（通常使用 localstorage，也可以使用 cookie）
  - 当用户希望访问一个受保护的路由或者资源的时候，需要请求头的 Authorization 字段中使用Bearer 模式添加 JWT，其内容看起来是下面这样

Authorization: Bearer

- 服务端的保护路由将会检查请求头 Authorization 中的 JWT 信息，如果合法，则允许用户的行为
- 因为 JWT 是自包含的（内部包含了一些会话信息），因此减少了需要查询数据库的需要
- 因为 JWT 并不使用 Cookie 的，所以你可以使用任何域名提供你的 API 服务而不需要担心跨域资共享问题（CORS）
- 因为用户的状态不再存储在服务端的内存中，所以这是一种无状态的认证机制

JWT 的使用方式

- 客户端收到服务器返回的 JWT，可以储存在 Cookie 里面，也可以储存在 localStorage。

方式一

● 当用户希望访问一个受保护的路由或者资源的时候，可以把它放在 Cookie 里面自动发送，但是这不能跨域，所以更好的做法是放在 HTTP 请求头信息的 Authorization 字段里，使用 Bearer 模式添加 JWT。

GET /calendar/v1/eventsHost: api.example.comAuthorization: Bearer

● 用户的状态不会存储在服务端的内存中，这是一种 **无状态的认证机制**

● 服务端的保护路由将会检查请求头 Authorization 中的 JWT 信息，如果合法，则允许用户的行为。

● 由于 JWT 是自包含的，因此减少了需要查询数据库的需要

● JWT 的这些特性使得我们可以完全依赖其无状态的特性提供数据 API 服务，甚至是创建一个下载服务。

● 因为 JWT 并不使用 Cookie ，所以你可以使用任何域名提供你的 API 服务而 **不需要担心跨域资共享问题**（CORS）

方式二

● 跨域的时候，可以把 JWT 放在 POST 请求的数据体里。

方式三

● 通过 URL 传输

http://www.example.com/user?token=xxx

项目中使用 JWT

项目地址：https://github.com/yjdjiayou/jwt-demo

Token 和 JWT 的区别

相同：

● 都是访问资源的令牌

● 都可以记录用户的信息

● 都是使服务端无状态化

● 都是只有验证成功后，客户端才能访问服务端上受保护的资源

区别：

● Token：服务端验证客户端发送过来的 Token 时，还需要查询数据库获取用户信息，然后验证 Tok n 是否有效。

● JWT：将 Token 和 Payload 加密后存储于客户端，服务端只需要使用密钥解密进行校验（校验也是 JWT 自己实现的）即可，不需要查询或者减少查询数据库，因为 JWT 自包含了用户信息和加密的数 。

8、常见的前后端鉴权方式

1. Session-Cookie

2. Token 验证（包括 JWT，SSO）

3. OAuth2.0（开放授权）

9、常见的加密算法

SHA 家族函数的比较

算法及其变体		输出长度 (位)	内部状态大小 (位)	块大小 (位)	最大消息长度 (位)	循环	操作	安全性 (位)	示例的性能[14] (MiB/s)
MD5 (作为参考)		128	128 (4 × 32)	512	$2^{64}-1$	64	按位与, 按位异或, 循环移位, 填充(求模 2^{32}), 按位或	<18 (已发现碰撞)	335
SHA-0		160	160 (5 × 32)	512	$2^{64}-1$	80	按位与, 按位异或, 循环移位, 填充(求模 2^{32}),按位或	<34 (已发现碰撞)	-
SHA-1		160	160 (5 × 32)	512	$2^{64}-1$	80		<63 (已发现碰撞[15])	192
SHA-2	SHA-224	224	256 (8 × 32)	512	$2^{64}-1$	64	按位与, 按位异或, 循环移位, 填充(求模 2^{32}), 按位或, 移位	是 112/128	139
	SHA-256	256							
	SHA-384	384	512 (8 × 64)	1024	$2^{128}-1$	80	按位与, 按位异或, 循环移位, 填充(求模 2^{64}), 按位或, 移位	是 192/256/112/128	154
	SHA-512	512							
	SHA-512/224	224							
	SHA-512/256	256							
SHA-3	SHA3-224	224	1600 (5 × 5 × 64)	1152	无限制	24	按位与, 按位异或, 循环移位, 取反	是 112/128/192/256	-
	SHA3-256	256		1088					
	SHA3-384	384		832					
	SHA3-512	512		576					
	SHAKE128	d (可变长)		1344				是 min (d/2, 128)	-
	SHAKE256	d (可变长)		1088				min (d/2, 256)	

