第6章-密码加密与微服务鉴权JWT

学习目标：

- 能够使用BCrypt密码加密算法实现注册与登陆功能
- 能够说出常见的认证机制
- 能够说出JWT的组成部分，以及使用JWT的优点
- 能够使用JJWT 创建和解析token
- 能够使用JJWT完成微服务鉴权

1 BCrypt密码加密

1.1 准备工作

任何应用考虑到安全，绝不能明文的方式保存密码。密码应该通过哈希算法进行加密。有很多标准的算法比如SHA或者MD5，结合salt(盐)是一个不错的选择。 Spring Security提供了BCryptPasswordEncoder类,实现Spring的PasswordEncoder接口使用BCrypt强哈希方法来加密密码。

BCrypt强哈希方法 每次加密的结果都不一样。

（1）tensquare_user工程的pom引入依赖

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
```

（2）添加配置类 （资源/工具类中提供）

我们在添加了spring security依赖后，所有的地址都被spring security所控制了，我们目前只是需要用到BCrypt密码加密的部分，所以我们要添加一个配置类，配置为所有地址都可以匿名访问。

```
/**
 * 安全配置类
 */
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter{
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                .antMatchers("/**").permitAll()
                .anyRequest().authenticated()
                .and().csrf().disable();
    }
}
```

（3）修改tensquare_user工程的Application，配置bean

```
    @Bean
    public BCryptPasswordEncoder bcryptPasswordEncoder(){
        return new BCryptPasswordEncoder();
    }
```

1.2 管理员密码加密

1.2.1 新增管理员密码加密

修改tensquare_user工程的AdminService

```
    @Autowired
    BCryptPasswordEncoder encoder;

    public void add(Admin admin) {
        admin.setId(idWorker.nextId()+""); //主键值
        //密码加密
        String newpassword = encoder.encode(admin.getPassword());//加密后
的密码
        admin.setPassword(newpassword);
        adminDao.save(admin);
    }
```

1.2.2 管理员登陆密码校验

（1）AdminDao增加方法定义

```
    public Admin findByLoginname(String loginname);
```

（2）AdminService增加方法

```
    /**
     * 根据登陆名和密码查询
     * @param loginname
     * @param password
     * @return
     */
    public Admin findByLoginnameAndPassword(String loginname, String
password){
        Admin admin = adminDao.findByLoginname(loginname);
        if( admin!=null && encoder.matches(password,admin.getPassword()))
{
            return admin;
        }else{
            return null;
        }
    }
```

（3）AdminController增加方法

```
    /**
     * 用户登陆
     * @param loginname
     * @param password
     * @return
     */
    @RequestMapping(value="/login",method=RequestMethod.POST)
    public Result login(@RequestBody Map<String,String> loginMap){
        Admin admin =
adminService.findByLoginnameAndPassword(loginMap.get("loginname"),
loginMap.get("password"));
        if(admin!=null){
            return new Result(true,StatusCode.OK,"登陆成功");
        }else{
            return new Result(false,StatusCode.LOGINERROR,"用户名或密码错
误");
        }
    }
```

1.3 用户密码加密

1.3.1 用户注册密码加密

（4）修改tensquare_user工程的UserService 类，引入BCryptPasswordEncoder

```
    @Autowired
    BCryptPasswordEncoder encoder;
```

（5）修改tensquare_user工程的UserService 类的add方法，添加密码加密的逻辑

```
/**
     * 增加
     * @param user
     * @param code
     */
    public void add(User user,String code) {
        ........
        ........
        ........
        //密码加密
        String newpassword = encoder.encode(user.getPassword());//加密后的
密码
        user.setPassword(newpassword);
        userDao.save(user);
    }
```

（4）测试运行后，添加数据

```
{
    "mobile": "13901238899"
    "password": "123123",
}
```

数据库中的密码为以下形式

```
$2a$10$a/EYRjdKwQ6zjr0/HJ6RR.rcA1dwv1ys7Uso1xShUaBWlIWTyJl5S
```

1.3.2 用户登陆密码判断

（1）修改tensquare_user工程的UserDao接口，增加方法定义

