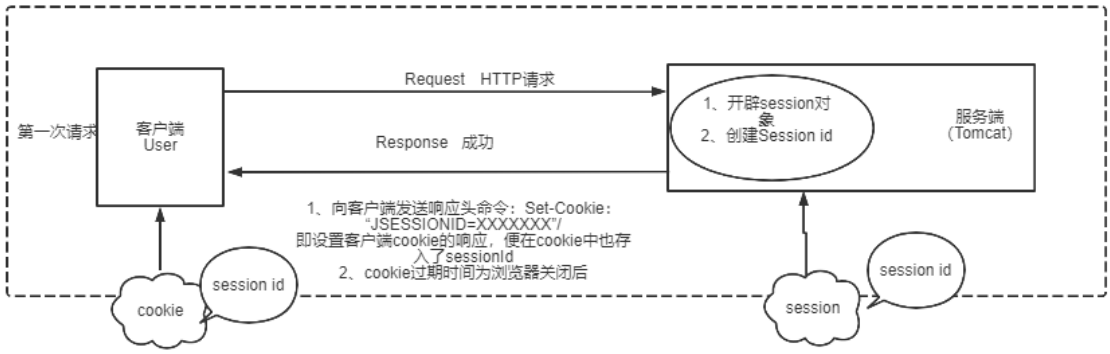
03-基于Spring Security + JWT实现第二版前后端分离架构实现

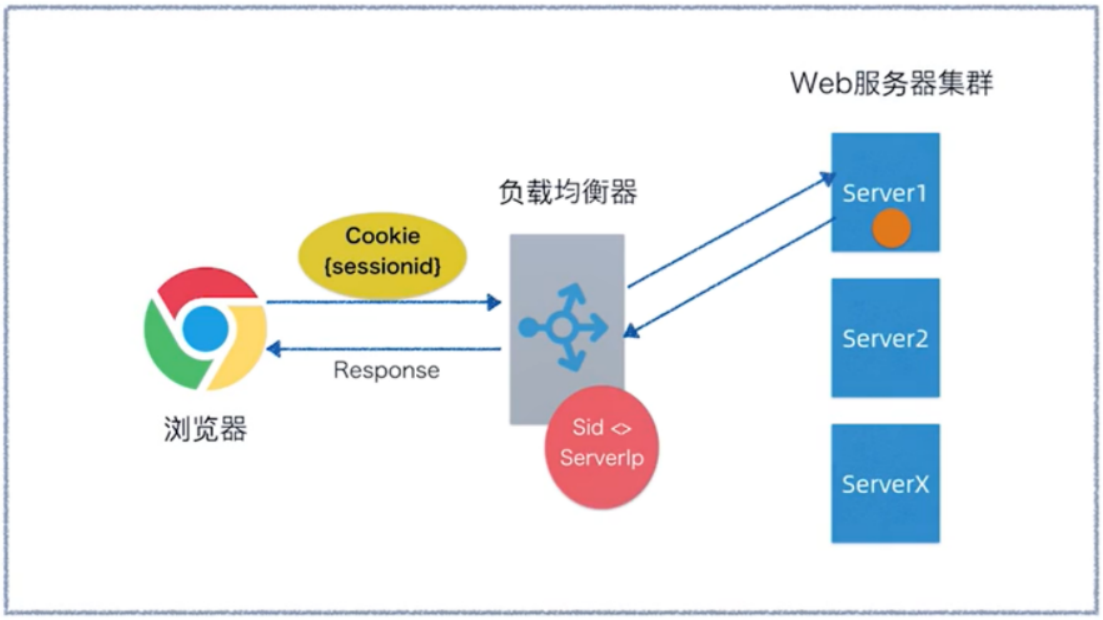
任务案例分析

在我们传统的B\S应用开发方式中，都是使用session进行状态管理的，比如说：保存登录、用户、权限等状态信息。这种方式的原理大致如下：



上面就是一种有状态服务。

当然，这整个过程中，cookies和sessionid都是服务端和浏览器端自动维护的。所以从编码层面是感知不到的，程序员只能感知到session数据的存取。但是，这种方式在有些情况下，是不适用的。比如：非浏览器的客户端、手机移动端等等，因为他们没有浏览器自动维护cookies的功能。比如：集群应用，同一个应用部署甲、乙、丙三个主机上，实现负载均衡应用，其中一个挂掉了其他的还能负载工作。要知道session是保存在服务器内存里面的，三个主机一定是不同的内存。那么你登录的时候访问甲，而获取接口数据的时候访问乙，就无法保证session的唯一性和共享性。当然以上的这些情况我们都有方案(如redis共享session等)，可以继续使用session来保存状态。但是还有另外一种做法就是不用session了，即开发一个无状态的应用，JWT就是这样的一种方案。



