JWT学习文档

# 官网

<https://jwt.io/>

# 入门

<https://www.cnblogs.com/wangshouchang/p/9551748.html>

## 什么是JWT

Json web token (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).该token被设计为紧凑且安全的，特别适用于分布式站点的单点登录（SSO）场景。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取资源，也可以增加一些额外的其它业务逻辑所必须的声明信息，该token也可直接被用于认证，也可被加密。

**简单来说就是** JWT(Json Web Token)是实现token技术的一种解决方案

### 为什么使用JWT

**token验证和session认证的区别**

传统的session认证

http协议本身是一种无状态的协议，而这就意味着如果用户向我们的应用提供了用户名和密码来进行用户认证，那么下一次请求时，用户还要再一次进行用户认证才行，因为根据http协议，我们并不能知道是哪个用户发出的请求，所以为了让我们的应用能识别是哪个用户发出的请求，我们只能在服务器存储一份用户登录的信息，这份登录信息会在响应时传递给浏览器，告诉其保存为cookie,以便下次请求时发送给我们的应用，这样我们的应用就能识别请求来自哪个用户了,这就是传统的基于session认证。

session缺点

基于session的认证使应用本身很难得到扩展，随着不同客户端用户的增加，独立的服务器已无法承载更多的用户

Session方式存储用户id的最大弊病在于要占用大量服务器内存，对于较大型应用而言可能还要保存许多的状态。

**基于session认证暴露的问题**

* **Session**需要在服务器保存，暂用资源
* **扩展性** session认证保存在内存中 ，无法扩展到其他机器中
* **CSRF** 基于cookie来进行用户识别的, cookie如果被截获，用户就会很容易受到跨站请求伪造的攻击。

### 基于token的鉴权机制

基于token的鉴权机制类似于http协议也是无状态的，它不需要在服务端去保留用户的认证信息或者会话信息。这就意味着基于token认证机制的应用不需要去考虑用户在哪一台服务器登录了，这就为应用的扩展提供了便利。

JWT方式将用户状态分散到了客户端中，可以明显减轻服务端的内存压力。除了用户id之外，还可以存储其他的和用户相关的信息，例如用户角色，用户性别等。

**请求流程**

1. 用户使用用户名密码来请求服务器
2. 服务器进行验证用户的信息
3. 服务器通过验证发送给用户一个token
4. 客户端存储token，并在每次请求时附送上这个token值
5. 服务端验证token值，并返回数据

这个token必须要在每次请求时传递给服务端，它应该保存在请求头里， 另外，服务端要支持 CORS(跨来源资源共享)策略，一般我们在服务端这么做就可以了 Access-Control-Allow-Origin: \*。

### JWT的结构

一个JWT是下面的结构

加密后jwt信息如下所示，是由.分割的三部分组成，分别为Header、Payload、Signature

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.

eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.

TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

**JWT 的组成**

* Head -主要包含两个部分,alg指加密类型，可选值为HS256、RSA等等，typ=JWT为固定值，表示token的类型
* Header:
* {
* "alg": "HS256",
* "typ": "JWT"

}

* Payload - Payload又被称为Claims包含您想要签署的任何信息
* Claims:
* {
* "sub": "1234567890",
* "name": "John Doe",
* "admin": true

}

JWT Payload的组成

Payload通常由三个部分组成，分别是 Registered Claims ; Public Claims ; Private Claims ;每个声明，都有各自的字段。

Registered Claims

iss 【issuer】发布者的url地址

sub 【subject】该JWT所面向的用户，用于处理特定应用，不是常用的字段

aud 【audience】接受者的url地址

exp 【expiration】 该jwt销毁的时间；unix时间戳

nbf 【not before】 该jwt的使用时间不能早于该时间；unix时间戳

iat 【issued at】 该jwt的发布时间；unix 时间戳

jti 【JWT ID】 该jwt的唯一ID编号

* Signature 对 则为对Header、Payload的签名
* Signature:

base64UrlEncode(Header) + "." + base64UrlEncode(Claims)