```
    /**
     * 根据手机号查询用户
     * @param mobile
     * @return
     */
    public User findByMobile(String mobile);
```

（2）修改tensquare_user工程的UserService 类，增加方法

```
    /**
     * 根据手机号和密码查询用户
     * @param mobile
     * @param password
     * @return
     */
    public User findByMobileAndPassword(String mobile,String password){
        User user = userDao.findByMobile(mobile);
        if(user!=null &&  encoder.matches(password,user.getPassword())){
            return user;
        }else{
            return null;
        }
    }
```

（4）修改tensquare_user工程的UserController类，增加login方法

```
    /**
     * 用户登陆
     * @param mobile
     * @param password
     * @return
     */
    @RequestMapping(value="/login",method=RequestMethod.POST)
    public Result login(String mobile,String password){
        User user = userService.findByMobileAndPassword(mobile,password);
        if(user!=null){
            return new Result(true,StatusCode.OK,"登陆成功");
        }else{
            return new Result(false,StatusCode.LOGINERROR,"用户名或密码错
误");
        }
    }
```

（4）使用刚才新增加的账号进行测试，查看返回结果

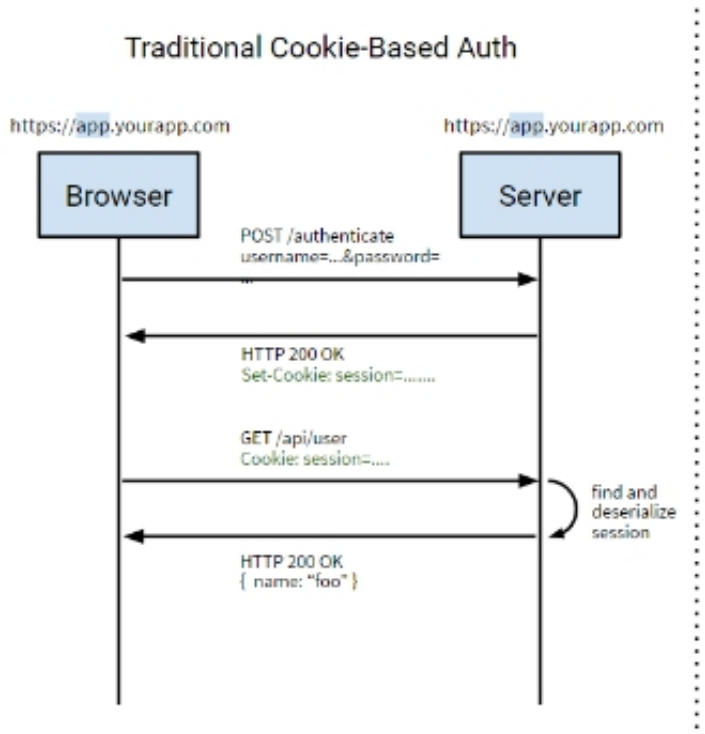
2 常见的认证机制

2.1 HTTP Basic Auth

HTTP Basic Auth简单点说明就是每次请求API时都提供用户的username和password，简言之，Basic Auth是配合RESTful API 使用的最简单的认证方式，只需提供用户名密码即可，但由于有把用户名密码暴露给第三方客户端的风险，在生产环境下被使用的越来越少。因此，在开发对外开放的RESTful API时，尽量避免采用HTTP Basic Auth

2.2 Cookie Auth

Cookie认证机制就是为一次请求认证在服务端创建一个Session对象，同时在客户端的浏览器端创建了一个Cookie对象；通过客户端带上来Cookie对象来与服务器端的session对象匹配来实现状态管理的。默认的，当我们关闭浏览器的时候，cookie会被删除。但可以通过修改cookie 的expire time使cookie在一定时间内有效；



2.3 OAuth