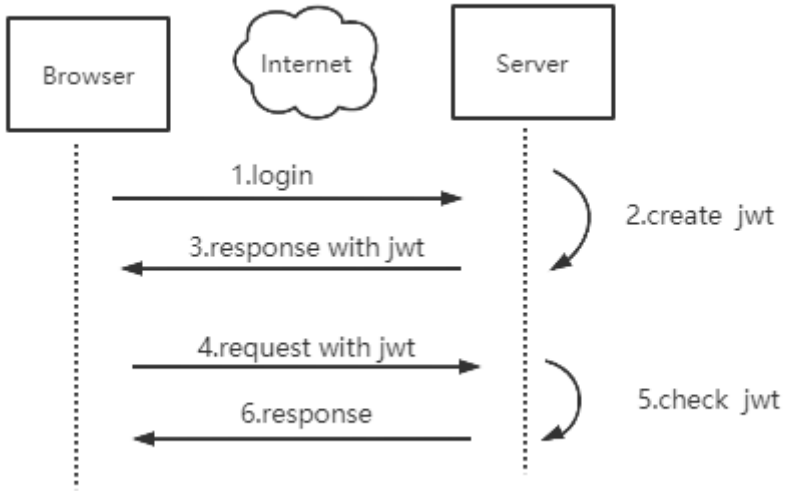
那上面说的无状态是什么?

微服务集群中的每个服务，对外提供的都使用RESTful风格的接口。而RESTful风格的一个最重要的规范就是：服务的无状态性，即：服务端不保存任何客户端请求者信息，客户端的每次请求必须具备自描述信息，通过这些信息识别客户端身份那么这种无状态性有哪些好处呢?
客户端请求不依赖服务端的信息，多次请求不需要必须访问到同一台服务器服务端的集群和状态对客户端透明服务端可以任意的迁移和伸缩（可以方便的进行集群化部署）减小服务端存储压力。

