链滴

shiro 集成 jwt

作者：wenyl

原文链接：https://ld246.com/article/1639620086212

来源网站：链滴

1、POM依赖

spring boot 版本2.5.1

shiro依赖

```
<spring-shiro.version>1.6.0</spring-shiro.version>

        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-spring</artifactId>
            <version>${spring-shiro.version}</version>
        </dependency>
```

jwt依赖

```
<jwt.version>3.3.0</jwt.version>

        <dependency>
            <groupId>com.auth0</groupId>
            <artifactId>java-jwt</artifactId>
            <version>${jwt.version}</version>
        </dependency>
```

redis依赖，使用redis存储token

```
<jedis.version>2.9.0</jedis.version>

        <dependency>
            <groupId>redis.clients</groupId>
            <artifactId>jedis</artifactId>
            <version>${jedis.version}</version>
```

```
        </dependency>
```

2、基础配置

公钥私钥可以用ssh生成，我将token存储到了redis，所以这里配置了redis

```
rsaPrivateKey=MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAIxgS3b5J9IRY
vEFIEeDeQCGkOI5pT+Nl3wNe0fdWIiw36g4sH1l2sLuTZ6bew9YRLIapude6ORGZF5UfNy9cor3
7n3ew/fXCEKVRC6Kg+cREm1rqyMjDc9NtfksXG4RGf7GNeoTmUDVStsnXvoLnzvrE7FvbB12XK
hTQDnSZAgMBAAECgYBcyET45SP5x/2/87EtymSaAP3FB5aIgiIDIwMxsKpQa/PVHZfjZWVonn4
0QYYsFUaKhe0tXmEGiLRMWQGSkTEGfZ5l7uRmrNZ0Nk9asu4/fyjwZNHYDDGAELU5R4WgLv
09PdVLG/uyIxXh9qg9y9OpYM4KoATnsH7t7TPdI5gQJBAMnB6nz18BKSsHX7qDkWWpxZUOZ
cKIsZoaDlz1NkoAKrOuH8TYc75uwLhR17nOOI9kO+10FKlYQ/5+yUiQIE+kCQQCyHcU/0mlvvBy
JsYJMDUMtDk5/BtCWD6UZan/X1GH2EHx1W5LuL+wyIr6CUrY5G3osVU5ZvLly4zTFRQHr10xA
EAqFJT4zT7uUMQXR47VoVDyzTovY+xYFtSnd6jCs3w70n4wfeEUfUKIgV2LDPHYDixx6EsLIrmqy
7VGxdAxqKIQJAe2bOyvnfXK9KeXWCjMkeZ+/RGhBFXoC+0pdg4PHMDb7RaVgCc2nLPRKj3rlj
sNUNnvt3LgrnvOYXIHk/7IKcQJAGakaB+211CbKnIHtjxSsg07EiM3dZVA1jrXTbJ6NuhURZTIOR
YJsVdWoEbwBLv/STTvc7+G+wo1yuzzBAse+w==
rsaPublicKey=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCMYEt2+SfSEWJbxBSBHg
kAhpDiOaU/jZd8DXtH3ViIsN+oOLB9ZdrC7k2em3sPWESyGqbnXujkRmReVHzcvXKK9ze593sP
1whClUQuioPnERJta6sjIw3PTbX5LFxuERn+xjXqE5lA1UrbJ176C5876xOxb2wddlyn4U0A50mQI
AQAB
# AES密码加密私钥(Base64加密)
encryptAESKey=V2FuZzkyNjQ1NGRTQkFQSUpXVA==
# JWT认证加密私钥(Base64加密)
encryptJWTKey=U0JBUElKV1RkV2FuZzkyNjQ1NA==
# AccessToken过期时间-5分钟-5*60(秒为单位)
accessTokenExpireTime=300
# RefreshToken过期时间-30分钟-30*60(秒为单位)
refreshTokenExpireTime=1800
# Shiro缓存过期时间-5分钟-5*60(秒为单位)(一般设置与AccessToken过期时间一致)
shiroCacheExpireTime=10
```