OAuth（开放授权）是一个开放的授权标准，允许用户让第三方应用访问该用户在某一web服务上存储的私密的资源（如照片，视频，联系人列表），而无需将用户名和密码提供给第三方应用。

OAuth允许用户提供一个令牌，而不是用户名和密码来访问他们存放在特定服务提供者的数据。每一个令牌授权一个特定的第三方系统（例如，视频编辑网站)在特定的时段（例如，接下来的2小时内）内访问特定的资源（例如仅仅是某一相册中的视频）。这样，OAuth让用户可以授权第三方网站访问他们存储在另外服务提供者的某些特定信息，而非所有内容

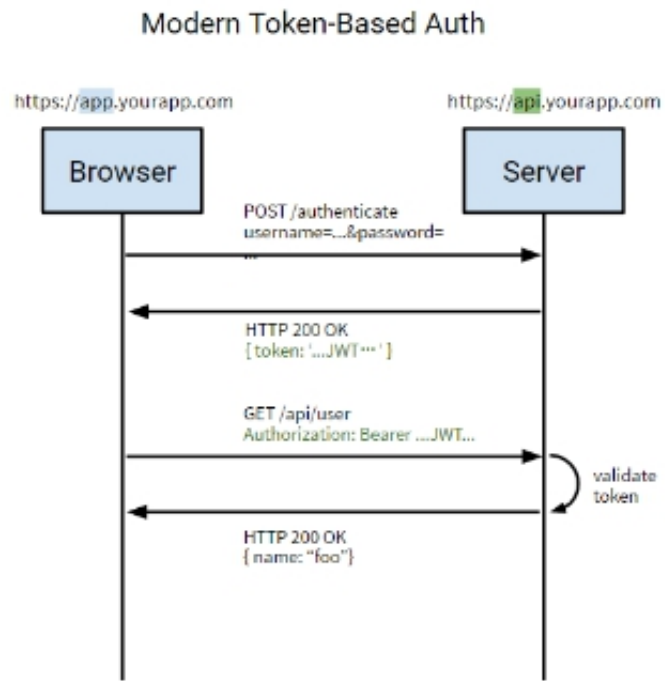
下面是OAuth2.0的流程：

这种基于OAuth的认证机制适用于个人消费者类的互联网产品，如社交类APP等应用，但是不太适合拥有自有认证权限管理的企业应用。

2.4 Token Auth

使用基于 Token 的身份验证方法，在服务端不需要存储用户的登录记录。大概的流程是这样的：

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，再把这个 Token 发送给客户端
4. 客户端收到 Token 以后可以把它存储起来，比如放在 Cookie 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，就向客户端返回请求的数据



Token Auth的优点

Token机制相对于Cookie机制又有什么好处呢？

- 支持跨域访问: Cookie是不允许垮域访问的，这一点对Token机制是不存在的，前提是传输的用户认证信息通过HTTP头传输.
- 无状态(也称：服务端可扩展行):Token机制在服务端不需要存储session信息，因为Token 自身包含了所有登录用户的信息，只需要在客户端的cookie或本地介质存储状态信息.
- 更适用**CDN**: 可以通过内容分发网络请求你服务端的所有资料（如：javascript，HTML,图片等），而你的服务端只要提供API即可.
- 去耦: 不需要绑定到一个特定的身份验证方案。Token可以在任何地方生成，只要在你的API被调用的时候，你可以进行Token生成调用即可.
- 更适用于移动应用: 当你的客户端是一个原生平台（iOS, Android，Windows 8等）时，Cookie是不被支持的（你需要通过Cookie容器进行处理），这时采用Token认证机制就会简单得多。
- **CSRF**:因为不再依赖于Cookie，所以你就不需要考虑对CSRF（跨站请求伪造）的防范。
- 性能: 一次网络往返时间（通过数据库查询session信息）总比做一次HMACSHA256计算 的Token验证和解析要费时得多.
- 不需要为登录页面做特殊处理: 如果你使用Protractor 做功能测试的时候，不再需要为登录页面做特殊处理.

- 基于标准化:你的API可以采用标准化的 JSON Web Token (JWT). 这个标准已经存在多个后端库（.NET, Ruby, Java,Python, PHP）和多家公司的支持（如：Firebase,Google, Microsoft）.

3 基于JWT的Token认证机制实现

3.1 什么是JWT

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。

3.2 JWT组成

一个JWT实际上就是一个字符串，它由三部分组成，头部、载荷与签名。

头部（**Header**）

头部用于描述关于该JWT的最基本的信息，例如其类型以及签名所用的算法等。这也可以被表示成一个JSON对象。