**头部、声明、签名用 . 号连在一起就得到了我们要的JWT** 也就是夏明这种类型的字符串

eyJhbGciOiJIUzI1NiJ9.

eyJleHAiOjE1MTUyOTgxNDEsImtleSI6InZhdWxlIn0.

orewTmil7YmIXKILHwFnw3Bq1Ox4maXEzp0NC5LRaFQ

其实这些事一行的，我只是让看的更直白点将其割开了。

### JAVA 实现

JAVA中使用JWT

使用Maven引入和Gradle引入

Maven

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

**JWT依赖于Jackson**

**签发JWT**

public static String createJWT() {

JwtBuilder builder = Jwts.builder()

.setId(id) // JWT\_ID

.setAudience("") // 接受者

.setClaims(null) // 自定义属性

.setSubject("") // 主题

.setIssuer("") // 签发者

.setIssuedAt(new Date()) // 签发时间

.setNotBefore(new Date()) // 失效时间

.setExpiration(long) // 过期时间

.signWith(SignatureAlgorithm.***HS512***, secretKey); // 签名算法以及密匙

return builder.compact();

}

**验证JWT**

**public** **static** Claims getClaim(String jwt) {

**return** Jwts.*parser*().setSigningKey(***SECRET***).parseClaimsJws(jwt).getBody();

}

**完整示例**

**pom.xml中的依赖**

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

<version>3.3.2</version>

</dependency>

</dependencies>

package org.lys.demo.jwt;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import org.apache.commons.lang3.StringUtils;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.ExpiredJwtException;

import io.jsonwebtoken.JwtBuilder;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

/\*\*

\* JWT工具类

\* @author lys

\*

\*/

public class JWTUtil {

private static final long EXPIRE = 60 \* 1000;

private static final String SECRET = "9RlGH4qWGQKjOkJj";

public static final String UID = "uid";

/\*\*

\*

\* @param subject

\* @param claims

\* @param key

\* @param issuedDate

\* @param expireDate

\* @return

\*/

public static String createJWT(String id, String subject, Map<String, Object> claims, String key, Date issuedDate,

Date expireDate) {

JwtBuilder builder = Jwts.builder();

if (StringUtils.isNoneBlank(id)) {

builder.setId(id);// 设置jti(JWT// ID)：是JWT的唯一标识，根据业务需要，这个可以设置为一个不重复的值，主要用来作为一次性token,从而回避重放攻击。

}

if (StringUtils.isNoneBlank(subject)) {

builder.setSubject(subject);// sub(Subject)：代表这个JWT的主体，即它的所有人，这个是一个json格式的字符串，可以存放什么userid，roldid之类的，作为什么用户的唯一标志。

}

if (claims != null && claims.size() > 0) {

builder.setClaims(claims);// 如果有私有声明，一定要先设置这个自己创建的私有的声明，这个是给builder的claim赋值，一旦写在标准的声明赋值之后，就是覆盖了那些标准的声明的

}

if (issuedDate != null) {

builder.setIssuedAt(issuedDate);// 签发时间

}

if (expireDate != null) {

builder.setExpiration(expireDate); // 设置过期时间

}

builder.signWith(SignatureAlgorithm.HS512, key);// 设置签名使用的签名算法和签名使用的秘钥

return builder.compact();

}

/\*\*

\* 生成

\*

\* @param uid

\* @return

\*/

public static String createToken(String uid) {

Date nowDate = new Date();

// 过期时间

Date expireDate = new Date(nowDate.getTime() + EXPIRE \* 1000);

Map<String, Object> claims = new HashMap<>(1);

claims.put(UID, uid);

return createJWT(null, null, claims, SECRET, nowDate, expireDate);

}

/\*\*

\* 获取Claims

\* @param jwt

\* @return

\*/

public static Claims getClaim(String jwt) {

return Jwts.parser().setSigningKey(SECRET).parseClaimsJws(jwt).getBody();

}

/\*\*

\* 用户id

\* @param token

\* @return

\*/

public static String getUid(String token) {

return getClaim(token).get(UID).toString();