3、JWT工具类

```
import com.auth0.jwt.JWT;
import com.auth0.jwt.JWTVerifier;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.exceptions.JWTDecodeException;
import com.auth0.jwt.interfaces.DecodedJWT;
import com.kmair.ky.contract.common.entity.Constant;
import com.kmair.ky.contract.utils.security.Base64ConvertUtil;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import java.io.UnsupportedEncodingException;
import java.util.Date;

/**
 * JAVA-JWT工具类
 * @author Wang926454
```

```
 * @date 2018/8/30 11:45
 */
@Component
public class JwtUtil {

    /**
     * logger
     */
    private static final Logger logger = LoggerFactory.getLogger(JwtUtil.class);

    /**
     * 过期时间改为从配置文件获取
     */
    private static String accessTokenExpireTime;

    /**
     * JWT认证加密私钥(Base64加密)
     */
    private static String encryptJWTKey;

    @Value("${accessTokenExpireTime}")
    public void setAccessTokenExpireTime(String accessTokenExpireTime) {
        JwtUtil.accessTokenExpireTime = accessTokenExpireTime;
    }

    @Value("${encryptJWTKey}")
    public void setEncryptJWTKey(String encryptJWTKey) {
        JwtUtil.encryptJWTKey = encryptJWTKey;
    }

    /**
     * 校验token是否正确
     * @param token Token
     * @return boolean 是否正确
     * @author Wang926454
     * @date 2018/8/31 9:05
     */
    public static boolean verify(String token) {
        try {
            // 帐号加JWT私钥解密
            String secret = getClaim(token, Constant.ACCOUNT) + Base64ConvertUtil.decode(enc
yptJWTKey);
            Algorithm algorithm = Algorithm.HMAC256(secret);
            JWTVerifier verifier = JWT.require(algorithm).build();
            verifier.verify(token);
            return true;
        } catch (UnsupportedEncodingException e) {
            logger.error("JWTToken认证解密出现UnsupportedEncodingException异常:{}", e.getMes
age());
        }
        return false;
    }

    /**
```

```
     * 获得Token中的信息无需secret解密也能获得
     * @param token
     * @param claim
     * @return java.lang.String
     * @author Wang926454
     * @date 2018/9/7 16:54
     */
    public static String getClaim(String token, String claim) {
        try {
            DecodedJWT jwt = JWT.decode(token);
            // 只能输出String类型，如果是其他类型返回null
            return jwt.getClaim(claim).asString();
        } catch (JWTDecodeException e) {
            logger.error("解密Token中的公共信息出现JWTDecodeException异常:{}", e.getMessage());
            e.printStackTrace();
        }
        return null;
    }

    /**
     * 生成签名
     * @param account 帐号
     * @return java.lang.String 返回加密的Token
     * @author Wang926454
     * @date 2018/8/31 9:07
     */
    public static String sign(String account, String currentTimeMillis) {
        try {
            // 帐号加JWT私钥加密
            String secret = account + Base64ConvertUtil.decode(encryptJWTKey);
            // 此处过期时间是以毫秒为单位，所以乘以1000
            Date date = new Date(System.currentTimeMillis() + Long.parseLong(accessTokenExpir
Time) * 1000);
            Algorithm algorithm = Algorithm.HMAC256(secret);
            // 附带account帐号信息
            return JWT.create()
                    .withClaim("account", account)
                    .withClaim("currentTimeMillis", currentTimeMillis)
                    .withExpiresAt(date)
                    .sign(algorithm);
        } catch (UnsupportedEncodingException e) {
            logger.error("JWTToken加密出现UnsupportedEncodingException异常:{}", e.getMessage
));
            e.printStackTrace();
        }
        return null;
    }
}
```

4、redis工具类

序列化工具

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.*;

/**
 * Serializable工具(JDK)(也可以使用Protobuf自行百度)
 * @author dolyw.com
 * @date 2018/9/4 15:13
 */
public class SerializableUtil {

    private SerializableUtil() {}

    /**
     * logger
     */
    private static final Logger logger = LoggerFactory.getLogger(SerializableUtil.class);

    /**
     * 序列化
     * @param object
     * @return byte[]
     * @author dolyw.com
     * @date 2018/9/4 15:14
     */
    public static byte[] serializable(Object object) {
        ByteArrayOutputStream baos = null;
        ObjectOutputStream oos = null;
        try {
            baos = new ByteArrayOutputStream();
            oos = new ObjectOutputStream(baos);
            oos.writeObject(object);
            return baos.toByteArray();
        } catch (IOException e) {
            logger.error("SerializableUtil工具类序列化出现IOException异常:{}", e.getMessage());
            e.printStackTrace();
        } finally {
            try {
                if (oos != null) {
                    oos.close();
                }
                if (baos != null) {
                    baos.close();
                }
            } catch (IOException e) {
                logger.error("SerializableUtil工具类反序列化出现IOException异常:{}", e.getMessage());
                e.printStackTrace();
            }
        }
        return null;
    }

    /**
```

```
     * 反序列化
     * @param bytes
     * @return java.lang.Object
     * @author dolyw.com
     * @date 2018/9/4 15:14
     */
    public static Object unserializable(byte[] bytes) {
        ByteArrayInputStream bais = null;
        ObjectInputStream ois = null;
        try {
            bais = new ByteArrayInputStream(bytes);
            ois = new ObjectInputStream(bais);
            return ois.readObject();
        } catch (ClassNotFoundException e) {
            logger.error("SerializableUtil工具类反序列化出现ClassNotFoundException异常:{}", e.get
essage());
        } catch (IOException e) {
            logger.error("SerializableUtil工具类反序列化出现IOException异常:{}", e.getMessage());
        } finally {
            try {
                if (ois != null) {
                    ois.close();
                }
                if (bais != null) {
                    bais.close();
                }
            } catch (IOException e) {
                logger.error("SerializableUtil工具类反序列化出现IOException异常:{}", e.getMessage());
            }
        }
        return null;
    }

}
```

字符串工具类

```
package com.kmair.ky.contract.utils.common;