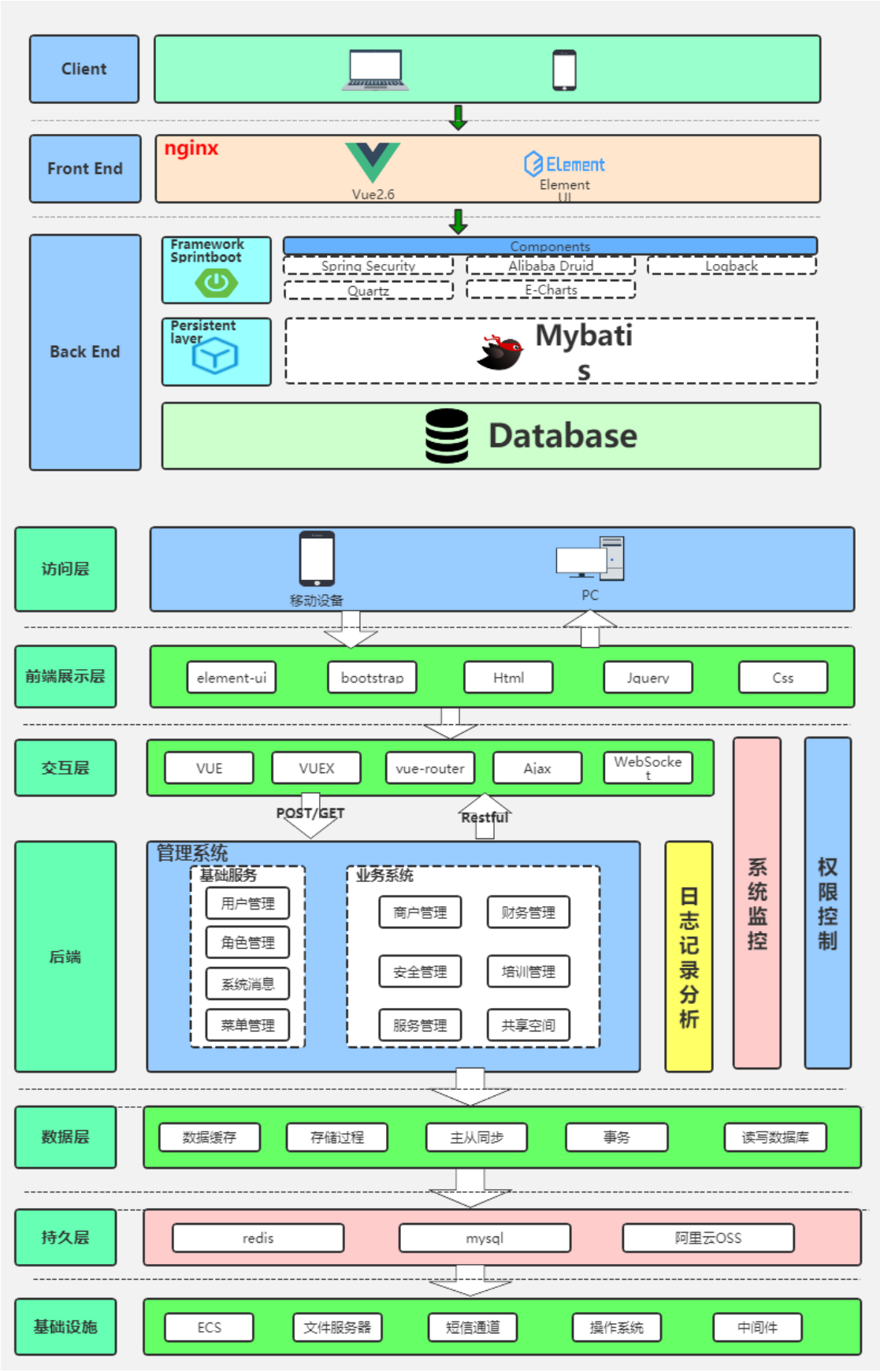
如何实现无状态无状态登录的流程:

首先客户端发送账户名/密码到服务端进行认证认证通过后，服务端将用户信息加密并且编码成一个token，返回给客户端以后客户端每次发送请求，都需要携带认证的token服务端

对客户端发送来的token进行解密，判断是否有效，并且获取用户登录信息



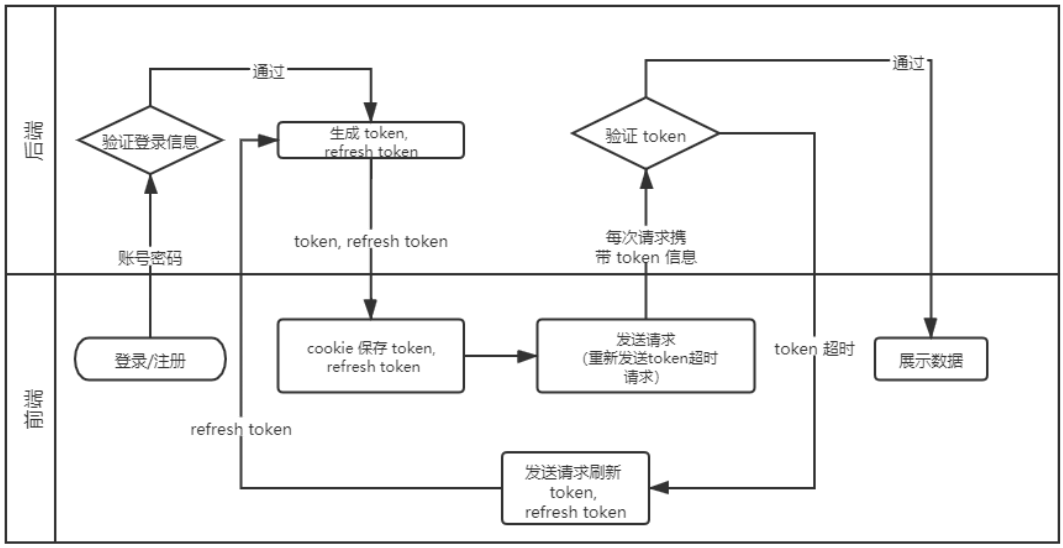
在前后端分离的项目中，登录策略也有不少，不过 JWT 算是目前比较流行的一种解决方案了，本文就和大家来分享一下如何将 Spring Security 和 JWT 结合在一起使用，进而实现前后端分离时的登录解决方案。

