Atitit [常用的登录认证法](https://www.cnblogs.com/fengzheng/p/8416393.html)

[目录](https://www.cnblogs.com/fengzheng/p/8416393.html)

[2. 表单验证 1](#_Toc10564)

[3. OAuth 认证 1](#_Toc22331)

[4. Web票据模式验证 1](#_Toc13663)

[4.1. Token验证 1](#_Toc30854)

[4.2. Cookie-Session 认证 1](#_Toc5448)

[4.3. Cookie-Session 改造版 2](#_Toc20366)

[4.4. 基于JWT的Token认证 2](#_Toc25995)

[5. HTTP Basic Authentication 2](#_Toc12758)

[6. Other 2](#_Toc19487)

[6.1. 生物数据验证 2](#_Toc16884)

[7. Ref 2](#_Toc3030)

# [表单验证](https://www.cnblogs.com/fengzheng/p/8416393.html)

# 表单验证

# ****OAuth 认证****

# ****Web票据模式验证****

## ****Tok****en验证

## Cook****ie-Session 认证****

## ****Cookie-Session 改造版****

由于传统的 Cookie-Session 认证存在诸多问题，可以把上面的方案改造一下。改动的地方如下：

* 不用 cookie 做客户端存储，改用其他方式，web 下使用 local storage，APP 中使用客户端数据库，这样就实现了跨域，并且避免了 CSRF ;
* 服务端也不存 Session 了，把 Session 信息拿出来存到 Redis 等内存数据库中，这样即提高了速度，又避免了 Session 同步问题；

经过改造之后变成了如下的认证过程：

1. 用户输入用户名、密码或者用短信验证码方式登录系统；
2. 服务端经过验证，将认证信息构造好的数据结构存储到 Redis 中，并将 key 值返回给客户端；
3. 客户端拿到返回的 key，存储到 local storage 或本地数据库；
4. 下次客户端再次请求，把 key 值附加到 header 或者 请求体中；
5. 服务端根据获取的 key，到 Redis 中获取认证信息；

## 基于JWT的Token认证

上面的方案虽然经过了改版，但还是需要客户端和服务器端维持一个状态信息，比如用 cookie 换 session ,或者用 key 换 Redis 的 value 信息，基于 JWT 的 Token 认证方案可以省去这个过程。

JSON Web Token（JWT）是一个非常轻巧的规范。这个规范允许我们使用JWT在用户和服务器之间传递安全可靠的信息

说一说几种常用的登录认证方式，你用的哪种 - 风的姿态 - 博客园.html

# ****HTTP Basic Authentication****

# ****Other****

## 生物数据验证

# Ref

前端的登录验证是怎么实现的？掌握这4种方法，再也不担心鉴权了 – 柚柚科技.html