/**
 * String工具
 * @author dolyw.com
 * @date 2018/9/4 14:48
 */
public class StringUtil {

    private StringUtil() {}

    /**
     * 定义下划线
     */
    private static final char UNDERLINE = '_';

    /**
```

```
     * String为空判断(不允许空格)
     * @param str
     * @return boolean
     * @author dolyw.com
     * @date 2018/9/4 14:49
     */
    public static boolean isBlank(String str) {
        return str == null || "".equals(str.trim());
    }

    /**
     * String不为空判断(不允许空格)
     * @param str
     * @return boolean
     * @author dolyw.com
     * @date 2018/9/4 14:51
     */
    public static boolean isNotBlank(String str) {
        return !isBlank(str);
    }

    /**
     * Byte数组为空判断
     * @param bytes
     * @return boolean
     * @author dolyw.com
     * @date 2018/9/4 15:39
     */
    public static boolean isNull(byte[] bytes) {
        // 根据byte数组长度为0判断
        return bytes == null || bytes.length == 0;
    }

    /**
     * Byte数组不为空判断
     * @param bytes
     * @return boolean
     * @author dolyw.com
     * @date 2018/9/4 15:41
     */
    public static boolean isNotNull(byte[] bytes) {
        return !isNull(bytes);
    }

    /**
     * 驼峰转下划线工具
     * @param param
     * @return java.lang.String
     * @author dolyw.com
     * @date 2018/9/4 14:52
     */
    public static String camelToUnderline(String param) {
        if (isNotBlank(param)) {
            int len = param.length();
```

```
        StringBuilder sb = new StringBuilder(len);
        for (int i = 0; i < len; i++) {
            char c = param.charAt(i);
            if (Character.isUpperCase(c)) {
                sb.append(UNDERLINE);
                sb.append(Character.toLowerCase(c));
            } else {
                sb.append(c);
            }
        }
        return sb.toString();
    } else {
        return "";
    }
}

/**
 * 下划线转驼峰工具
 * @param param
 * @return java.lang.String
 * @author dolyw.com
 * @date 2018/9/4 14:52
 */
public static String underlineToCamel(String param) {
    if (isNotBlank(param)) {
        int len = param.length();
        StringBuilder sb = new StringBuilder(len);
        for (int i = 0; i < len; i++) {
            char c = param.charAt(i);
            if (c == 95) {
                i++;
                if (i < len) {
                    sb.append(Character.toUpperCase(param.charAt(i)));
                }
            } else {
                sb.append(c);
            }
        }
        return sb.toString();
    } else {
        return "";
    }
}

/**
 * 在字符串两周添加''
 * @param param
 * @return java.lang.String
 * @author dolyw.com
 * @date 2018/9/4 14:53
 */
public static String addSingleQuotes(String param) {
    return "\'" + param + "\'";
}
```

```
}
```

redis工具类

```
import com.kmair.ky.contract.common.entity.Constant;
import com.kmair.ky.contract.utils.common.SerializableUtil;
import com.kmair.ky.contract.utils.common.StringUtil;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;

import java.util.Set;

/**
 * JedisUtil(推荐存Byte数组，存Json字符串效率更慢)
 * @author dolyw.com
 * @date 2018/9/4 15:45
 */
@Component
public class JedisUtil {
    private static final Logger logger = LoggerFactory.getLogger(SerializableUtil.class);
    /**
     * 静态注入JedisPool连接池
     * 本来是正常注入JedisUtil，可以在Controller和Service层使用，但是重写Shiro的CustomCach
无法注入JedisUtil
     * 现在改为静态注入JedisPool连接池，JedisUtil直接调用静态方法即可
     * https://blog.csdn.net/W_Z_W_888/article/details/79979103
     */
    private static JedisPool jedisPool;

    @Autowired
    public void setJedisPool(JedisPool jedisPool) {
        JedisUtil.jedisPool = jedisPool;
    }

    /**
     * 获取Jedis实例
     * @param
     * @return redis.clients.jedis.Jedis
     * @author dolyw.com
     * @date 2018/9/4 15:47
     */
    public static synchronized Jedis getJedis() {
        try {
            if (jedisPool != null) {
                return jedisPool.getResource();
            } else {
                return null;
            }
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
        return null;
    }

    /**
     * 释放Jedis资源
     * @param
     * @return void
     * @author dolyw.com
     * @date 2018/9/5 9:16
     */
    public static void closePool() {
        try {
            jedisPool.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * 获取redis键值-object
     * @param key
     * @return java.lang.Object
     * @author dolyw.com
     * @date 2018/9/4 15:47
     */
    public static Object getObject(String key) {
        try (Jedis jedis = jedisPool.getResource()) {
            byte[] bytes = jedis.get(key.getBytes());
            if (StringUtil.isNotNull(bytes)) {
                return SerializableUtil.unserializable(bytes);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    /**
     * 设置redis键值-object
     * @param key
     * @param value
     * @return java.lang.String
     * @author dolyw.com
     * @date 2018/9/4 15:49
     */
    public static String setObject(String key, Object value) {
        try (Jedis jedis = jedisPool.getResource()) {
            return jedis.set(key.getBytes(), SerializableUtil.serializable(value));
        } catch (Exception e) {
            logger.error("设置Redis键值setObject方法异常:key=" + key + " value=" + value + " cau
e=" + e.getMessage());
        }
```