● 哈希算法(Hash Algorithm)又称散列算法、散列函数、哈希函数，是一种从任何一种数据中创建小数字“指纹”的方法。哈希算法将数据重新打乱混合，重新创建一个哈希值。

● 哈希算法主要用来保障数据真实性(即完整性)，即发信人将原始消息和哈希值一起发送，收信人通相同的哈希函数来校验原始数据是否真实。

● 哈希算法通常有以下几个特点：

- 2 的 128 次方为 340282366920938463463374607431768211456，也就是 10 的 39 次方级别
- 2 的 160 次方为 1.46150163733090291820368483271 63e+48，也就是 10 的 48 次方级别
- 2 的 256 次方为 1.15792089237316195423570985008 69 × 10 的 77 次方，也就是 10 的 77 次方
- 正像快速：原始数据可以快速计算出哈希值
- 逆向困难：通过哈希值基本不可能推导出原始数据
- 输入敏感：原始数据只要有一点变动，得到的哈希值差别很大
- 冲突避免：很难找到不同的原始数据得到相同的哈希值，宇宙中原子数大约在 10 的 60 次方到 0 次方之间，所以 2 的 256 次方有足够的空间容纳所有的可能，算法好的情况下冲突碰撞的概率很：

注意：

1. 以上不能保证数据被恶意篡改，原始数据和哈希值都可能被恶意篡改，要保证不被篡改，可以使用RA 公钥私钥方案，再配合哈希值。

2. 哈希算法主要用来防止计算机传输过程中的错误，早期计算机通过前 7 位数据第 8 位奇偶校验码保障（12.5% 的浪费效率低），对于一段数据或文件，通过哈希算法生成 128bit 或者 256bit 的哈值，如果校验有问题就要求重传。

10、常见问题

使用 cookie 时需要考虑的问题

- 因为存储在客户端，容易被客户端篡改，使用前需要验证合法性
- 不要存储敏感数据，比如用户密码，账户余额
- 使用 httpOnly 在一定程度上提高安全性
- 尽量减少 cookie 的体积，能存储的数据量不能超过 4kb
- 设置正确的 domain 和 path，减少数据传输
- **cookie 无法跨域**
- 一个浏览器针对一个网站最多存 20 个Cookie，浏览器一般只允许存放 300 个Cookie
- **移动端对 cookie 的支持不是很好，而 session 需要基于 cookie 实现，所以移动端常用的是 token**

使用 session 时需要考虑的问题

- 将 session 存储在服务器里面，当用户同时在线量比较多时，这些 session 会占据较多的内存，需在服务端定期的去清理过期的 session
- 当网站采用 **集群部署**的时候，会遇到多台 web 服务器之间如何做 session 共享的问题。因为 sess on 是由单个服务器创建的，但是处理用户请求的服务器不一定是那个创建 session 的服务器，那么服务器就无法拿到之前已经放入到 session 中的登录凭证之类的信息了。
- 当多个应用要共享 session 时，除了以上问题，还会遇到跨域问题，因为不同的应用可能部署的主不一样，需要在各个应用做好 cookie 跨域的处理。
- **sessionId 是存储在 cookie 中的，假如浏览器禁止 cookie 或不支持 cookie 怎么办？** 一般会把 s ssionId 跟在 url 参数后面即重写 url，所以 session 不一定非得需要靠 cookie 实现
- **移动端对 cookie 的支持不是很好，而 session 需要基于 cookie 实现，所以移动端常用的是 token**

