# 尚马教育 JAVA 高级课程

# JJWT组件与token使用

文档编号：C10\_2

创建日期： 2021-02-26

最后修改日期：2021-02-26

版 本 号：V1.0

电子版文件名：尚马教育-第三阶段-10\_2.jjwt组件与token使用.docx

**文档修改记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| 更新日期 | 更新作者 | 更新说明 | 版本号 |
| 2021-02-26 | 冯勇涛 | 认识token，使用jjwt，项目使用 | V1.0 |

目录

[尚马教育 JAVA 高级课程 1](#_Toc16566)

[JJWT组件与token使用 1](#_Toc24337)

[1. Token介绍 3](#_Toc24764)

[1.1. 概念 3](#_Toc9660)

[1.2. token作用 3](#_Toc6184)

[2. Jjwt组件 3](#_Toc24142)

[2.1. 添加依赖 4](#_Toc20110)

[2.2. 生成秘钥 4](#_Toc9448)

[2.3. 生成token并加密 5](#_Toc7932)

[2.4. 解析token 6](#_Toc8713)

[2.5. 测试工具类 7](#_Toc19191)

[3. 登陆接口 8](#_Toc26650)

[3.1. 首次登录服务端生成token发送客户端 8](#_Toc32342)

[3.2. 客户端接收服务端返回的token并保存 9](#_Toc4711)

[3.3. 客户端后续请求携带token发送服务端 10](#_Toc13018)

[3.4. 服务端拦截器进行token校验 11](#_Toc22721)

[3.5. 客户端axios后拦截器保存token 12](#_Toc14781)

[4. 注销接口 12](#_Toc9571)

[4.1. 用户注销登录 12](#_Toc25234)

[5. 安全问题 13](#_Toc26321)

知识点：

|  |
| --- |
| 1. 掌握token概念 2. 掌握jjwt组件使用 3. 项目引入token进行客户端校验 |

## Token介绍

### 概念

Token是当用户第一次访问服务端，由服务端生成的一串加密字符串，以作后续客户端进行请求的一个通行令牌。当第一次登录后，服务器生成一个Token字符串，并将此字符串返回给客户端，以后客户端请求需要带上这个Token发送服务器，进行请求数据即可，无需再次带上用户名和密码。

类似于城主发放的路引，进城需要出示路引做身份证明。

### token作用

Token的目的是为了减轻服务器的压力，减少频繁的查询数据库进行客户端校验，使服务器更加健壮。

在tomcat集群环境下，解决基于内存的session共享问题。

## Jjwt组件

生成解析token字符串的常用组件有auth0，jjwt，这里选用jjwt进行学习。

JJWT旨在成为最易于使用和理解的库，用于在jvm和Android上创建和验证JSON Web Token（JWT）。对JWT进行加密签名后，称为JWS

官网：<https://github.com/jwtk/jjwt>

jWT表示形式是一个字符串，该字符串包含三个部分，每个部分之间都用.进行分隔，每个部分都是[Base64URL](https://en.wikipedia.org/wiki/Base64)编码的。如下：

|  |
| --- |
| eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJKb2UifQ.ipevRNuRP6HflG8cFKnmUPtypruRC4fb1DWtoLL62SY |

第一部分：header标头，需要指定用于签署JWT的算法

{

"alg": "HS256"

}

第二部分：body身体，jwt中需要包含的Claims认证数据，claims分为标准cliams与自定义。

{

"sub": "Joe"

}

第三部分：签名，它是通过将标头和正文的组合通过标头中指定的算法加密来计算的。起到鉴伪作用。

### 添加依赖

|  |
| --- |
| <dependency>  <groupId>io.jsonwebtoken</groupId>  <artifactId>jjwt-api</artifactId>  <version>0.11.2</version>  </dependency>  <dependency>  <groupId>io.jsonwebtoken</groupId>  <artifactId>jjwt-impl</artifactId>  <version>0.11.2</version>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>io.jsonwebtoken</groupId>  <artifactId>jjwt-jackson</artifactId> <!-- or jjwt-gson if Gson is preferred -->  <version>0.11.2</version>  <scope>runtime</scope>  </dependency> |