```
        return null;
    }

    /**
     * 设置redis键值-object-expiretime
     * @param key
     * @param value
     * @param expiretime
     * @return java.lang.String
     * @author dolyw.com
     * @date 2018/9/4 15:50
     */
    public static String setObject(String key, Object value, int expiretime) {
        String result;
        try (Jedis jedis = jedisPool.getResource()) {
            result = jedis.set(key.getBytes(), SerializableUtil.serializable(value));
            if (Constant.OK.equals(result)) {
                jedis.expire(key.getBytes(), expiretime);
            }
            return result;
        } catch (Exception e) {
            logger.error("设置Redis键值setObject方法异常:key=" + key + " value=" + value + " cau
e=" + e.getMessage());
        }
        return null;
    }

    /**
     * 获取redis键值-Json
     * @param key
     * @return java.lang.Object
     * @author dolyw.com
     * @date 2018/9/4 15:47
     */
    public static String getJson(String key) {
        try (Jedis jedis = jedisPool.getResource()) {
            return jedis.get(key);
        } catch (Exception e) {
            logger.error("获取Redis键值getJson方法异常:key=" + key + " cause=" + e.getMessage()
;
        }
        return null;
    }

    /**
     * 设置redis键值-Json
     * @param key
     * @param value
     * @return java.lang.String
     * @author Wang926454
     * @date 2018/9/4 15:49
     */
    public static String setJson(String key, String value) {
        try (Jedis jedis = jedisPool.getResource()) {
```

```
            return jedis.set(key, value);
        } catch (Exception e) {
            logger.error("设置Redis键值setJson方法异常:key=" + key + " value=" + value + " cause
" + e.getMessage());
        }
        return null;
    }

    /**
     * 设置redis键值-Json-expiretime
     * @param key
     * @param value
     * @param expiretime
     * @return java.lang.String
     * @author Wang926454
     * @date 2018/9/4 15:50
     */
    public static String setJson(String key, String value, int expiretime) {
        String result;
        try (Jedis jedis = jedisPool.getResource()) {
            result = jedis.set(key, value);
            if (Constant.OK.equals(result)) {
                jedis.expire(key, expiretime);
            }
            return result;
        } catch (Exception e) {
            logger.error("设置Redis键值setJson方法异常:key=" + key + " value=" + value + " cause
" + e.getMessage());
        }
        return null;
    }

    /**
     * 删除key
     * @param key
     * @return java.lang.Long
     * @author Wang926454
     * @date 2018/9/4 15:50
     */
    public static Long delKey(String key) {
        try (Jedis jedis = jedisPool.getResource()) {
            return jedis.del(key.getBytes());
        } catch (Exception e) {
            logger.error("删除Redis的键delKey方法异常:key=" + key + " cause=" + e.getMessage());
        }
        return null;
    }

    /**
     * key是否存在
     * @param key 键
     * @return java.lang.Boolean
     * @author Wang926454
     * @date 2018/9/4 15:51
```

```
     */
    public static Boolean exists(String key) {
        Boolean exists = false;
        try (Jedis jedis = jedisPool.getResource()) {
            exists = jedis.exists(key.getBytes());
        } catch (Exception e) {
            e.printStackTrace();
            logger.error("查询Redis的键是否存在exists方法异常:key=" + key + " cause=" + e.getMe
sage());
        }
        return exists;
    }

    /**
     * 模糊查询获取key集合(keys的速度非常快，但在一个大的数据库中使用它仍然可能造成性能问题
生产不推荐使用)
     * @param key
     * @return java.util.Set<java.lang.String>
     * @author Wang926454
     * @date 2018/9/6 9:43
     */
    public static Set<String> keysS(String key) {
        try (Jedis jedis = jedisPool.getResource()) {
            return jedis.keys(key);
        } catch (Exception e) {
            e.printStackTrace();
            logger.error("模糊查询Redis的键集合keysS方法异常:key=" + key + " cause=" + e.getMe
sage());
        }
        return null;
    }

    /**
     * 模糊查询获取key集合(keys的速度非常快，但在一个大的数据库中使用它仍然可能造成性能问题
生产不推荐使用)
     * @param key
     * @return java.util.Set<java.lang.String>
     * @author Wang926454
     * @date 2018/9/6 9:43
     */
    public static Set<byte[]> keysB(String key) {
        try (Jedis jedis = jedisPool.getResource()) {
            return jedis.keys(key.getBytes());
        } catch (Exception e) {
            e.printStackTrace();
            logger.error("模糊查询Redis的键集合keysB方法异常:key=" + key + " cause=" + e.getMe
sage());
        }
        return null;
    }

    /**
     * 获取过期剩余时间
     * @param key
```

```
     * @return java.lang.String
     * @author Wang926454
     * @date 2018/9/11 16:26
     */
    public static Long ttl(String key) {
        Long result = -2L;
        try (Jedis jedis = jedisPool.getResource()) {
            result = jedis.ttl(key);
            return result;
        } catch (Exception e) {
            e.printStackTrace();
            logger.error("获取Redis键过期剩余时间ttl方法异常:key=" + key + " cause=" + e.getMess
ge());
        }
        return null;
    }
}
```

5、配置信息读取

