```
package com.doglast.boot.security.service;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.util.Base64Utils;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

/**
 * AES加密工具类
 *
 * @author ACGkaka
 * @since 2021-06-18 19:11:03
 */
public class AESUtil {
    /**
     * 日志相关
     */
    private static final Logger LOGGER = LoggerFactory.getLogger(AESUtil.class);
    /**
     * 编码
     */
    private static final String ENCODING = "UTF-8";
    /**
     * 算法定义
     */
    private static final String AES_ALGORITHM = "AES";
    /**
     * 指定填充方式
     * ECB (Electroniccodebook，电子密码本)、CBC (Cipher-block chaining，密码分组链接)、
     * CFB (Cipher feedback，密文反馈)、OFB (Output feedback，输出反馈)、
     * PCBC (Propagating cipher-block chaining，增强型密码分组链接) 。
     *
     * 1、ECB模式：
     *
     * ECB （电子密码本）模式是最简单的块密码加密模式,加密前根据数据块大小（如AES为128位）分成若干块,
     * 之后将每块使用相同的密钥单独通过块加密器密器。这种加密模式的优点就是简单,不需要初始化向量（IV）,
     * 每个数据块独立进行加/解密，利于并行计算，加/解密效率很高。但这种模式中，所有数据都采用相同密钥进行加/解密,
     * 也没有经过任何逻辑运算，相同明文得到相同的密文，所以可能导致“选择明文攻击”的发生。
```

```
     *
     * 2、CBC模式：
     *
     * CBC （密码分组链接）模式是先将明文切分成若干小块，然后每个小块与初始块或者上一段的密文段进行逻辑异或运算后，
     * 再用密钥进行加密。第一个明文块与一个叫初始化向量的数据块进行逻辑异或运算。
     * 这样就有效的解决了ECB模式所暴露出来的问题，即使两个明文块相同，加密后得到的密文块也不相同。
     * 但是缺点也相当明显，如加密过程复杂，效率低等。
     *
     * 3、CFB模式：
     *
     * 与ECB和CBC模式只能够加密块数据不同，CFB模式能够将密文转化成为流密文。
     * 这种加密模式中，由于加密流程和解密流程中被块加密器加密的数据是前块的密文，
     * 因此即使本块明文数据的长度不是数据块大小的整数倍也是不需要填充的，这保证了数据长度在加密前后是相同的。
     *
     * 4、OFB模式：
     *
     * 不再直接加密明文块，其加密过程是先使用块加密器生成密钥流，然后再将密钥流和明文流进行逻辑异或运算得到密文流。
     *
     *
     * NoPadding：
     *
     * 不做任何填充，但是要求明文必须是16字节的整数倍。
     *
     * PKCS5Padding（默认）：
     *
     * 如果明文块少于16个字节（128bit），在明文块末尾补足相应数量的字符，且每个字节的值等于缺少的字符数。
     *
     * 比如明文：{1,2,3,4,5,a,b,c,d,e},缺少6个字节，则补全为{1,2,3,4,5,a,b,c,d,e,6,6,6,6,6,6}
     *
     * ISO10126Padding：
     *
     * 如果明文块少于16个字节（128bit），在明文块末尾补足相应数量的字节，最后一个字符值等于缺少的字符数，其他字符填充随机数。
     *
     * 比如明文：{1,2,3,4,5,a,b,c,d,e},缺少6个字节，则可能补全为{1,2,3,4,5,a,b,c,d,e,5,c,3,G,$,6}
     *
     *
     */
    private static final String CIPHER_PADDING = "AES/ECB/PKCS5Padding";
    private static final String CIPHER_CBC_PADDING = "AES/CBC/PKCS5Padding";
    /**
     * 偏移量(CBC中使用，增强加密算法强度)
     */
```

```
    private static final String IV_SEED = "1234567812345678";

    /**
     * AES加密
     * @param content 待加密内容
     * @param aesKey  密钥
     * @return
     */
    public static String encrypt(String content, String aesKey){
        if(StringUtils.isBlank(content)){
            LOGGER.info("AES encrypt: the content is null!");
            return null;
        }
        //判断秘钥是否为16位
        if(StringUtils.isNotBlank(aesKey) && aesKey.length() == 16){
            try {
                //对密码进行编码
                byte[] bytes = aesKey.getBytes(ENCODING);
                //设置加密算法，生成秘钥
                SecretKeySpec skeySpec = new SecretKeySpec(bytes, AES_ALGORITHM);
                // "算法/模式/补码方式"
                Cipher cipher = Cipher.getInstance(CIPHER_PADDING);
                //选择加密
                cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
                //根据待加密内容生成字节数组
                byte[] encrypted = cipher.doFinal(content.getBytes(ENCODING));
                //返回base64字符串
                return Base64Utils.encodeToString(encrypted);
            } catch (Exception e) {
                LOGGER.info("AES encrypt exception:" + e.getMessage());
                throw new RuntimeException(e);
            }

        }else {
            LOGGER.info("AES encrypt: the aesKey is null or error!");
            return null;
        }
    }

    /**
     * 解密
     *
     * @param content 待解密内容
     * @param aesKey  密码
     * @return
     */
    public static String decrypt(String content, String aesKey){
        if(StringUtils.isBlank(content)){
            LOGGER.info("AES decrypt: the content is null!");
```

```
            return null;
        }
        //判断秘钥是否为16位
        if(StringUtils.isNotBlank(aesKey) && aesKey.length() == 16){
            try {
                //对密码进行编码
                byte[] bytes = aesKey.getBytes(ENCODING);
                //设置解密算法，生成秘钥
                SecretKeySpec skeySpec = new SecretKeySpec(bytes, AES_ALGORITHM);
                // "算法/模式/补码方式"
                Cipher cipher = Cipher.getInstance(CIPHER_PADDING);