### 生成秘钥

JWT规范确定了12种标准签名算法：3种对称密钥算法和9种非对称密钥算法。

|  |
| --- |
| * HS256：使用SHA-256的HMAC * HS384：使用SHA-384的HMAC * HS512：使用SHA-512的HMAC * ES256：使用P-256和SHA-256的ECDSA * ES384：使用P-384和SHA-384的ECDSA * ES512：使用P-521和SHA-512的ECDSA * RS256：使用SHA-256的RSASSA-PKCS-v1\_5 * RS384：使用SHA-384的RSASSA-PKCS-v1\_5 * RS512：使用SHA-512的RSASSA-PKCS-v1\_5 * PS256：使用SHA-256和MGF1与SHA-256的RSASSA-PSS * PS384：使用SHA-384和MGF1与SHA-384的RSASSA-PSS * PS512：使用SHA-512和MGF1与SHA-512的RSASSA-PSS   前三者是对称秘钥算法，后9个是非对称算法。 |

以下代码采用HS256对称秘钥算法。

|  |
| --- |
| //生成符合HS256加密算法要求的秘钥，该秘钥要保存为常量，进行后续解密  public static void generateKey() {  Key key = Keys.secretKeyFor(SignatureAlgorithm.HS256);  String secretString = Encoders.BASE64.encode(key.getEncoded());  System.out.println(secretString);  } |

### 生成token并加密

以下代码中成的jwt中的body部分包含如下claims：

标准claim：issuedAt签发时间，expiration过期时间

自定义claim：p手机号

{issuedAt:XXXX,expiration:XXXX,p:XXX}

|  |
| --- |
| //token中含有用户信息，并使用秘钥加密  public static String generate(String uphone) {  Key key = Keys.hmacShaKeyFor(Decoders.BASE64.decode(keyStr));  Date current = new Date();  Date expirDate = new Date(current.getTime() + expirMills);  Map<String, String> claims = new HashMap<>();  claims.put("p", uphone);  String jws = Jwts.builder().setClaims(claims).setIssuedAt(current).setExpiration(expirDate).signWith(key).compact();  return jws;  } |

### 解析token

|  |
| --- |
| //解析token字符串，使用秘钥解密  public static Claims parse(String token) {  Jws<Claims> jws = Jwts.parserBuilder()  .setSigningKey(keyStr).build().parseClaimsJws(token);  return jws.getBody();  }  //获取解析后字符串中的手机号签名信息  public static String getUphone(Claims claim){  return (String)claim.get("p");  }  //获取解析后字符串中的签发时间信息  public static Date getIssuedAt(Claims claim){  return claim.getIssuedAt();  }  //获取解析后字符串中的过期时间信息  public static Date getExpiation(Claims claim){  return claim.getExpiration();  } |

### 测试工具类

首先需要使用生成秘钥方法生成秘钥字符串，保存到静态常量；

|  |
| --- |
| //保存生成的秘钥  private static String keyStr = "Y35yyCm16xZ7TwcNIAZRjDtiNc/loypN0GN331O/3r4=";  //设置超期时间为1周  private static Long expirMills = 7\*24\*60\*60\*1000L;  public static void main(String[] args) throws InterruptedException {  //1.生成token  String t = generate("13663830111");  System.out.println(t);  //2.解析token，解析失败抛出异常  Claims claims = parse(t);  //3.成功解析后，获取签名信息  String uphone = getUphone(claims);  Date expiation = getExpiation(claims);  System.out.println(uphone+"--"+expiation);  } |

## 登陆接口

### 首次登录服务端生成token发送客户端

|  |
| --- |
| //用户名密码登录  @PostMapping("login")  public ResponseEntity dologin(@RequestBody Sysuser user){  //根据用户名查询用户对象  Sysuser loginuser = us.selectUserByUname(user);  //用户名正确与否  if(loginuser==null){  return ResponseEntity.ok(new RB(SE.LOGIN\_UNAME\_ERROR));  }  //判断密码有否有误  boolean isok = us.comparseUpwd(loginuser,user);  if(!isok){  return ResponseEntity.ok(new RB(SE.LOGIN\_UPWD\_ERROR));  }  //登录成功，生成token  String token = JwtUtil.generate(loginuser.getUname());  loginuser.setUpwd("");  //把token放在响应头中返回客户端  HttpHeaders headers = new HttpHeaders();  headers.add("token",token);  return new ResponseEntity(new RB(SE.SUC,loginuser),headers,HttpStatus.OK);  } |