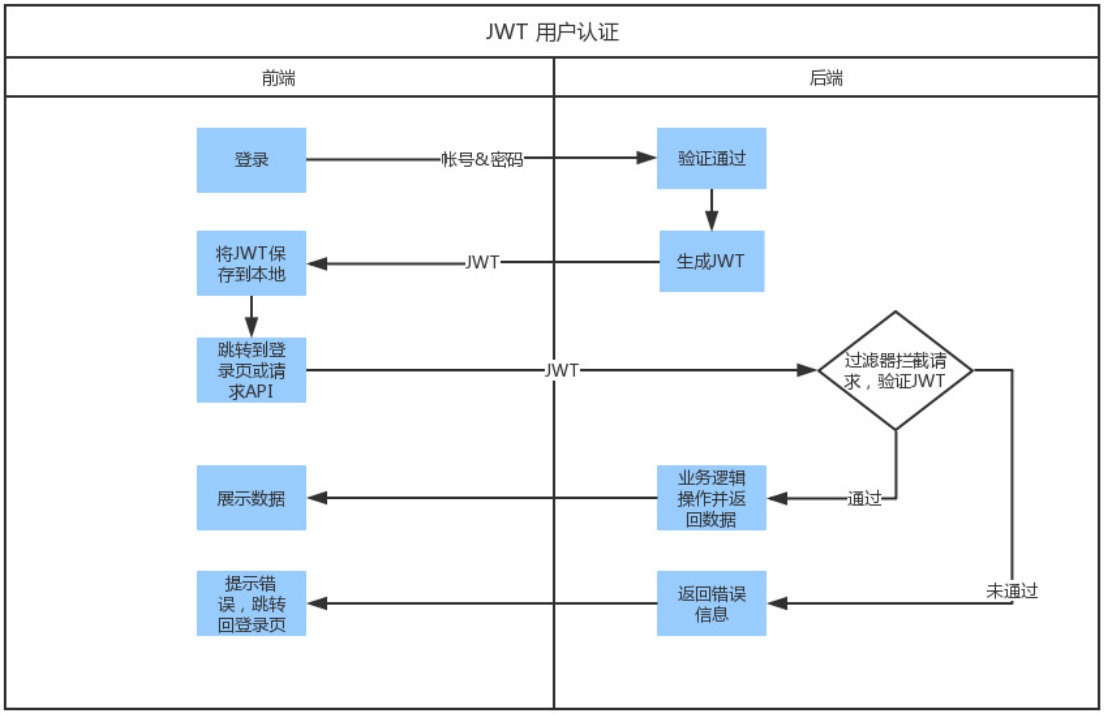

Client
Front End
nginx
Vue2.6
ELement
Element UI
Back End
Framework Sprintboot
Components
Spring Security
Alibaba Druid
Logback
Quartz
E-Charts
Persistent layer
Mybatis
Database
访问层
移动设备
PC
前端展示层
element-ui
bootstrap
Html
Jquery
Css
交互层
VUE
VUEX
vue-router
Ajax
WebSocket
POST/GET
Restful
后端
管理系统
基础服务
用户管理
角色管理
系统消息
菜单管理
业务系统
商户管理
财务管理
安全管理
培训管理
服务管理
共享空间
日志记录分析
系统监控
权限控制
数据层
数据缓存
存储过程
主从同步
事务
读写数据库
持久层
redis
mysql
阿里云OSS
基础设施
ECS
文件服务器
短信通道
操作系统
中间件

什么是JWT?