                //选择解密
                cipher.init(Cipher.DECRYPT_MODE, skeySpec);

                //先进行Base64解码
                byte[] decodeBase64 = Base64Utils.decodeFromString(content);

                //根据待解密内容进行解密
                byte[] decrypted = cipher.doFinal(decodeBase64);
                //将字节数组转成字符串
                return new String(decrypted, ENCODING);
            } catch (Exception e) {
                LOGGER.info("AES decrypt exception:" + e.getMessage());
                throw new RuntimeException(e);
            }

        }else {
            LOGGER.info("AES decrypt: the aesKey is null or error!");
            return null;
        }
    }



    /**
     * AES_CBC加密
     *
     * @param content 待加密内容
     * @param aesKey  密码
     * @return
     */
    public static String encryptCBC(String content, String aesKey){
        if(StringUtils.isBlank(content)){
            LOGGER.info("AES_CBC encrypt: the content is null!");
            return null;
        }
        //判断秘钥是否为16位
        if(StringUtils.isNotBlank(aesKey) && aesKey.length() == 16){
            try {
```

```
            //对密码进行编码
            byte[] bytes = aesKey.getBytes(ENCODING);
            //设置加密算法，生成秘钥
            SecretKeySpec skeySpec = new SecretKeySpec(bytes, AES_ALGORITHM);
            // "算法/模式/补码方式"
            Cipher cipher = Cipher.getInstance(CIPHER_CBC_PADDING);
            //偏移
            IvParameterSpec iv = new IvParameterSpec(IV_SEED.getBytes(ENCODING));
            //选择加密
            cipher.init(Cipher.ENCRYPT_MODE, skeySpec, iv);
            //根据待加密内容生成字节数组
            byte[] encrypted = cipher.doFinal(content.getBytes(ENCODING));
            //返回base64字符串
            return Base64Utils.encodeToString(encrypted);
        } catch (Exception e) {
            LOGGER.info("AES_CBC encrypt exception:" + e.getMessage());
            throw new RuntimeException(e);
        }

    }else {
        LOGGER.info("AES_CBC encrypt: the aesKey is null or error!");
        return null;
    }
}

/**
 * AES_CBC解密
 *
 * @param content 待解密内容
 * @param aesKey  密码
 * @return
 */
public static String decryptCBC(String content, String aesKey){
    if(StringUtils.isBlank(content)){
        LOGGER.info("AES_CBC decrypt: the content is null!");
        return null;
    }
    //判断秘钥是否为16位
    if(StringUtils.isNotBlank(aesKey) && aesKey.length() == 16){
        try {
            //对密码进行编码
            byte[] bytes = aesKey.getBytes(ENCODING);
            //设置解密算法，生成秘钥
            SecretKeySpec skeySpec = new SecretKeySpec(bytes, AES_ALGORITHM);
            //偏移
            IvParameterSpec iv = new IvParameterSpec(IV_SEED.getBytes(ENCODING));
            // "算法/模式/补码方式"
            Cipher cipher = Cipher.getInstance(CIPHER_CBC_PADDING);
            //选择解密
```

```
                cipher.init(Cipher.DECRYPT_MODE, skeySpec, iv);

                //先进行Base64解码
                byte[] decodeBase64 = Base64Utils.decodeFromString(content);

                //根据待解密内容进行解密
                byte[] decrypted = cipher.doFinal(decodeBase64);
                //将字节数组转成字符串
                return new String(decrypted, ENCODING);
            } catch (Exception e) {
                LOGGER.info("AES_CBC decrypt exception:" + e.getMessage());
                throw new RuntimeException(e);
            }

        }else {
            LOGGER.info("AES_CBC decrypt: the aesKey is null or error!");
            return null;
        }
    }

////////////////////////////////////////////////////////////////


    /**
     * 生成AES秘钥，然后Base64编码
     * 密钥是AES算法实现加密和解密的根本。对称加密算法之所以对称，是因为这类算法对明文的加密和解密需要使用同一个密钥。
     * AES算法在对明文加密的时候，并不是把整个明文一股脑加密成一整段密文，而是把明文拆分成一个个独立的明文块，每一个明文块长度128bit。
     *
     * @return Base64编码
     * @throws Exception
     */
    public static String genKeyAES() throws Exception {
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(128);
        SecretKey key = keyGen.generateKey();
        return Base64.getEncoder().encodeToString(key.getEncoded());
    }

    /**
     * 将Base64编码后的AES秘钥转换成SecretKey对象
     * @param base64Key
     * @return SecretKey对象
     * @throws Exception
     */
    public static SecretKey loadKeyAES(String base64Key) throws Exception {
        byte[] bytes = Base64.getDecoder().decode(base64Key);
        return new SecretKeySpec(bytes, "AES");
