# 加密工具类RSAUtils

import sun.misc.BASE64Decoder;  
import sun.misc.BASE64Encoder;  
  
import javax.crypto.Cipher;  
import java.math.BigInteger;  
import java.security.KeyFactory;  
import java.security.KeyPair;  
import java.security.KeyPairGenerator;  
import java.security.NoSuchAlgorithmException;  
import java.security.interfaces.RSAPrivateKey;  
import java.security.interfaces.RSAPublicKey;  
import java.security.spec.\*;  
import java.util.HashMap;  
  
public class RSAUtils {  
  
 /\*\*  
 \* 生成公钥和私钥  
 \* @throws java.security.NoSuchAlgorithmException  
 \*  
 \*/  
 public static HashMap<String, Object> getKeys() throws NoSuchAlgorithmException{  
 HashMap<String, Object> map = new HashMap<String, Object>();  
 KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("RSA");  
 keyPairGen.initialize(1024);  
 KeyPair keyPair = keyPairGen.generateKeyPair();  
 RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();  
 RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();  
 map.put("public", publicKey);  
 map.put("private", privateKey);  
 return map;  
 }  
 /\*\*  
 \* 使用模和指数生成RSA公钥  
 \* 注意：【此代码用了默认补位方式，为RSA/None/PKCS1Padding，不同JDK默认的补位方式可能不同，如Android默认是RSA  
 \* /None/NoPadding】  
 \*  
 \* @param modulus  
 \* 模  
 \* @param exponent  
 \* 指数  
 \* @return  
 \*/  
 public static RSAPublicKey getPublicKey(String modulus, String exponent) {  
 try {  
 BigInteger b1 = new BigInteger(modulus);  
 BigInteger b2 = new BigInteger(exponent);  
 KeyFactory keyFactory = KeyFactory.getInstance("RSA");  
 RSAPublicKeySpec keySpec = new RSAPublicKeySpec(b1, b2);  
 return (RSAPublicKey) keyFactory.generatePublic(keySpec);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 /\*\*  
 \* 使用模和指数生成RSA私钥  
 \* 注意：【此代码用了默认补位方式，为RSA/None/PKCS1Padding，不同JDK默认的补位方式可能不同，如Android默认是RSA  
 \* /None/NoPadding】  
 \*  
 \* @param modulus  
 \* 模  
 \* @param exponent  
 \* 指数  
 \* @return  
 \*/  
 public static RSAPrivateKey getPrivateKey(String modulus, String exponent) {  
 try {  
 BigInteger b1 = new BigInteger(modulus);  
 BigInteger b2 = new BigInteger(exponent);  
 KeyFactory keyFactory = KeyFactory.getInstance("RSA");  
 RSAPrivateKeySpec keySpec = new RSAPrivateKeySpec(b1, b2);  
 return (RSAPrivateKey) keyFactory.generatePrivate(keySpec);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 /\*\*  
 \* 公钥加密  
 \*  
 \* @param data  
 \* @param publicKey  
 \* @return  
 \* @throws Exception  
 \*/  
 public static String encryptByPublicKey(String data, RSAPublicKey publicKey)  
 throws Exception {  
 Cipher cipher = Cipher.getInstance("RSA");  
 cipher.init(Cipher.ENCRYPT\_MODE, publicKey);  
 // 模长  
 int key\_len = publicKey.getModulus().bitLength() / 8;  
 // 加密数据长度 <= 模长-11  
 String[] datas = splitString(data, key\_len - 11);  
 String mi = "";  
 //如果明文长度大于模长-11则要分组加密  
 for (String s : datas) {  
 mi += bcd2Str(cipher.doFinal(s.getBytes()));  
 }  
 return mi;  
 }  
  
 public static String encrypt2Base64ByPublicKey(String data, RSAPublicKey publicKey)  
 throws Exception {  
 Cipher cipher = Cipher.getInstance("RSA");  
 cipher.init(Cipher.ENCRYPT\_MODE, publicKey);  
 // 模长  
 int key\_len = publicKey.getModulus().bitLength() / 8;  
 // 加密数据长度 <= 模长-11  
 String[] datas = splitString(data, key\_len - 11);  
 String mi = "";  
 BASE64Encoder base64Encoder = new BASE64Encoder();  
 //如果明文长度大于模长-11则要分组加密  
 for (String s : datas) {  
 mi += base64Encoder.encode(cipher.doFinal(s.getBytes()));  
 }  
 return mi;  
 }  
  
 /\*\*  
 \* 私钥解密  
 \*  
 \* @param data  
 \* @param privateKey  
 \* @return  
 \* @throws Exception  
 \*/  
 public static String decryptByPrivateKey(String data, RSAPrivateKey privateKey)  
 throws Exception {  
 Cipher cipher = Cipher.getInstance("RSA");  
 cipher.init(Cipher.DECRYPT\_MODE, privateKey);  
 //模长  
 int key\_len = privateKey.getModulus().bitLength() / 8;  
 byte[] bytes = data.getBytes();  
 byte[] bcd = ASCII\_To\_BCD(bytes, bytes.length);  
 //System.err.println(bcd.length);  
 //如果密文长度大于模长则要分组解密  
 String ming = "";  
 byte[][] arrays = splitArray(bcd, key\_len);  
 for(byte[] arr : arrays){  
 ming += new String(cipher.doFinal(arr));  
 }  
 return ming;  
 }  
  