```
{"typ":"JWT","alg":"HS256"}
```

在头部指明了签名算法是HS256算法。 我们进行BASE64编码http://base64.xpcha.com/，编码后的字符串如下：

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
```

> 小知识：Base64是一种基于64个可打印字符来表示二进制数据的表示方法。由于2的6次方等于64，所以每6个比特为一个单元，对应某个可打印字符。三个字节有24个比特，对应于4个Base64单元，即3个字节需要用4个可打印字符来表示。JDK 中提供了非常方便的 **BASE64Encoder** 和 **BASE64Decoder**，用它们可以非常方便的完成基于 BASE64 的编码和解码

载荷（**playload**）

载荷就是存放有效信息的地方。这个名字像是特指飞机上承载的货品，这些有效信息包含三个部分

（1）标准中注册的声明（建议但不强制使用）

```
iss: jwt签发者
sub: jwt所面向的用户
aud: 接收jwt的一方
exp: jwt的过期时间，这个过期时间必须要大于签发时间
nbf: 定义在什么时间之前，该jwt都是不可用的.
iat: jwt的签发时间
jti: jwt的唯一身份标识，主要用来作为一次性token,从而回避重放攻击。
```

（2）公共的声明

公共的声明可以添加任何的信息，一般添加用户的相关信息或其他业务需要的必要信息.但不建议添加敏感信息，因为该部分在客户端可解密.

（3）私有的声明

私有声明是提供者和消费者所共同定义的声明，一般不建议存放敏感信息，因为base64是对称解密的，意味着该部分信息可以归类为明文信息。

这个指的就是自定义的claim。比如前面那个结构举例中的admin和name都属于自定的claim。这些claim跟JWT标准规定的claim区别在于：JWT规定的claim，JWT的接收方在拿到JWT之后，都知道怎么对这些标准的claim进行验证(还不知道是否能够验证)；而private claims不会验证，除非明确告诉接收方要对这些claim进行验证以及规则才行。

定义一个payload:

```
{"sub":"1234567890","name":"John Doe","admin":true}
```

然后将其进行base64编码，得到Jwt的第二部分。

```
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9
```

签证（**signature**）

jwt的第三部分是一个签证信息，这个签证信息由三部分组成：

> header (base64后的)
>
> payload (base64后的)
>
> secret

这个部分需要base64加密后的header和base64加密后的payload使用.连接组成的字符串，然后通过header中声明的加密方式进行加盐secret组合加密，然后就构成了jwt的第三部分。

```
TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

将这三部分用.连接成一个完整的字符串,构成了最终的jwt:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

注意：**secret**是保存在服务器端的，**jwt**的签发生成也是在服务器端的，**secret**就是用来进行**jwt**的签发和**jwt**的验证，所以，它就是你服务端的私钥，在任何场景都不应该流露出去。一旦客户端得知这个**secret,** 那就意味着客户端是可以自我签发**jwt**了。

4 Java的JJWT实现JWT

4.1 什么是JJWT

JJWT是一个提供端到端的JWT创建和验证的Java库。永远免费和开源(Apache License，版本2.0)，JJWT很容易使用和理解。它被设计成一个以建筑为中心的流畅界面，隐藏了它的大部分复杂性。

4.2 JJWT快速入门

4.2.1 token的创建

（1）创建maven工程，引入依赖

```
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.6.0</version>
        </dependency>
```

（2）创建类CreateJwtTest，用于生成token