}

/\*\*

\* 过期时间

\* @param token

\* @return

\*/

public static Date getExpiration(String token) {

return getClaim(token).getExpiration();

}

/\*\*

\* 是否过期

\* @param token

\* @return

\*/

public static boolean isExpired(String token) {

try {

final Date expiration = getExpiration(token);

return expiration.before(new Date());

} catch (ExpiredJwtException expiredJwtException) {

return true;

}

}

/\*\*

\* 签发时间

\* @param token

\* @return

\*/

public static Date getIssuedAt(String token) {

return getClaim(token).getIssuedAt();

}

public static void main(String[] args) {

String token = JWTUtil.*createToken*("user");

System.***out***.println(token);

String uid = JWTUtil.*getUid*(token);

System.***out***.println(uid);

System.***out***.println(JWTUtil.*getExpiration*(token));

System.***out***.println(JWTUtil.*getIssuedAt*(token));

System.***out***.println(JWTUtil.*isExpired*(token));

}

}

**输出示例**

eyJhbGciOiJIUzUxMiJ9.eyJleHAiOjE1NTM2Njc1MzEsInVpZCI6InVzZXIiLCJpYXQiOjE1NTM2MDc1MzF9.4xhUY0PeX51nZxrpoYlpcyES4mqZK175W6cVpIeJgraVxWs720\_6PcUjgvqqcXMhdqWvsIjysdy\_87HTNHAnJQ

user

Wed Mar 27 14:18:51 CST 2019

Tue Mar 26 21:38:51 CST 2019

false

## 总结

**优点**

* 因为json的通用性，所以JWT是可以进行跨语言支持的，像JAVA,JavaScript,NodeJS,PHP等很多语言都可以使用。
* 因为有了payload部分，所以JWT可以在自身存储一些其他业务逻辑所必要的非敏感信息。
* 便于传输，jwt的构成非常简单，字节占用很小，所以它是非常便于传输的。
* 它不需要在服务端保存会话信息, 所以它易于应用的扩展

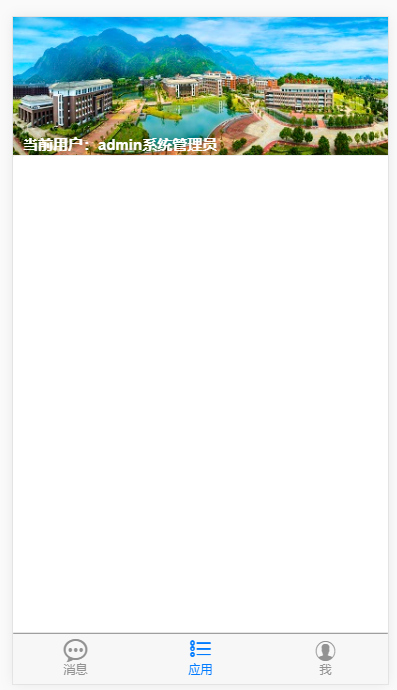
**安全相关**

* 不应该在jwt的payload部分存放敏感信息，因为该部分是客户端可解密的部分。
* 保护好secret私钥，该私钥非常重要。
* 如果可以，请使用https[协议](https://www.cnblogs.com/wangshouchang/p/9551748.html)

# 实际项目中应用

## 场景介绍

用户登录成功转向主界面，主界面显示当前的账号信息。



## 设计思路

1. 用户登录时，验证成功用户信息后，使用JWT生成token，将token返回给登录界面。
2. 登录界面接收到登录成功返回的token后，将token存放到本地缓存localStorage
3. 主界面发送ajax请求（header携带token）：取当前用户信息
4. 添加拦截器，拦截请求，解析token，如果解析正常，则解析到的用户存放到当前线程ThreadLocal中
5. 控制层对应的方法中，取得当前线程中的用户信息，并返回

## 具体实现

### 登录

$("#loginBtn").click(**function**(){

$.post("../user/api/login.jhtml", { username: $("#username").val(), password: $("#password").val() },

**function**(res){

**if**(res.isSuccess){

localStorage.token=res.data.token;

window.location.href='main.html';