```
import lombok.Data;

/**
 * 系统属性
 * @author Mr.Wen
 */
@Data
public class SysProperties {
    private String rsaPublicKey;
    private String rsaPrivateKey;
    private String encryptAESKey;
    private String encryptJWTKey;
    private int accessTokenExpireTime;
    private int refreshTokenExpireTime;
    private int shiroCacheExpireTime;
    private String fileCachePath;
}


import lombok.Data;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

/**
 * 系统配置
 * @author Mr.Wen
 */
@Data
@EnableAutoConfiguration
@Configuration
```

```
@PropertySource("classpath:config.properties")
public class SysConfig {
    @Value("${rsaPublicKey}")
    private String rsaPublicKey;
    @Value("${rsaPrivateKey}")
    private String rsaPrivateKey;
    @Value("${encryptAESKey}")
    private String encryptAESKey;
    @Value("${encryptJWTKey}")
    private String encryptJWTKey;
    @Value("${accessTokenExpireTime}")
    private int accessTokenExpireTime;
    @Value("${refreshTokenExpireTime}")
    private int refreshTokenExpireTime;
    @Value("${shiroCacheExpireTime}")
    private int shiroCacheExpireTime;
    @Value("${fileCachePath}")
    private String fileCachePath;

    @Bean(value = "sysProperties",name = "sysProperties")
    public SysProperties init(){
        SysProperties sysProperties = new SysProperties();
        sysProperties.setRsaPublicKey(rsaPublicKey);
        sysProperties.setRsaPrivateKey(rsaPrivateKey);
        sysProperties.setEncryptAESKey(this.encryptAESKey);
        sysProperties.setEncryptJWTKey(this.encryptJWTKey);
        sysProperties.setAccessTokenExpireTime(this.accessTokenExpireTime);
        sysProperties.setRefreshTokenExpireTime(this.refreshTokenExpireTime);
        sysProperties.setShiroCacheExpireTime(this.shiroCacheExpireTime);
        sysProperties.setFileCachePath(fileCachePath);
        return sysProperties;
    }
}
```

6、常量类

```
/**
 * 常量
 * @author dolyw.com
 * @date 2018/9/3 16:03
 */
public class Constant {

    private Constant() {}

    /**
     * redis-OK
     */
    public static final String OK = "OK";

    /**
     * redis过期时间，以秒为单位，一分钟
     */
    public static final int EXRP_MINUTE = 60;
```

```
    /**
     * redis过期时间，以秒为单位，一小时
     */
    public static final int EXRP_HOUR = 60 * 60;

    /**
     * redis过期时间，以秒为单位，一天
     */
    public static final int EXRP_DAY = 60 * 60 * 24;

    /**
     * redis-key-前缀-shiro:cache:
     */
    public static final String PREFIX_SHIRO_CACHE = "shiro:cache:";

    /**
     * redis-key-前缀-shiro:access_token:
     */
    public static final String PREFIX_SHIRO_ACCESS_TOKEN = "shiro:access_token:";
    /**
     * redis-key-前缀-shiro:refresh_token_prefix:
     */
    public static final String PREFIX_SHIRO_USER = "shiro:user_prefix:";
    /**
     * redis-key-前缀-shiro:refresh_token:
     */
    public static final String PREFIX_SHIRO_REFRESH_TOKEN = "shiro:refresh_token:";

    /**
     * JWT-account:
     */
    public static final String ACCOUNT = "account";

    /**
     * JWT-currentTimeMillis:
     */
    public static final String CURRENT_TIME_MILLIS = "currentTimeMillis";

    /**
     * PASSWORD_MAX_LEN
     */
    public static final Integer PASSWORD_MAX_LEN = 8;


    /**
     * 资源目录菜单标识
     */
    public static final String MENU_CODE = "menu";

}
```

7、shiro配置

7.1、权限管理器

shiro的权限管理器，可以继承AuthorizingRealm，重写授权和认证的方法

UserDataServiceImpl是用户信息查询接口，SysRoleServiceImpl是系统角色查询接口，UserData是用户信息存储，SysRole就是系统角色，因为各个系统对此设计不一，就不贴这几个的代码了，具实现参考关键思路即可

```
import com.kmair.ky.contract.common.entity.Constant;
import com.kmair.ky.contract.config.jwt.JwtToken;
import com.kmair.ky.contract.system.user.entity.SysRole;
import com.kmair.ky.contract.system.user.entity.UserData;
import com.kmair.ky.contract.system.user.service.impl.SysRoleServiceImpl;
import com.kmair.ky.contract.system.user.service.impl.UserDataServiceImpl;
import com.kmair.ky.contract.utils.auth.JwtUtil;
import com.kmair.ky.contract.utils.cache.JedisUtil;
import com.kmair.ky.contract.utils.common.StringUtil;
import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authc.SimpleAuthenticationInfo;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.authz.SimpleAuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;

import javax.annotation.Resource;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