### Springmvc配置自定义返回响应头

3.1代码中添加的自定义响应头token，前端默认收不到该信息，需要在跨域过滤器中配置自定义响应头。

|  |
| --- |
| <bean id="corsFilter" class="org.springframework.web.filter.CorsFilter">  <constructor-arg name="configSource">  <bean class="org.springframework.web.cors.UrlBasedCorsConfigurationSource">  <property name="corsConfigurations">  <map>  <entry key="/\*\*">  <bean class="org.springframework.web.cors.CorsConfiguration">  <property name="allowCredentials" value="true"/>  <property name="allowedMethods">  <list>  <value>GET</value>  <value>POST</value>  <value>HEAD</value>  <value>PUT</value>  <value>DELETE</value>  <value>OPTIONS</value>  </list>  </property>  <property name="allowedHeaders" value="\*"/>  <property name="allowedOrigins" value="\*"/>  <!--允许客户端收到token响应头-->  <property name="exposedHeaders" value="token"></property><!--暴露的响应头-->  </bean>  </entry>  </map>  </property>  </bean>  </constructor-arg>  </bean> |

### 客户端接收token并保存

在axios的后拦截器中获取token字符串，并保存起来。

|  |
| --- |
| //axios后拦截器  \_axios.interceptors.response.use(function(resp) {  let code = resp.data.code  if(code==50004){  router.push("/")  }    if('token' in resp.headers){  //获取响应头中的token信息  let token = resp.headers['token']  if(token!=null && token!=undefined){  //把token保存locatlStorage域  localStorage.setItem("token",token);  }  }  return resp.data;  },  function(error) {  return Promise.reject(error);  }  ); |

### 客户端后续请求携带token发送服务端

前端axios请求，获取localStorage中的token加入请求头

|  |
| --- |
| //axios前拦截器  \_axios.interceptors.request.use(function(config) {  let url= config.url;  if(url=='/user/login'){  return config;  }else{  let loginuser = localStorage.getItem("loginuser")  if(loginuser==null||loginuser==undefined){  router.push("/");  }else{  let token = localStorage.getItem("token")  if(token==null || token==undefined){  router.push("/");  }else{  config.headers['token']=token;  return config;  }  }  }  },  function(error) {  return Promise.reject(error);  }  ); |

### 服务端拦截器进行token校验

|  |
| --- |
| @Override  public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {  String token = request.getHeader("token");  System.out.println(token);  if(token==null)  throw new WebException(SE.NO\_LOGIN);  //解析token  Claims claims = null;  try{  claims = JwtUtil.parse(token);  }catch (Exception e){  throw new WebException(SE.NO\_LOGIN);  }  //从redis中获取登录用户  String uname = JwtUtil.getCustomClaim(claims);  String userStr = rs.get(RedisKey.users + uname);  if(userStr==null)  throw new WebException(SE.NO\_LOGIN);    //刷新token  String newToken = JwtUtil.generate(uname);  response.addHeader("token",newToken);    Sysuser u = JSON.parseObject(userStr,Sysuser.class);  //使用线程变量把u登录用户对象保存，确保当前线程中都可以取得登录用户  CurrentLoginUser.setLoginuser(u);  return true;  } |
| public class CurrentLoginUser {  private static ThreadLocal<Sysuser> loginuser = new ThreadLocal<>();  public static void setLoginuser(Sysuser u){  loginuser.set(u);  }  public static Sysuser getLoginuser(){  return loginuser.get();  }  } |

## 注销接口

### 用户注销登录

注销操作，需要前端删除token

|  |
| --- |
| localStorage.removeItem('token');  router.push("/"); |