```

```
    }

//    /**
//     * AES加密
//     * @param source 加密内容
//     * @param key SecretKey对象
//     * @return 加密后的字节数组
//     * @throws Exception
//     */
//    public static byte[] encryptAES(byte[] source, SecretKey key) throws Exception {
//        Cipher cipher = Cipher.getInstance("AES");
//        cipher.init(Cipher.ENCRYPT_MODE, key);
//        return cipher.doFinal(source);
//    }
//    /**
//     * AES解密
//     * @param source 解密内容
//     * @param key SecretKey对象
//     * @return 解密后的字节数组
//     * @throws Exception
//     */
//    public static byte[] decryptAES(byte[] source, SecretKey key) throws Exception {
//        Cipher cipher = Cipher.getInstance("AES");
//        cipher.init(Cipher.DECRYPT_MODE, key);
//        return cipher.doFinal(source);
//    }
//
//    /**
//     * 加密内容
//     * @param content 需要加密的内容
//     * @return 返回String
//     * @throws Exception
//     */
//    public static String appEncrypt(String content) throws Exception {
//        SecretKey secretKey = AESUtil.loadKeyAES(genKeyAES());
//        // 用AES秘钥加密请求内容
//        byte[] encryptContentRequest = AESUtil.encryptAES(content.getBytes("utf-8"),
secretKey);
//        String data = Base64.getEncoder().encodeToString(encryptContentRequest);
//        return data;
//    }
//
//    /**
//     * 解密内容
//     * @param content 内容
//     * @return
//     * @throws Exception
//     */
//    public static String appDecrypt(String content) throws Exception {
```

```
//        // 用AES秘钥解密请求内容
//        SecretKey secretKey = AESUtil.loadKeyAES(aesKey);
//        byte[] response = AESUtil.decryptAES(Base64.getDecoder().decode(content),
secretKey);
//        return new String(response);
//    }
//
//
//    //加密解密秘钥KEY
//    public static final String aesKey = "a3z4v/RxfgHTRZtlHlrw+5Q==";


    public static String encryptKey(String key){


        char[] chars = key.toCharArray();


        for(int i = 0; i < chars.length; i++ ){

            chars[i] = (char)(chars[i]^'c'^'e'^'t'^'c'^'2'^'8');

        }

        return new String(chars);
    }

    public static void main(String[] args) throws Exception{

        // 前面再加一层位置打乱算法之类的

        String s1 = encryptKey("哥哥你好111");
        System.err.println(s1);

//        String random = RandomStringUtils.random(16,
"abcdefghijklmnopqrstuvwxyz1234567890");
        String random = "1111222233334444";
        System.out.println("随机key:" + random);
        System.out.println("---------加密---------");
        String aesResult = encrypt(s1, random);
        System.out.println(aesResult);


        System.out.println("---------解密---------");
        String decrypt = decrypt(aesResult, random);
        System.out.println("aes解密结果:" + decrypt);
        String s3 = encryptKey(decrypt);
        System.err.println(s3);
```

```
        // 加密:  AES加密 -> Base64加密 -> 密文
        //
        //解密:  Base64解密 -> AES解密 -> 明文
        //
        //测试地址: http://tool.chacuo.net/cryptaes
        // AES支持三种长度的密钥: 128位、192位、256位。
        // 代码中这种就是128位的加密密钥, 16字节 * 8位/字节 = 128位。
//        String random = RandomStringUtils.random(16,
"abcdefghijklmnopqrstuvwxyz1234567890");
//        System.out.println("随机key:" + random);
//        System.out.println();
//
//        System.out.println("---------加密---------");
//        String aesResult = encrypt("123456", random);
//        System.out.println("aes加密结果:" + aesResult);
//        System.out.println();
//
//        System.out.println("---------解密---------");
//        String decrypt = decrypt(aesResult, random);
//        System.out.println("aes解密结果:" + decrypt);
//        System.out.println();
//
//
//        System.out.println("--------AES_CBC加密解密---------");
//        String cbcResult = encryptCBC("测试AES加密12456", random);
//        System.out.println("aes_cbc加密结果:" + cbcResult);
//        System.out.println();
//
//        System.out.println("---------解密CBC---------");
//        String cbcDecrypt = decryptCBC(cbcResult, random);
//        System.out.println("aes解密结果:" + cbcDecrypt);
//        System.out.println();


        ////////////////////////////////////////

//        String s = genKeyAES();
//        System.err.println(s);
//        // SecretKey
//        loadKeyAES(s);
//
//        System.err.println(appEncrypt("sdsdsd"));



    }
```

```
}
```