 /\*\*  
 \* 私钥解密  
 \*  
 \* @param data  
 \* @param privateKey  
 \* @return  
 \* @throws Exception  
 \*/  
 public static String decryptBase64StrByPrivateKey(String data, RSAPrivateKey privateKey)  
 throws Exception {  
 Cipher cipher = Cipher.getInstance("RSA");  
 cipher.init(Cipher.DECRYPT\_MODE, privateKey);  
 //模长  
 int key\_len = privateKey.getModulus().bitLength() / 8;  
 BASE64Decoder base64Decoder= new BASE64Decoder();  
 byte[] buffer= base64Decoder.decodeBuffer(data);  
 //如果密文长度大于模长则要分组解密  
 String ming = "";  
 byte[][] arrays = splitArray(buffer, key\_len);  
 for(byte[] arr : arrays){  
 ming += new String(cipher.doFinal(arr));  
 }  
 return ming;  
 }  
  
 public static String decryptByPrivateKey(String data, RSAPrivateKey privateKey,int flag) throws Exception {  
 if(flag == 1){  
 return decryptByPrivateKey(data,privateKey);  
 }else{  
 return decryptBase64StrByPrivateKey(data,privateKey);  
 }  
 }  
  
 /\*\*  
 \* ASCII码转BCD码  
 \*  
 \*/  
 public static byte[] ASCII\_To\_BCD(byte[] ascii, int asc\_len) {  
 byte[] bcd = new byte[asc\_len / 2];  
 int j = 0;  
 for (int i = 0; i < (asc\_len + 1) / 2; i++) {  
 bcd[i] = asc\_to\_bcd(ascii[j++]);  
 bcd[i] = (byte) (((j >= asc\_len) ? 0x00 : asc\_to\_bcd(ascii[j++])) + (bcd[i] << 4));  
 }  
 return bcd;  
 }  
  
 public static byte asc\_to\_bcd(byte asc) {  
 byte bcd;  
  
 if ((asc >= '0') && (asc <= '9'))  
 bcd = (byte) (asc - '0');  
 else if ((asc >= 'A') && (asc <= 'F'))  
 bcd = (byte) (asc - 'A' + 10);  
 else if ((asc >= 'a') && (asc <= 'f'))  
 bcd = (byte) (asc - 'a' + 10);  
 else  
 bcd = (byte) (asc - 48);  
 return bcd;  
 }  
  
 /\*\*  
 \* BCD转字符串  
 \*/  
 public static String bcd2Str(byte[] bytes) {  
 char temp[] = new char[bytes.length \* 2], val;  
  
 for (int i = 0; i < bytes.length; i++) {  
 val = (char) (((bytes[i] & 0xf0) >> 4) & 0x0f);  
 temp[i \* 2] = (char) (val > 9 ? val + 'A' - 10 : val + '0');  
  
 val = (char) (bytes[i] & 0x0f);  
 temp[i \* 2 + 1] = (char) (val > 9 ? val + 'A' - 10 : val + '0');  
 }  
 return new String(temp);  
 }  
  
 /\*\*  
 \* 拆分字符串  
 \*/  
 public static String[] splitString(String string, int len) {  
 int x = string.length() / len;  
 int y = string.length() % len;  
 int z = 0;  
 if (y != 0) {  
 z = 1;  
 }  
 String[] strings = new String[x + z];  
 String str = "";  
 for (int i=0; i<x+z; i++) {  
 if (i==x+z-1 && y!=0) {  
 str = string.substring(i\*len, i\*len+y);  
 }else{  
 str = string.substring(i\*len, i\*len+len);  
 }  
 strings[i] = str;  
 }  
 return strings;  
 }  
  
 /\*\*  
 \*拆分数组  
 \*/  
 public static byte[][] splitArray(byte[] data,int len){  
 int x = data.length / len;  
 int y = data.length % len;  
 int z = 0;  
 if(y!=0){  
 z = 1;  
 }  
 byte[][] arrays = new byte[x+z][];  
 byte[] arr;  
 for(int i=0; i<x+z; i++){  
 arr = new byte[len];  
 if(i==x+z-1 && y!=0){  
 System.arraycopy(data, i\*len, arr, 0, y);  
 }else{  
 System.arraycopy(data, i\*len, arr, 0, len);  
 }  
 arrays[i] = arr;  
 }  
 return arrays;  
 }  
  
 public static RSAPublicKey loadPublicKey(String publicKey) throws Exception{  
 try {  
 byte[] buffer= Base64.decode(publicKey, Base64.NO\_WRAP);  
 KeyFactory keyFactory= KeyFactory.getInstance("RSA");  
 X509EncodedKeySpec keySpec= new X509EncodedKeySpec(buffer);  
 return (RSAPublicKey) keyFactory.generatePublic(keySpec);  
 } catch (NoSuchAlgorithmException e) {  
 throw new Exception("无此算法");  
 } catch (InvalidKeySpecException e) {  
 throw new Exception("公钥非法");  
 } catch (NullPointerException e) {  
 throw new Exception("公钥数据为空");  
 }  
 }  
  
 public static RSAPrivateKey loadPrivateKey(String privateKeyStr)  
 throws Exception {  
 try {  
 byte[] buffer = Base64.decode(privateKeyStr, Base64.NO\_WRAP);  
 PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(buffer);  
 KeyFactory keyFactory = KeyFactory.getInstance("RSA");  
 return (RSAPrivateKey) keyFactory.generatePrivate(keySpec);  
 } catch (NoSuchAlgorithmException e) {  
 throw new Exception("无此算法");  
 } catch (InvalidKeySpecException e) {  
 throw new Exception("私钥非法");  
 } catch (NullPointerException e) {  
 throw new Exception("私钥数据为空");  
 }  
 }  
  
}