/**
 * shiro权限管理器
 * @author Mr.Wen
 */
public class SysShiroRealm extends AuthorizingRealm {
    @Resource
    private UserDataServiceImpl userDataService;
    @Resource
    private SysRoleServiceImpl sysRoleService;

    @Override
    public boolean supports(AuthenticationToken authenticationToken) {
        return authenticationToken instanceof JwtToken;
    }

    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection)
{
        //获取登录用户名
        String workId = (String) principalCollection.getPrimaryPrincipal();
        //查询用户名称
        UserData user = userDataService.getUserByWorkId(workId);
        //根据用户名查询角色
```

```
        List<SysRole> sysRoleList = sysRoleService.selectByUserId(user.getId());
        //添加角色和权限
        SimpleAuthorizationInfo simpleAuthorizationInfo = new SimpleAuthorizationInfo();
        for (SysRole role : sysRoleList) {
            //添加角色
            simpleAuthorizationInfo.addRole(role.getRoleCode());
        }
        return simpleAuthorizationInfo;
    }

    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticatio
Token) throws AuthenticationException {
        String token = (String) authenticationToken.getCredentials();
        // 解密获得account，用于和数据库进行对比
        String account = JwtUtil.getClaim(token, Constant.ACCOUNT);
        // 帐号为空
        if (StringUtil.isBlank(account)) {
            throw new AuthenticationException("Token中帐号为空(The account in Token is empty.)
);
        }
        // 查询用户是否存在
        UserData userData = userDataService.getUserByWorkId(account);
        if (userData == null) {
            throw new AuthenticationException("该帐号不存在(The account does not exist.)");
        }
        // 验证token和refreshToken
        Boolean exists = JedisUtil.exists(Constant.PREFIX_SHIRO_REFRESH_TOKEN + account);
        if (JwtUtil.verify(token) && exists != null && exists) {
            return new SimpleAuthenticationInfo(token, token, "userRealm");
        }
        throw new AuthenticationException("Token已过期(Token expired or incorrect.)");
    }
}
```

7.2、认证过滤器

实现AuthenticationToken接口，定义一个自己的token

```
import lombok.Data;
import org.apache.shiro.authc.AuthenticationToken;

/**
 * @author Mr.Wen
 * token信息
 */
@Data
public class JwtToken implements AuthenticationToken {
    /**
     * Token
     */
    private String token;
```

```
    public JwtToken(String token) {
        this.token = token;
    }

    @Override
    public Object getPrincipal() {
        return token;
    }

    @Override
    public Object getCredentials() {
        return token;
    }
}
```

定义过滤器

代码中的HttpResult是我自定义的http请求返回的信息，换成自己的即可，这里的token过期是通过redis的数据过期来实现的，一共有两个token，一个是refreshToken，另一个是accessToken，accessToken过期，则判断refreshToken是否过期，没过期，则重新颁发一个accessToken，否则让用户重登录

```
/**
 * @author Mr.Wen
 * 过滤器拦截token请求
 */
public class JwtFilter extends BasicHttpAuthenticationFilter implements Filter {
    /**
     * logger
     */
    private static final Logger logger = LoggerFactory.getLogger(JwtFilter.class);

    @Override
    protected boolean isAccessAllowed(ServletRequest request, ServletResponse response, Obj
ct mappedValue) {
        if (this.isLoginAttempt(request, response)) {
            try {
                this.executeLogin(request, response);
            } catch (Exception e) {
                String msg = e.getMessage();
                Throwable throwable = e.getCause();
                if (throwable instanceof SignatureVerificationException) {
                    msg = "Token或者密钥不正确(" + throwable.getMessage() + ")";
                    logger.info(msg);
                } else if (throwable instanceof TokenExpiredException) {
                    HttpServletRequest httpRequest = WebUtils.toHttp(request);
                    String accessToken = httpRequest.getHeader("Authorization");
                    String username = JwtUtil.getClaim(accessToken, Constant.ACCOUNT);
                    Boolean exist = JedisUtil.exists(Constant.PREFIX_SHIRO_REFRESH_TOKEN + usern
me);
                    if(exist != null && exist){
                        return refreshToken(request,response);
                    }else{
                        this.response(4011,response,"token过期，需要重新登录");
```

```
                    logger.info("token过期，需要重新登录",throwable);
                }
                return false;
            } else {
                if (throwable != null) {
                    this.response(500,response,throwable.getMessage());
                    logger.error("服务器错误",throwable);
                }
            }
            return false;
        }
    } else {
        HttpServletRequest httpServletRequest = WebUtils.toHttp(request);
        String httpMethod = httpServletRequest.getMethod();
        String requestUrl = httpServletRequest.getRequestURI();
        logger.info("当前请求 {} Authorization属性(Token)为空 请求类型 {}", requestUrl, httpMet
od);
        this.response(HttpStatus.UNAUTHORIZED.value(),response, "请先登录");
        return false;
    }
    return true;
}