}**else**{

alert(res.message)

}

},'json');

})

用户登录时，验证成功用户信息后，使用JWT生成token，将token返回给登录界面。

//生成Token

JWTInfo jwtInfo = **new** JWTInfo(user.getId().toString(),user.getUsername(),user.getName());

String token = JWTUtil.*createToken*(jwtInfo);

result.addData("token", token);

result.setIsSuccess(**true**);

**return** result;

package top.linyisong.core.jwt;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import org.apache.commons.lang3.StringUtils;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.ExpiredJwtException;

import io.jsonwebtoken.JwtBuilder;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import top.linyisong.core.utils.CommonUtil;

/\*\*

\* JWT工具类

\* @author lys

\*

\*/

public class JWTUtil {

private static final long EXPIRE = 60 \* 1000;

private static final String SECRET = "9RlGH4qWGQKjOkJj";

public static final String UID = "uid";

/\*\*

\*

\* @param subject

\* @param claims

\* @param key

\* @param issuedDate

\* @param expireDate

\* @return

\*/

public static String createJWT(String id, String subject, Map<String, Object> claims, String key, Date issuedDate,

Date expireDate) {

JwtBuilder builder = Jwts.builder();

if (StringUtils.isNoneBlank(id)) {

builder.setId(id);// 设置jti(JWT// ID)：是JWT的唯一标识，根据业务需要，这个可以设置为一个不重复的值，主要用来作为一次性token,从而回避重放攻击。

}

if (StringUtils.isNoneBlank(subject)) {

builder.setSubject(subject);// sub(Subject)：代表这个JWT的主体，即它的所有人，这个是一个json格式的字符串，可以存放什么userid，roldid之类的，作为什么用户的唯一标志。

}

if (claims != null && claims.size() > 0) {

builder.setClaims(claims);// 如果有私有声明，一定要先设置这个自己创建的私有的声明，这个是给builder的claim赋值，一旦写在标准的声明赋值之后，就是覆盖了那些标准的声明的

}

if (issuedDate != null) {

builder.setIssuedAt(issuedDate);// 签发时间

}

if (expireDate!= null) {

builder.setExpiration(expireDate); // 设置过期时间

}

builder.signWith(SignatureAlgorithm.HS512, key);// 设置签名使用的签名算法和签名使用的秘钥

return builder.compact();

}

/\*\*

\* 生成

\*

\* @param uid

\* @return

\*/

public static String createToken(String uid) {

Date nowDate = new Date();

// 过期时间

Date expireDate = new Date(nowDate.getTime() + EXPIRE \* 1000);

Map<String, Object> claims = new HashMap<>(1);

claims.put(UID, uid);

return createJWT(null, null, claims, SECRET, nowDate, expireDate);

}

public static String createToken(JWTInfo jwtInfo) {

Date nowDate = new Date();

// 过期时间

Date expireDate = new Date(nowDate.getTime() + CommonUtil.getIntegerAppProParam("business.jwtExpire") \* 1000);

Map<String, Object> claims = new HashMap<>(1);

claims.put("userId", jwtInfo.getId());

claims.put("username", jwtInfo.getUsername());

claims.put("name", jwtInfo.getName());

return createJWT(null, null, claims, CommonUtil.getAppProParam("business.jwtsecret"), nowDate, expireDate);

}

public static String createToken(JWTInfo jwtInfo,String key, int expire) {

Date nowDate = new Date();

// 过期时间

Date expireDate = new Date(nowDate.getTime() + expire \* 1000);

Map<String, Object> claims = new HashMap<>(1);

claims.put("userId", jwtInfo.getId());

claims.put("username", jwtInfo.getUsername());

claims.put("name", jwtInfo.getName());

return createJWT(null, null, claims, key, nowDate, expireDate);

}

/\*\*

\* 获取Claims

\* @param jwt

\* @return

\*/

public static Claims getClaim(String jwt) {

String secret = CommonUtil.getAppProParam("business.jwtsecret");

if(StringUtils.isBlank(secret)){

secret = SECRET;