使用 token 时需要考虑的问题

- 如果你认为用数据库来存储 token 会导致查询时间太长，可以选择放在内存当中。比如 redis 很适你对 token 查询的需求。
- **token 完全由应用管理，所以它可以避开同源策略**
- **token 可以避免 CSRF 攻击(因为不需要 cookie 了)**
- **移动端对 cookie 的支持不是很好，而 session 需要基于 cookie 实现，所以移动端常用的是 token**

使用 JWT 时需要考虑的问题

- 因为 JWT 并不依赖 Cookie 的，所以你可以使用任何域名提供你的 API 服务而不需要担心跨域资共享问题（CORS）
- JWT 默认是不加密，但也是可以加密的。生成原始 Token 以后，可以用密钥再加密一次。
- JWT 不加密的情况下，不能将秘密数据写入 JWT。
- JWT 不仅可以用于认证，也可以用于交换信息。有效使用 JWT，可以降低服务器查询数据库的次。
- JWT 最大的优势是服务器不再需要存储 Session，使得服务器认证鉴权业务可以方便扩展。但这也 JWT 最大的缺点：由于服务器不需要存储 Session 状态，因此使用过程中无法废弃某个 Token 或者改 Token 的权限。也就是说一旦 JWT 签发了，到期之前就会始终有效，除非服务器部署额外的逻辑。

- JWT 本身包含了认证信息，一旦泄露，任何人都可以获得该令牌的所有权限。为了减少盗用，JWT 有效期应该设置得比较短。对于一些比较重要的权限，使用时应该再次对用户进行认证。
- JWT 适合一次性的命令认证，颁发一个有效期极短的 JWT，即使暴露了危险也很小，由于每次操都会生成新的 JWT，因此也没必要保存 JWT，真正实现无状态。
- 为了减少盗用，JWT 不应该使用 HTTP 协议明码传输，要使用 HTTPS 协议传输。

使用加密算法时需要考虑的问题

- 绝不要以 **明文存储**密码
- **永远使用 哈希算法 来处理密码，绝不要使用 Base64 或其他编码方式来存储密码，这和以明文存密码是一样的，使用哈希，而不要使用编码。**编码以及加密，都是双向的过程，而密码是保密的，应只被它的所有者知道， 这个过程必须是单向的。哈希正是用于做这个的，从来没有解哈希这种说法，但是编码就存在解码，加密就存在解密。
- 绝不要使用弱哈希或已被破解的哈希算法，像 MD5 或 SHA1 ，只使用强密码哈希算法。
- 绝不要以明文形式显示或发送密码，即使是对密码的所有者也应该这样。如果你需要 “忘记密码”的功能，可以随机生成一个新的 **一次性的**（这点很重要）密码，然后把这个密码发送给用户。

分布式架构下 session 共享方案

1. session 复制

- 任何一个服务器上的 session 发生改变（增删改），该节点会把这个 session 的所有内容序列化，后广播给所有其它节点，不管其他服务器需不需要 session ，以此来保证 session 同步

优点： 可容错，各个服务器间 session 能够实时响应。

缺点： 会对网络负荷造成一定压力，如果 session 量大的话可能会造成网络堵塞，拖慢服务器性能。

2. 粘性 session /IP 绑定策略

- **采用 Ngnix 中的 ip_hash 机制，将某个 ip的所有请求都定向到同一台服务器上，即将用户与服务绑定。** 用户第一次请求时，负载均衡器将用户的请求转发到了 A 服务器上，如果负载均衡器设置了性 session 的话，那么用户以后的每次请求都会转发到 A 服务器上，相当于把用户和 A 服务器粘到一块，这就是粘性 session 机制。