```
public class CreateJwtTest {

    public static void main(String[] args) {

        JwtBuilder builder= Jwts.builder().setId("888")
                .setSubject("小白")
                .setIssuedAt(new Date())
                .signWith(SignatureAlgorithm.HS256,"itcast");
        System.out.println( builder.compact() );

    }
}
```

setIssuedAt用于设置签发时间

signWith用于设置签名秘钥

（3）测试运行，输出如下:

```
eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiLlsI_nmb0iLCJpYXQiOjE1MjM0M
TM0NTh9.gq0J-cOM_qCNqU_s-d_IrRytaNenesPmqAIhQpYXHZk
```

再次运行，会发现每次运行的结果是不一样的，因为我们的载荷中包含了时间。

4.2.2 token的解析

我们刚才已经创建了token，在web应用中这个操作是由服务端进行然后发给客户端，客户端在下次向服务端发送请求时需要携带这个token（这就好像是拿着一张门票一样），那服务端接到这个token 应该解析出token中的信息（例如用户id）,根据这些信息查询数据库返回相应的结果。

创建ParseJwtTest

```
public class ParseJwtTest {

    public static void main(String[] args) {
        String
token="eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiLlsI_nmb0iLCJpYXQiO
jE1MjM0MTM0NTh9.gq0J-cOM_qCNqU_s-d_IrRytaNenesPmqAIhQpYXHZk";
        Claims claims =
Jwts.parser().setSigningKey("itcast").parseClaimsJws(token).getBody();
        System.out.println("id:"+claims.getId());
        System.out.println("subject:"+claims.getSubject());
        System.out.println("IssuedAt:"+claims.getIssuedAt());
    }
}
```

试着将token或签名秘钥篡改一下，会发现运行时就会报错，所以解析token也就是验证token

4.2.3 token过期校验

有很多时候，我们并不希望签发的token是永久生效的，所以我们可以为token添加一个过期时间。

创建CreateJwtTest2

```
public class CreateJwtTest2 {

    public static void main(String[] args) {

        //为了方便测试，我们将过期时间设置为1分钟
        long now = System.currentTimeMillis();//当前时间
        long exp = now + 1000*60;//过期时间为1分钟
        JwtBuilder builder= Jwts.builder().setId("888")
                .setSubject("小白")
                .setIssuedAt(new Date())
                .signWith(SignatureAlgorithm.HS256,"itcast")
                .setExpiration(new Date(exp));
        System.out.println( builder.compact() );
    }
}
```

setExpiration 方法用于设置过期时间

修改ParseJwtTest

```
public class ParseJwtTest {

    public static void main(String[] args) {
        String
compactJws="eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiLlsI_nmb0iLCJp
YXQiOjE1MjM0MTY1NjksImV4cCI6MTUyMzQxNjYyOX0.Tk91b6mvyjpKcldkic8DgXz0zsPFF
nRgTgkgcAsa9cc";
        Claims claims =
Jwts.parser().setSigningKey("itcast").parseClaimsJws(compactJws).getBody(
);
        System.out.println("id:"+claims.getId());
        System.out.println("subject:"+claims.getSubject());
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");

        System.out.println("签发时间:"+sdf.format(claims.getIssuedAt()));
        System.out.println("过期时
间:"+sdf.format(claims.getExpiration()));
        System.out.println("当前时间:"+sdf.format(new Date()) );

    }
}
```

测试运行，当未过期时可以正常读取，当过期时会引发io.jsonwebtoken.ExpiredJwtException异常。

```
Exception in thread "main" io.jsonwebtoken.ExpiredJwtException: JWT
expired at 2018-06-08T21:44:55+0800. Current time: 2018-06-
08T21:44:56+0800
        at
io.jsonwebtoken.impl.DefaultJwtParser.parse(DefaultJwtParser.java:365)
        at
io.jsonwebtoken.impl.DefaultJwtParser.parse(DefaultJwtParser.java:458)
        at
io.jsonwebtoken.impl.DefaultJwtParser.parseClaimsJws(DefaultJwtParser.java
:518)
        at cn.itcast.demo.ParseJwtTest.main(ParseJwtTest.java:13)
```

4.2.4 自定义claims

我们刚才的例子只是存储了id和subject两个信息，如果你想存储更多的信息（例如角色）可以定义自定义claims 创建CreateJwtTest3