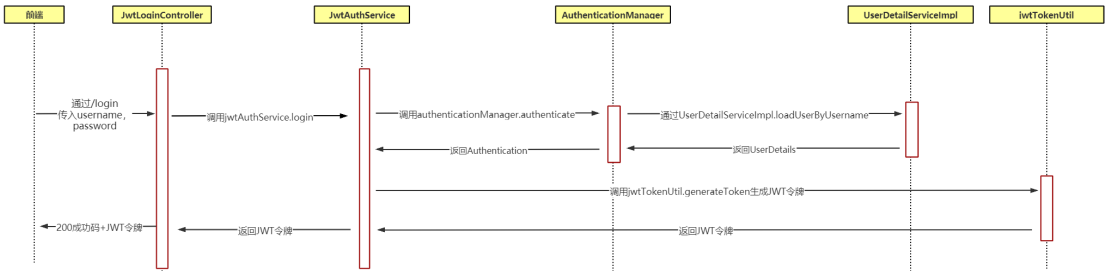
JWT，全称是Json Web Token， 是一种JSON风格的轻量级的授权和身份认证规范，可实现无状态、分布式的Web应用授权。
JWT是一个加密后的接口访问密码，并且该密码里面包含用户名信息。这样既可以知道你是谁？又可以知道你是否可以访问应用？

JWT交互流程流程图：





认证代码实现：



1.在pom.xml中引入jwt工具包

```
<!--Token生成与解析-->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.0</version>
</dependency>
```

2.在application.yml中加入如下自定义一些关于JWT的配置

```
# token配置
token:
  # 令牌自定义标识
  header: Authorization
  # 令牌秘钥
  #  secret: abcdefghijklmnopqrstuvwxyz
  secret: (EMOK:)_$^11244^%$_(IS:)_@@++--(COOL:)_++++_.sds_(GUY:)
  # 令牌有效期（默认30分钟）
  expireTime: 3600000
```

其中header是携带JWT令牌的HTTP的Header的名称。虽然我这里叫做Authorization，但是在实际生产中可读性越差越安全。secret是用来为JWT基础信息加密和解密的密钥。虽然我在这里在配置文件写死了，但是在实际生产中通常不直接写在配置文件里面。而是通过应用的启动参数传递，并且需要定期修改。expiration是JWT令牌的有效时间。

3.编写JwtLoginController类

```
@RestController
public class JwtLoginController {
```

```
    @Autowired
    JwtAuthService jwtAuthService;

    /**
     * 登录方法
     *
     * @param username 用户名
     * @param password 密码
     * @return 结果
     */
    @PostMapping({"/login", "/"})
    public RestResult login(String username, String password) {
        RestResult result = RestResult.success();
        String token = jwtAuthService.login(username, password);
        result.put("token", token);
        return result;
    }
}
```

4.编写JwtAuthService类

```
@Service
public class JwtAuthService {

    @Resource
    private AuthenticationManager authenticationManager;

    @Resource
    private jwtTokenUtil jwtTokenUtil;

    /**
     * 登录认证换取JWT令牌
     *
     * @param username 用户名
     * @param password 密码
     * @return 结果
     */
    public String login(String username, String password)  {
        // 用户验证
        Authentication authentication = null;
        try {
```

```
            // 该方法会去调用UserDetailsServiceImpl.loadUserByUsername
            authentication = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(username, password));
        } catch (Exception e) {
                throw new RuntimeException("用户名密码错误");
        }
        Users loginUser = (Users) authentication.getPrincipal();
        // 生成token
        return jwtTokenUtil.generateToken(loginUser);
    }
}
```