}

return Jwts.parser().setSigningKey(secret).parseClaimsJws(jwt).getBody();

}

/\*\*

\* 获取Claims

\* @param jwt

\* @param key

\* @return

\*/

public static Claims getClaim(String token,String key) {

return Jwts.parser().setSigningKey(key).parseClaimsJws(token).getBody();

}

/\*\*

\* 用户id

\* @param token

\* @return

\*/

public static String getUid(String token) {

return getClaim(token).get(UID).toString();

}

public static JWTInfo getJWTInfo(String token) {

String secret = CommonUtil.getAppProParam("business.jwtsecret");

if(StringUtils.isBlank(secret)){

secret = SECRET;

}

Claims claims = getClaim(token,secret);

String username = claims.get("username").toString();

String userId = claims.get("userId").toString();

String name = claims.get("name").toString();

return new JWTInfo(userId,username , name);

}

/\*\*

\* 过期时间

\* @param token

\* @return

\*/

public static Date getExpiration(String token) {

return getClaim(token).getExpiration();

}

/\*\*

\* 是否过期

\* @param token

\* @return

\*/

public static boolean isExpired(String token) {

try {

final Date expiration = getExpiration(token);

return expiration.before(new Date());

} catch (ExpiredJwtException expiredJwtException) {

return true;

}

}

/\*\*

\* 签发时间

\* @param token

\* @return

\*/

public static Date getIssuedAt(String token) {

return getClaim(token).getIssuedAt();

}

public static void main(String[] args) {

String token = "eyJhbGciOiJIUzUxMiJ9.eyJleHAiOjE1NTMxMjMzNjQsInVpZCI6IjMzMzMiLCJpYXQiOjE1NTMwNjMzNjR9.s52EdId95odQoNoQkgreVb2Z9j3GGLswIavQK5BrAnZl01haFEjuihhTgf3IkMH7hRoZJ6ME0\_ESBlhoeEo1nQ";

String uid = JWTUtil.getUid(token);

System.out.println(uid);

System.out.println(JWTUtil.getExpiration(token));

System.out.println(JWTUtil.getIssuedAt(token));

System.out.println(JWTUtil.isExpired(token));

}

}

**package** top.linyisong.core.jwt;

**public** **class** JWTInfo{

**private** String userId;

**private** String username;

**private** String name;

**public** JWTInfo(String username, String userId, String name) {

**this**.username = username;

**this**.userId = userId;

**this**.name = name;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getId() {

**return** userId;

}

**public** **void** setUserId(String userId) {

**this**.userId = userId;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

@Override

**public** **boolean** equals(Object o) {

**if** (**this** == o) {

**return** **true**;

}

**if** (o == **null** || getClass() != o.getClass()) {

**return** **false**;

}

JWTInfo jwtInfo = (JWTInfo) o;

**if** (username != **null** ? !username.equals(jwtInfo.username) : jwtInfo.username != **null**) {

**return** **false**;

}

**return** userId != **null** ? userId.equals(jwtInfo.userId) : jwtInfo.userId == **null**;

}

@Override

**public** **int** hashCode() {

**int** result = username != **null** ? username.hashCode() : 0;

result = 31 \* result + (userId != **null** ? userId.hashCode() : 0);

**return** result;

}

**public** String getUsername() {

**return** username;

}

}

### 主界面

$.ajaxSetup({

headers: {

'accessToken': localStorage.token

}

});

$.post("../user/api/selfInfo.jhtml",{},

**function**(res){

**if**(res.isSuccess){

baseInfoData.username = res.data.user.userName;

baseInfoData.name = res.data.user.name;

}**else**{

alert(res.message)

}

},'json');