```
public class CreateJwtTest3 {

    public static void main(String[] args) {

        //为了方便测试，我们将过期时间设置为1分钟
        long now = System.currentTimeMillis();//当前时间
        long exp = now + 1000*60;//过期时间为1分钟
        JwtBuilder builder= Jwts.builder().setId("888")
                .setSubject("小白")
                .setIssuedAt(new Date())
                .signWith(SignatureAlgorithm.HS256,"itcast")
                .setExpiration(new Date(exp))
                .claim("roles","admin")
                .claim("logo","logo.png");
        System.out.println( builder.compact() );
    }
}
```

修改ParseJwtTest

```
public class ParseJwtTest {

    public static void main(String[] args) {
        String
compactJws="eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiLlsI_nmb0iLCJp
YXQiOjE1MjM0MTczMjMsImV4cCI6MTUyMzQxNzM4Mywicm9sZXMiOiJhZG1pbiIsImxvZ28iO
iJsb2dvLnBuZyJ9.b11p4g4rE94rqFhcfzdJTPCORikqP_1zJ1MP8KihYTQ";
        Claims claims =
Jwts.parser().setSigningKey("itcast").parseClaimsJws(compactJws).getBody(
);
        System.out.println("id:"+claims.getId());
        System.out.println("subject:"+claims.getSubject());
        System.out.println("roles:"+claims.get("roles"));
        System.out.println("logo:"+claims.get("logo"));

        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");

        System.out.println("签发时间:"+sdf.format(claims.getIssuedAt()));
        System.out.println("过期时
间:"+sdf.format(claims.getExpiration()));
        System.out.println("当前时间:"+sdf.format(new Date()) );

    }
}
```

5 十次方微服务鉴权

5.1 JWT工具类编写

（1）tensquare_common工程引入依赖（考虑到工具类的通用性）

```
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.6.0</version>
        </dependency>
```

（2）修改tensquare_common工程，创建util.JwtUtil

```
@ConfigurationProperties("jwt.config")
public class JwtUtil {

    private String key ;

    private long ttl ;//一个小时

    public String getKey() {
        return key;
    }

    public void setKey(String key) {
        this.key = key;
    }

    public long getTtl() {
        return ttl;
    }

    public void setTtl(long ttl) {
        this.ttl = ttl;
    }

    /**
     * 生成JWT
     *
     * @param id
     * @param subject
     * @return
     */
    public String createJWT(String id, String subject, String roles) {
        long nowMillis = System.currentTimeMillis();
        Date now = new Date(nowMillis);
        JwtBuilder builder = Jwts.builder().setId(id)
                .setSubject(subject)
                .setIssuedAt(now)
                .signWith(SignatureAlgorithm.HS256, key).claim("roles",
roles);
        if (ttl > 0) {

            builder.setExpiration( new Date( nowMillis + ttl));
```

```
        }
        return builder.compact();
    }

    /**
     * 解析JWT
     * @param jwtStr
     * @return
     */
    public Claims parseJWT(String jwtStr){
        return  Jwts.parser()
                .setSigningKey(key)
                .parseClaimsJws(jwtStr)
                .getBody();
    }

}
```

（3）修改tensquare_user工程的application.yml，添加配置

```
jwt:
 config:
    key: itcast
    ttl: 360000
```

5.2 管理员登陆后台签发token

（1）配置bean .修改tensquare_user工程Application类

```
    @Bean
    public JwtUtil jwtUtil(){
        return new util.JwtUtil();
    }
```

（2）修改AdminController的login方法

```
    @Autowired
    private JwtUtil jwtUtil;