5.编写JWT令牌生成、刷新的工具类

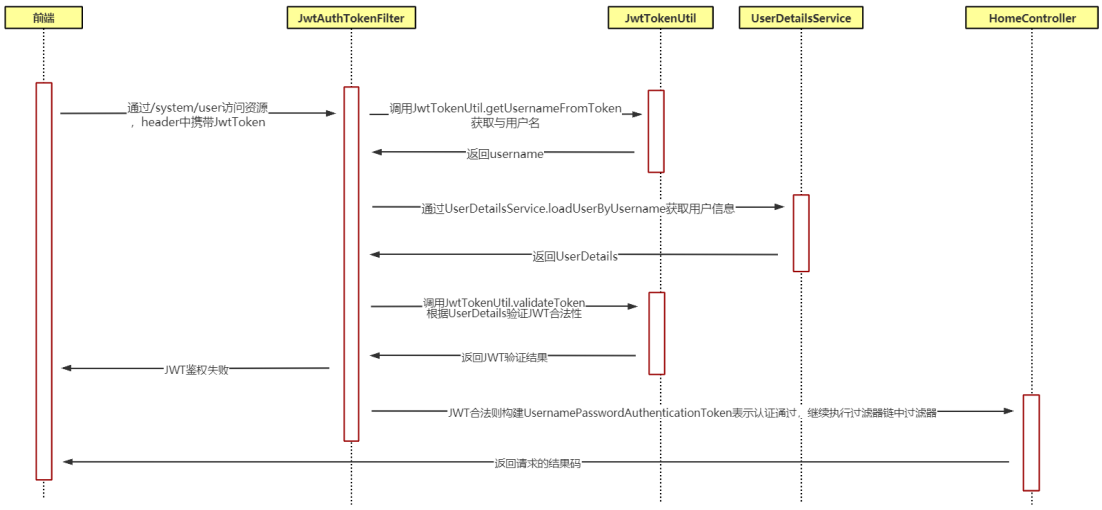
```
@Data
@Component
public class jwtTokenUtil {

    @Value("${token.secret}")
    private String secret;

    @Value("${token.expireTime}")
    private Long expiration;

    @Value("${token.header}")
    private String header;



    /**
     * 生成token令牌
     *
     * @param userDetails 用户
     * @return 令token牌
     */
    public String generateToken(UserDetails userDetails) {
        Map<String, Object> claims = new HashMap<>(2);
        claims.put("sub", userDetails.getUsername());
        claims.put("created", new Date());

        return generateToken(claims);
    }
```

```
    /**
     * 从令牌中获取用户名
     *
     * @param token 令牌
     * @return 用户名
     */
    public String getUsernameFromToken(String token) {
        String username;
        try {
            Claims claims = getClaimsFromToken(token);
            username = claims.getSubject();
        } catch (Exception e) {
            username = null;
        }
        return username;
    }

    /**
     * 判断令牌是否过期
     *
     * @param token 令牌
     * @return 是否过期
     */
    public Boolean isTokenExpired(String token) {
        try {
            Claims claims = getClaimsFromToken(token);
            Date expiration = claims.getExpiration();
            return expiration.before(new Date());
        } catch (Exception e) {
            return false;
        }
    }

    /**
     * 刷新令牌
     *
     * @param token 原令牌
     * @return 新令牌
     */
    public String refreshToken(String token) {
        String refreshedToken;
        try {
            Claims claims = getClaimsFromToken(token);
            claims.put("created", new Date());
```

```
            refreshedToken = generateToken(claims);
        } catch (Exception e) {
            refreshedToken = null;
        }
        return refreshedToken;
    }

    /**
     * 验证令牌
     *
     * @param token       令牌
     * @param userDetails 用户
     * @return 是否有效
     */
    public Boolean validateToken(String token, UserDetails userDetails) {

        String username = getUsernameFromToken(token);
        return (username.equals(userDetails.getUsername()) &&
!isTokenExpired(token));
    }



    /**
     * 从claims生成令牌,如果看不懂就看谁调用它
     *
     * @param claims 数据声明
     * @return 令牌
     */
    private String generateToken(Map<String, Object> claims) {
        Date expirationDate = new Date(System.currentTimeMillis() +
expiration);
        return Jwts.builder().setClaims(claims)
                .setExpiration(expirationDate)
                .signWith(SignatureAlgorithm.HS512, secret)
                .compact();
    }

    /**
     * 从令牌中获取数据声明,如果看不懂就看谁调用它
     *
     * @param token 令牌
     * @return 数据声明
     */
    private Claims getClaimsFromToken(String token) {
```

```
        Claims claims;
        try {
            claims =
Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();
        } catch (Exception e) {
            claims = null;
        }
        return claims;
    }
}
```