优点： 简单，不需要对 session 做任何处理。

缺点： 缺乏容错性，如果当前访问的服务器发生故障，用户被转移到第二个服务器上时，他的 session 信息都将失效。

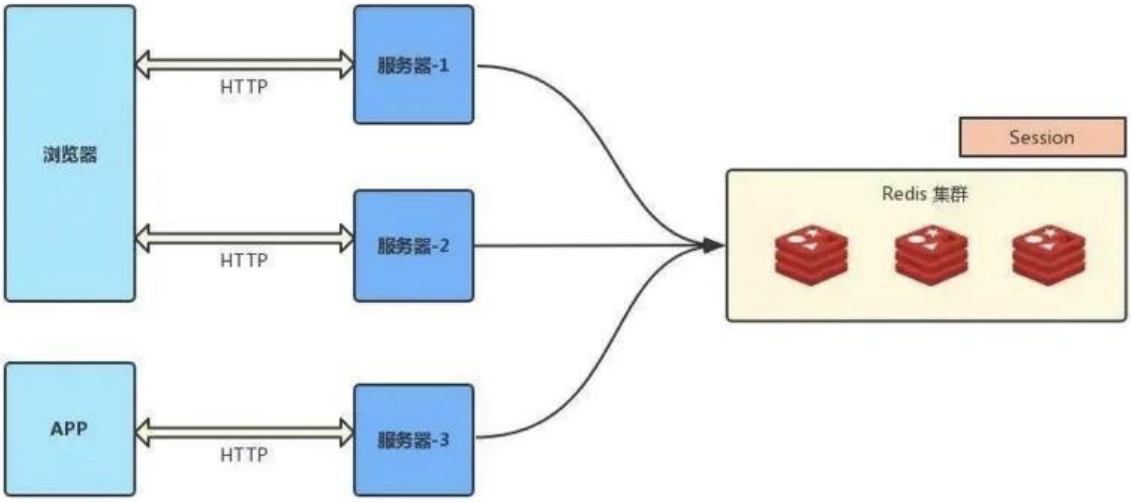
适用场景： 发生故障对客户产生的影响较小；服务器发生故障是低概率事件。

实现方式： 以 Nginx 为例，在 upstream 模块配置 ip_hash 属性即可实现粘性 session。

3. session 共享（常用）

- 使用分布式缓存方案比如 Memcached 、Redis 来缓存 session，但是要求 Memcached 或 Redis 必须是集群
- 把 session 放到 Redis 中存储，虽然架构上变得复杂，并且需要多访问一次 Redis ，但是这种方案来的好处也是很大的：

- 实现了 session 共享；
  - 可以水平扩展（增加 Redis 服务器）；
  - 服务器重启 session 不丢失（不过也要注意 session 在 Redis 中的刷新/失效机制）；
  - 不仅可以跨服务器 session 共享，甚至可以跨平台（例如网页端和 APP 端）



4. session 持久化

- 将 session 存储到数据库中，保证 session 的持久化

优点： 服务器出现问题，session 不会丢失

缺点： 如果网站的访问量很大，把 session 存储到数据库中，会对数据库造成很大压力，还需要增加外的开销维护数据库。

只要关闭浏览器，session 真的就消失了？

不对。对 session 来说，除非程序通知服务器删除一个 session，否则服务器会一直保留，程序一般是在用户做 log off 的时候发个指令去删除 session。

然而浏览器从来不会主动在关闭之前通知服务器它将要关闭，因此服务器根本不会有机会知道浏览器经关闭，之所以会有这种错觉，是大部分 session 机制都使用会话 cookie 来保存 session id，而关浏览器后这个 session id 就消失了，再次连接服务器时也就无法找到原来的 session。

如果服务器设置的 cookie 被保存在硬盘上，或者使用某种手段改写浏览器发出的 HTTP 请求头，把来的 session id 发送给服务器，则再次打开浏览器仍然能够打开原来的 session。

恰恰是**由于关闭浏览器不会导致 session 被删除，迫使服务器为 session 设置了一个失效时间，当离客户端上一次使用 session 的时间超过这个失效时间时，服务器就认为客户端已经停止了活动，才把 session 删除以节省存储空间。**

项目地址：https://github.com/yjdjiayou/jwt-demo

本文只是基于自己的理解讲了理论知识，因为对后端/算法知识不是很熟，如有谬误，还请告知，万感谢！

原文链接：Cookie、Session、Token、JWT 都是什么？今天我们彻底了解一下！