    /**
     * 用户登陆
     * @param loginname
     * @param password
     * @return
     */
    @RequestMapping(value="/login",method=RequestMethod.POST)
    public Result login(@RequestBody Map<String,String> loginMap){
        Admin admin =
adminService.findByLoginnameAndPassword(loginMap.get("loginname"),
loginMap.get("password"));
        if(admin!=null){
            //生成token
            String token = jwtUtil.createJWT(admin.getId(),
admin.getLoginname(), "admin");
            Map map=new HashMap();
            map.put("token",token);
            map.put("name",admin.getLoginname());//登陆名
            return new Result(true,StatusCode.OK,"登陆成功",map);
        }else{
            return new Result(false,StatusCode.LOGINERROR,"用户名或密码错
误");
        }
    }
```

测试运行结果

```
{
  "flag": true,
  "code": 20000,
  "message": "登陆成功",
  "data": {
    "token":
"eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI5ODQzMjc1MDc4ODI5MzgzNjgiLCJzdWIiOiJ4aWF
vbWkiLCJpYXQiOjE1MjM1MjQxNTksInJvbGVzIjoiYWRtaW4iLCJleHAiOjE1MjM1MjQ1MTl9
._YF3oftRNTbq9WCD8Jg1tqcez3cSWoQiDIxMuPmp73o",
    "name":"admin"
  }
}
```


5.3 删除用户功能鉴权

需求：删除用户，必须拥有管理员权限，否则不能删除。

前后端约定：前端请求微服务时需要添加头信息Authorization ,内容为Bearer+空格+token

（1）修改UserController的findAll方法 ，判断请求中的头信息，提取token并验证权限。

```
    @Autowired
    private HttpServletRequest request;

    /**
     * 删除
     * @param id
     */
    @RequestMapping(value="/{id}",method= RequestMethod.DELETE)
    public Result delete(@PathVariable String id ){

        String authHeader = request.getHeader("Authorization");//获取头信
息
        if(authHeader==null){
            return new Result(false,StatusCode.ACCESSERROR,"权限不足");
        }
        if(!authHeader.startsWith("Bearer ")){
            return new Result(false,StatusCode.ACCESSERROR,"权限不足");
        }
        String token=authHeader.substring(7);//提取token
        Claims claims = jwtUtil.parseJWT(token);
        if(claims==null){
            return new Result(false,StatusCode.ACCESSERROR,"权限不足");
        }
        if(!"admin".equals(claims.get("roles"))){
            return new Result(false,StatusCode.ACCESSERROR,"权限不足");
        }

        userService.deleteById(id);
        return new Result(true,StatusCode.OK,"删除成功");
    }
```

5.4 使用拦截器方式实现token鉴权

如果我们每个方法都去写一段代码，冗余度太高，不利于维护，那如何做使我们的代码看起来更清爽呢？我们可以将这段代码放入拦截器去实现

5.4.1 添加拦截器

Spring为我们提供了org.springframework.web.servlet.handler.HandlerInterceptorAdapter这个适配器，继承此类，可以非常方便的实现自己的拦截器。他有三个方法：

分别实现预处理、后处理（调用了Service并返回ModelAndView，但未进行页面渲染）、返回处理（已经渲染了页面）
在preHandle中，可以进行编码、安全控制等处理；
在postHandle中，有机会修改ModelAndView；
在afterCompletion中，可以根据ex是否为null判断是否发生了异常，进行日志记录。

（1）创建拦截器类。创建 com.tensquare.user.filter.JwtFilter

```
@Component
public class JwtFilter extends HandlerInterceptorAdapter {

    @Autowired
    private JwtUtil jwtUtil;

    @Override
    public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler)
            throws Exception {
        System.out.println("经过了拦截器");
        return true;
    }
}
```

（2）配置拦截器类,创建com.tensquare.user.ApplicationConfig

```
@Configuration
public class ApplicationConfig extends WebMvcConfigurationSupport {

    @Autowired
    private JwtFilter jwtFilter;

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(jwtFilter).
                addPathPatterns("/**").
                excludePathPatterns("/**/login");
    }
}
```

5.4.2 拦截器验证token

（1）修改拦截器类 JwtFilter