@Override
protected boolean onAccessDenied(ServletRequest request, ServletResponse response) thr
ws Exception {
    this.sendChallenge(request, response);
    return false;
}

@Override
protected boolean isLoginAttempt(ServletRequest request, ServletResponse response) {
    String token = this.getAuthzHeader(request);
    return token != null;
}


@Override
protected boolean executeLogin(ServletRequest request, ServletResponse response) throws
Exception {
    JwtToken token = new JwtToken(this.getAuthzHeader(request));
    this.getSubject(request, response).login(token);
    return true;
}


@Override
protected boolean preHandle(ServletRequest request, ServletResponse response) throws E
ception {
    return super.preHandle(request, response);
}

private boolean refreshToken(ServletRequest request, ServletResponse response) {
    HttpServletRequest httpServletRequest = WebUtils.toHttp(request);
```

```
        String refreshToken = httpServletRequest.getHeader("refreshToken");
        String username = JwtUtil.getClaim(refreshToken, Constant.ACCOUNT);
        if(username == null){
            return false;
        }
        username = username.replace(Constant.PREFIX_SHIRO_USER,"");
        Boolean exists = JedisUtil.exists(Constant.PREFIX_SHIRO_REFRESH_TOKEN + username);
        if (exists != null && exists) {
            //重新签发access_token
            String currentTimeMillis = String.valueOf(System.currentTimeMillis());
            String accessToken = JwtUtil.sign(username, currentTimeMillis);
            JwtToken jwtToken = new JwtToken(accessToken);
            SecurityUtils.getSubject().login(jwtToken);
            SysProperties sysProperties = SpringUtil.getBean("sysProperties");
            HttpServletResponse httpServletResponse = WebUtils.toHttp(response);
            httpServletResponse.setHeader("NEW_ACCESS_TOKEN", accessToken);
            httpServletResponse.setHeader("Access-Control-Expose-Headers", "Authorization");
            JedisUtil.setObject(Constant.PREFIX_SHIRO_ACCESS_TOKEN + currentTimeMillis, sysPr
perties.getAccessTokenExpireTime());
            return true;
        }
        return false;
    }


    private void response(int code,ServletResponse response, String msg) {
        HttpServletResponse httpServletResponse = WebUtils.toHttp(response);
        httpServletResponse.setStatus(HttpStatus.OK.value());
        httpServletResponse.setCharacterEncoding("UTF-8");
        httpServletResponse.setContentType("application/json; charset=utf-8");
        try (PrintWriter out = httpServletResponse.getWriter()) {
            out.append(JSON.toJSONString(HttpResult.customCode(code,msg,null)));
        } catch (IOException e) {
            logger.error("直接返回Response信息出现IOException异常:{}", e.getMessage());
        }
    }
}
```

7.3、shiro缓存处理

我们需要将shiro的数据存储到redis，要关闭shiro自带的缓存，自定义一个缓存管理器

```
import com.kmair.ky.contract.common.entity.Constant;
import com.kmair.ky.contract.config.common.SysProperties;
import com.kmair.ky.contract.utils.auth.JwtUtil;
import com.kmair.ky.contract.utils.cache.JedisUtil;
import com.kmair.ky.contract.utils.common.SerializableUtil;
import org.apache.shiro.cache.Cache;
import org.apache.shiro.cache.CacheException;

import javax.annotation.Resource;
import java.util.*;
```

```
/**
 * @author Mr.Wen
 * @param <K> 键
 * @param <V> 值
 */
@SuppressWarnings("unchecked")
public class CustomCache<K,V> implements Cache<K,V> {

    @Resource
    private SysProperties sysProperties;

    private String getKey(Object key) {
        return Constant.PREFIX_SHIRO_CACHE + JwtUtil.getClaim(key.toString(), Constant.ACCO
NT);
    }

    /**
     * 获取缓存
     */
    @Override
    public Object get(Object key) throws CacheException {
        if(Boolean.FALSE.equals(JedisUtil.exists(this.getKey(key)))){
            return null;
        }
        return JedisUtil.getObject(this.getKey(key));
    }

    /**
     * 保存缓存
     */
    @Override
    public Object put(Object key, Object value) throws CacheException {
        // 读取配置文件，获取Redis的Shiro缓存过期时间
        int shiroCacheExpireTime = sysProperties.getShiroCacheExpireTime();
        // 设置Redis的Shiro缓存
        return JedisUtil.setObject(this.getKey(key), value, shiroCacheExpireTime);
    }

    /**
     * 移除缓存
     */
    @Override
    public Object remove(Object key) throws CacheException {
        if(Boolean.FALSE.equals(JedisUtil.exists(this.getKey(key)))){
            return null;
        }
        JedisUtil.delKey(this.getKey(key));
        return null;
    }

    /**
     * 清空所有缓存
     */
    @Override
```

```
    public void clear() throws CacheException {
        Objects.requireNonNull(JedisUtil.getJedis()).flushDB();
    }

    /**
     * 缓存的个数
     */
    @Override
    public int size() {
        Long size = Objects.requireNonNull(JedisUtil.getJedis()).dbSize();
        return size.intValue();
    }