鉴权代码实现：



6.编写JwtAuthTokenFilter，完成鉴权

```
@Component
public class JwtAuthTokenFilter extends OncePerRequestFilter {
    @Resource
    private jwtTokenUtil jwtTokenUtil;

    @Resource
    UserDetailsService myUserDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) throws
ServletException, IOException {
```

```
        String jwtToken = request.getHeader(jwtTokenUtil.getHeader());
        if(!StringUtils.isEmpty(jwtToken)){
            String username = jwtTokenUtil.getUsernameFromToken(jwtToken);

            //如果可以正确的从JWT中提取用户信息，并且该用户未被授权
            if(username != null &&
SecurityContextHolder.getContext().getAuthentication() == null){
                UserDetails userDetails =
myUserDetailsService.loadUserByUsername(username);
                if(jwtTokenUtil.validateToken(jwtToken,userDetails)){
                    //给使用该JWT令牌的用户进行授权
                    UsernamePasswordAuthenticationToken authenticationToken =
new
UsernamePasswordAuthenticationToken(userDetails,null,userDetails.getAuthorities()

                    //交给spring security管理,在之后的过滤器中不会再被拦截进
行二次授权了

SecurityContextHolder.getContext().setAuthentication(authenticationToken);
                }
            }
        }
        filterChain.doFilter(request,response);
    }
}
```

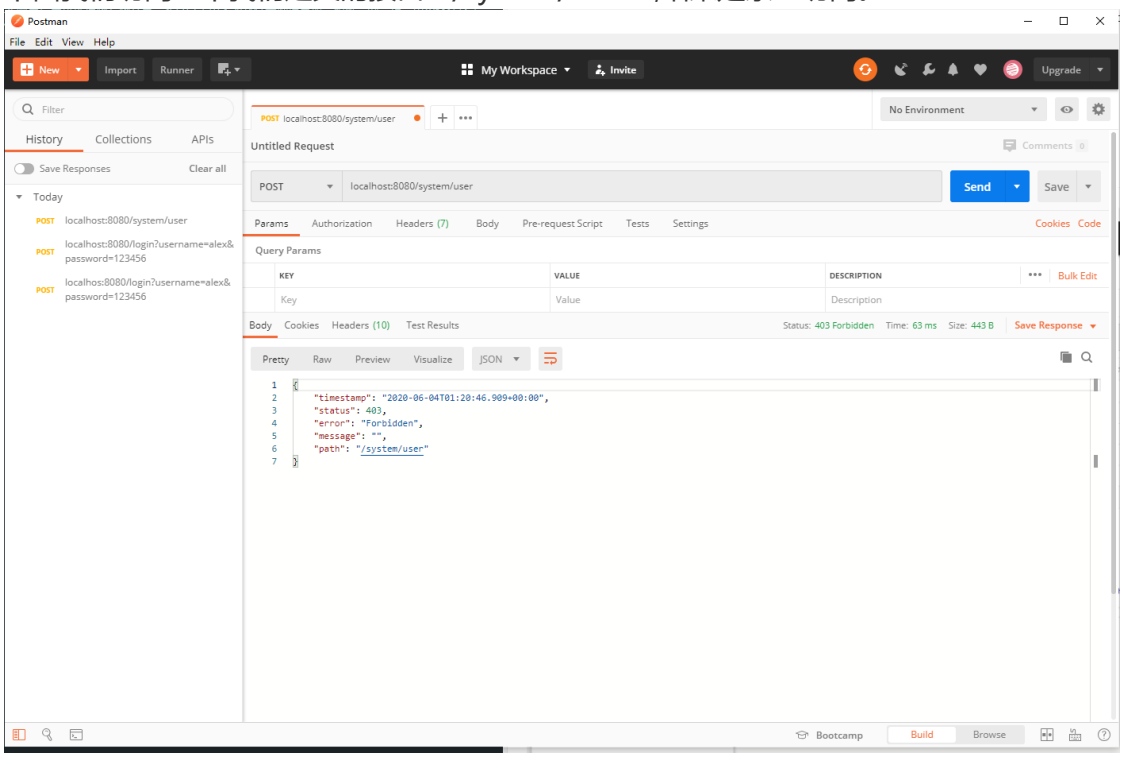
7.修改SecurityConfig.configure方法，配置无状态以及JWT filter过滤器

```
@Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity
                // 认证失败处理类

//.exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()
                // 基于token，所以不需要session

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()

                //配置权限
```