```
@Component
public class JwtFilter extends HandlerInterceptorAdapter {

    @Autowired
    private JwtUtil jwtUtil;

    @Override
    public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler)  throws Exception {
        System.out.println("经过了拦截器");
        final String authHeader = request.getHeader("Authorization");
        if (authHeader != null &&  authHeader.startsWith("Bearer ")) {
            final String token = authHeader.substring(7); // The part
after "Bearer "
            Claims claims = jwtUtil.parseJWT(token);
            if (claims != null) {
                if("admin".equals(claims.get("roles"))){//如果是管理员
                    request.setAttribute("admin_claims", claims);
                }
                if("user".equals(claims.get("roles"))){//如果是用户
                    request.setAttribute("user_claims", claims);
                }
            }
        }
        return true;
    }
}
```

（2）修改UserController的delete方法

```
    /**
     * 删除
     * @param id
     */
    @RequestMapping(value="/{id}",method= RequestMethod.DELETE)
    public Result delete(@PathVariable String id ){
        Claims claims=(Claims) request.getAttribute("admin_claims");
        if(claims==null){
            return new Result(true,StatusCode.ACCESSRROR,"无权访问");
        }
        userService.deleteById(id);
        return new Result(true,StatusCode.OK,"删除成功");
    }
```

6 发布信息验证Token

6.1 用户登陆签发 JWT

（2）修改UserController，引入JwtUtil 修改login方法 ，返回token，昵称，头像等信息

```
    @Autowired
    private JwtUtil jwtUtil;

    /**
     * 用户登陆
     * @param mobile
     * @param password
     * @return
     */
    @RequestMapping(value="/login",method=RequestMethod.POST)
    public Result login(@RequestBody Map<String,String> loginMap){
        User user =
userService.findByMobileAndPassword(loginMap.get("mobile"),loginMap.get("
password"));
        if(user!=null){
            String token = jwtUtil.createJWT(user.getId(),
user.getNickname(), "user");
            Map map=new HashMap();
            map.put("token",token);
            map.put("name",user.getNickname());//昵称
            map.put("avatar",user.getAvatar());//头像
            return new Result(true,StatusCode.OK,"登陆成功",map);
        }else{
            return new Result(false,StatusCode.LOGINERROR,"用户名或密码错
误");
        }
    }
```

（4）测试运行 http://localhost:9008/user/login (POST)，结果为如下形式

```
{
  "flag": true,
  "code": 20000,
  "message": "登陆成功",
  "data": {
    "token":
"eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI5ODM5ODc0ODk1MDcyNTAxNzYiLCJpYXQiOjE1MjM
1MDIzMDEsInJvbGVzIjoidXNlciIsImV4cCI6MTUyMzUwMjY2MX0.QH-
WMRgIAxHDcYReH7patg7qkcjc4ZuyDbHaIah-spQ"
  },
  "name":"小雅",
  "avatar":"......."
}
```


6.2 发布问题

（1）修改tensquare_qa工程的QaApplication，增加bean

```
    @Bean
    public JwtUtil jwtUtil(){
        return new util.JwtUtil();
    }
```

（2）tensquare_qa工程配置文件application.yml增加配置：

```
jwt:
  config:
    key: itcast
```

（3）增加拦截器类 （参考上面的拦截器代码）

（4）增加配置类ApplicationConfig （参考上面的配置类代码）

（5）修改ProblemController的add方法

```
    @Autowired
    private HttpServletRequest request;

    /**
     * 发布问题
     * @param problem
     */
    @RequestMapping(method=RequestMethod.POST)
    public Result add(@RequestBody Problem problem  ){
        Claims claims=(Claims)request.getAttribute("user_claims");
        if(claims==null){
            return new Result(false,StatusCode.ACCESSERROR,"无权访问");
        }
        problem.setUserid(claims.getId());
        problemService.add(problem);
        return new Result(true,StatusCode.OK,"增加成功");
    }
```

6.3 回答问题

学员实现，步骤略

6.4 发吐槽

学员实现，步骤略

6.5 发文章

学员实现，步骤略