    /**
     * 获取所有的key
     */
    @Override
    public Set keys() {
        Set<byte[]> keys = Objects.requireNonNull(JedisUtil.getJedis()).keys("*".getBytes());
        Set<Object> set = new HashSet<>();
        for (byte[] bs : keys) {
            set.add(SerializableUtil.unserializable(bs));
        }
        return set;
    }

    /**
     * 获取所有的value
     */
    @Override
    public Collection values() {
        Set keys = this.keys();
        List<Object> values = new ArrayList<>();
        for (Object key : keys) {
            values.add(JedisUtil.getObject(this.getKey(key)));
        }
        return values;
    }
}
```

缓存管理器，定义缓存操作

```
import org.apache.shiro.cache.Cache;
import org.apache.shiro.cache.CacheException;
import org.apache.shiro.cache.CacheManager;

/**
 * 自定义shiro缓存管理器，使用redis存储token信息
 */
public class CustomCacheManager implements CacheManager {
    @Override
    public <K, V> Cache<K, V> getCache(String s) throws CacheException {
        return new CustomCache<K,V>();
    }
```

}

7.4、shiro配置

配置自己的授权认证器、token过滤器，缓存到shiro中，并设置放行的请求

```
import com.kmair.ky.contract.config.jwt.JwtFilter;
import com.kmair.ky.contract.config.shiro.config.CustomCacheManager;
import com.kmair.ky.contract.system.login.service.impl.SysShiroRealm;
import org.apache.shiro.mgt.DefaultSessionStorageEvaluator;
import org.apache.shiro.mgt.DefaultSubjectDAO;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor;
import org.apache.shiro.spring.web.ShiroFilterFactoryBean;
import org.apache.shiro.web.mgt.DefaultWebSecurityManager;
import org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator;
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.servlet.Filter;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

/**
 * shiro配置类
 * @author Mr.Wen
 */
@Configuration
public class ShiroConfig {
    @Bean
    @ConditionalOnMissingBean
    public DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator() {
        DefaultAdvisorAutoProxyCreator defaultAap = new DefaultAdvisorAutoProxyCreator();
        defaultAap.setProxyTargetClass(true);
        return defaultAap;
    }

    @Bean
    public AuthorizingRealm sysShiroRealm() {
        return new SysShiroRealm();
    }


    /**
     * 权限管理，配置主要是Realm的管理认证
     * 需要使用redis存储认证信息，所以，关闭session，重写缓存管理器
     * @return 安全管理器
     */
    @Bean
    public org.apache.shiro.mgt.SecurityManager securityManager() {
        DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
```

```
        securityManager.setRealm(sysShiroRealm());

        // 关闭Shiro自带的session
        DefaultSubjectDAO subjectDAO = new DefaultSubjectDAO();
        DefaultSessionStorageEvaluator defaultSessionStorageEvaluator = new DefaultSessionSt
rageEvaluator();
        defaultSessionStorageEvaluator.setSessionStorageEnabled(false);
        subjectDAO.setSessionStorageEvaluator(defaultSessionStorageEvaluator);

        DefaultWebSecurityManager defaultWebSecurityManager = new DefaultWebSecurityM
nager();
        defaultWebSecurityManager.setSubjectDAO(subjectDAO);
        // 设置自定义Cache缓存
        defaultWebSecurityManager.setCacheManager(new CustomCacheManager());

        return securityManager;
    }

    /**
     * Filter工厂，设置对应的过滤条件和跳转条件
     * @param securityManager 安全管理器
     * @return 过滤组件
     */
    @Bean
    public ShiroFilterFactoryBean shiroFilterFactoryBean(org.apache.shiro.mgt.SecurityManager
securityManager) {
        ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();
        shiroFilterFactoryBean.setSecurityManager(securityManager);

        Map<String, Filter> filterMap = new HashMap<>(16);
        filterMap.put("jwtFilter", new JwtFilter());
        shiroFilterFactoryBean.setFilters(filterMap);

        Map<String, String> map = new HashMap<>();
        //登出
        map.put("/logout", "logout");
        //swagger放行
        map.put("/swagger-ui.html", "anon");
        map.put("/swagger-resources/**", "anon");
        map.put("/v2/api-docs", "anon");
        map.put("/webjars/**", "anon");
        //登录
        map.put("/user/loginByUsernameAndPassword", "anon");
        map.put("/user/refreshToken", "anon");
        map.put("/**","jwtFilter");
        shiroFilterFactoryBean.setLoginUrl("/login/loginPage");
        //首页
        shiroFilterFactoryBean.setSuccessUrl("/index");
        //错误页面，认证不通过跳转
        shiroFilterFactoryBean.setUnauthorizedUrl("/error");
        shiroFilterFactoryBean.setFilterChainDefinitionMap(map);
        return shiroFilterFactoryBean;
    }
```

```
    @Bean
    public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(org.apac
e.shiro.mgt.SecurityManager securityManager) {
        AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor = new Author
zationAttributeSourceAdvisor();
        authorizationAttributeSourceAdvisor.setSecurityManager(securityManager);
        return authorizationAttributeSourceAdvisor;
    }
}
```