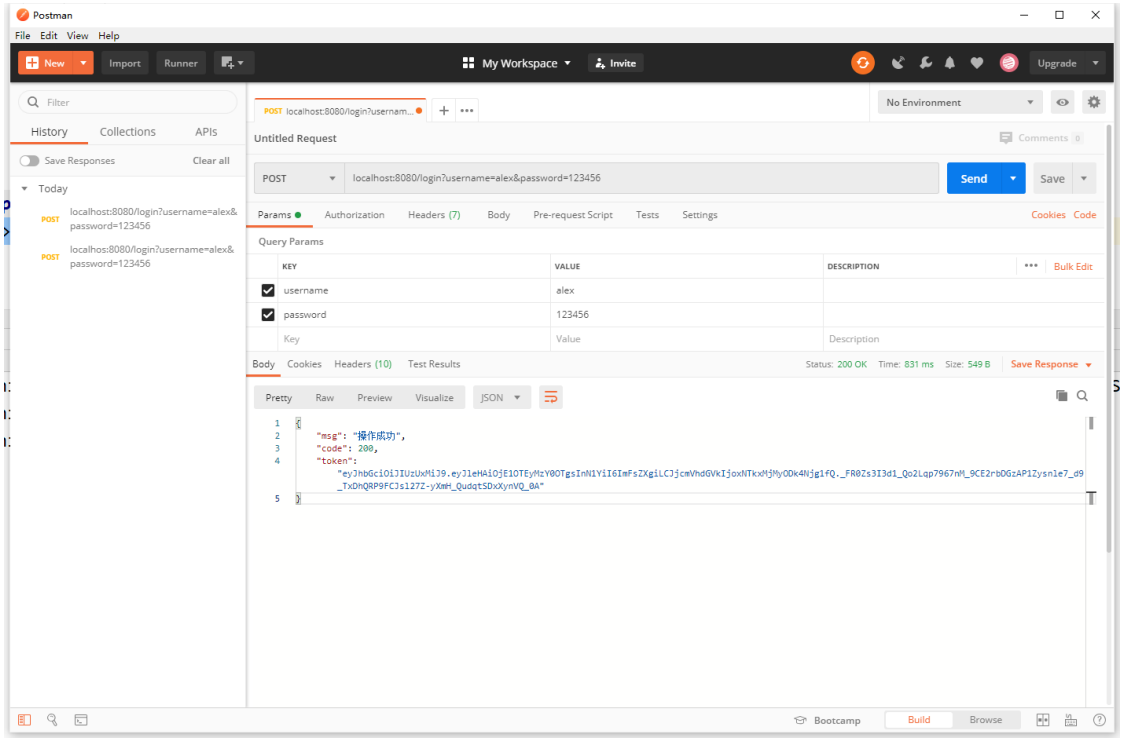
```
                .authorizeRequests()
                // 对于登录login 验证码captchaImage 允许匿名访问
                .antMatchers("/login").anonymous()
                .antMatchers(
                        HttpMethod.GET,
                        "/*.html",
                        "/**/*.html",
                        "/**/*.css",
                        "/**/*.js"
                ).permitAll()
                .antMatchers("/order") //需要对外暴露的资源路径
                .hasAnyAuthority("ROLE_USER", "ROLE_ADMIN")  //user角色和
admin角色都可以访问
                .antMatchers("/system/user", "/system/role", "/system/menu")
                .hasAnyRole("ADMIN")  //admin角色可以访问
                // 除上面外的所有请求全部需要鉴权认证
                .anyRequest().authenticated().and()//authenticated()要求在执
行该请求时，必须已经登录了应用
                // CRSF禁用，因为不使用session
                .csrf().disable() ;//禁用跨站csrf攻击防御，否则无法登陆成功
        //登出功能
        httpSecurity.logout().logoutUrl("/logout");
        // 添加JWT filter
        httpSecurity.addFilterBefore(jwtAuthTokenFilter,
UsernamePasswordAuthenticationFilter.class);
    }
```

8.使用postman进行测试

下面我们访问一个我们定义的接口“/system/user”,结果是禁止访问。



输入username：alex，password：123456，点击send按钮：



当我们将上一步返回的token，传递到header中，就能正常响应hello的接口结果。