### 拦截器

**package** top.linyisong.core.interceptor;

**import** java.io.IOException;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.web.method.HandlerMethod;

**import** org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

**import** com.baomidou.mybatisplus.toolkit.StringUtils;

**import** top.linyisong.core.annotation.AuthIgnore;

**import** top.linyisong.core.dto.UserLoginInfoDTO;

**import** top.linyisong.core.jwt.JWTInfo;

**import** top.linyisong.core.jwt.JWTUtil;

**import** top.linyisong.core.vo.ServiceResult;

**import** top.linyisong.core.vo.ThreadLocalSession;

**public** **class** AuthIntercepter **extends** HandlerInterceptorAdapter {

**private** **static** Logger *logger* = LoggerFactory.*getLogger*(AuthIntercepter.**class**);

@Override

**public** **boolean** preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler) {

String accessToken = **null**;

accessToken = request.getHeader("accessToken");

**if** (handler **instanceof** HandlerMethod) {

AuthIgnore authIgnoreAnnotation = ((HandlerMethod) handler).getMethodAnnotation(AuthIgnore.**class**);

**if** (authIgnoreAnnotation != **null**&&StringUtils.*isEmpty*(accessToken)) {

**return** **true**;

}

}**else**{

**return** **true**;

}

**if** (StringUtils.*isEmpty*(accessToken)) {

errorResponse(response, "为获取accessToken值");

**return** **false**;

}

**try** {

JWTInfo jwInfo = JWTUtil.*getJWTInfo*(accessToken);

**if** (jwInfo == **null**) {

*logger*.error("token解析失败");

errorResponse(response, "token解析失败");

**return** **false**;

}

UserLoginInfoDTO userInfo = **new** UserLoginInfoDTO();

userInfo.setId(jwInfo.getId());

userInfo.setUserName(jwInfo.getUsername());

userInfo.setName(jwInfo.getName());

ThreadLocalSession.*set*(userInfo);

**return** **super**.preHandle(request, response, handler);

} **catch** (Exception e1) {

e1.printStackTrace();

*logger*.error("token解析失败");

errorResponse(response, "token解析失败");

**return** **false**;

}**finally** {

}

}

/\*\*

\* 写入异常信息

\*

\* **@param** response response对象

\* **@param** errorMsg 错误信息

\*/

**private** **void** errorResponse(HttpServletResponse response, String errorMsg) {

ServiceResult result = **new** ServiceResult(**false**);

result.setMessage(errorMsg);

response.setCharacterEncoding("UTF-8");

response.setContentType("application/json;charset=UTF-8");

**try** {

response.getWriter().write(result.toJSON());

} **catch** (IOException e) {

e.printStackTrace();

}

}

@Override

**public** **void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) **throws** Exception {

ThreadLocalSession.*remove*();

**super**.afterCompletion(request, response, handler, ex);

}

}

/\*\*

\* 登录排除注解

\*

\* **@author** tjianqun@linewell.com

\*

\* **@date** 2017年7月20日 上午11:09:32

\*/

@Target(ElementType.***METHOD***)

@Retention(RetentionPolicy.***RUNTIME***)

@Documented

**public** **@interface** AuthIgnore {

}

**public** **class** UserLoginInfoDTO {

**private** String id;

**private** String userName;

**private** String name;

**public** String getId() {

**return** id;

}

**public** **void** setId(String id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getUserName() {

**return** userName;

}

**public** **void** setUserName(String userName) {

**this**.userName = userName;

}

}

/\*\*

\* 登录信息保存

\*

\* **@author** tjianqun@linewell.com

\*

\* **@date** 2017年5月19日 下午2:54:27

\*/

**public** **class** ThreadLocalSession {

**private** **static** ThreadLocal<UserLoginInfoDTO> *local* = **new** ThreadLocal<UserLoginInfoDTO>();

**public** **static** **void** set(UserLoginInfoDTO info){

*local*.set(info);

}

**public** **static** UserLoginInfoDTO get(){

UserLoginInfoDTO dto = *local*.get();

**return** dto;

}

**public** **static** **void** remove(){

*local*.remove();

}

}

### 取得当前用户

@RequestMapping(value = "/selfInfo")

@ResponseBody

**public** Object selfInfo() {

ServiceResult result = **new** ServiceResult(**false**);

UserLoginInfoDTO dto = ThreadLocalSession.*get*();

result.addData("user", dto);

result.setIsSuccess(**true**);

**return** result;

}