**java RSA加密教程**

**1.pom依赖**

<!-- rsa begin -->

<dependency>

<groupId>org.apache.tomcat</groupId>

<artifactId>tomcat-util</artifactId>

<version>9.0.8</version>

</dependency>

<dependency>

<groupId>org.bouncycastle</groupId>

<artifactId>bcprov-jdk15on</artifactId>

<version>1.52</version>

</dependency>

<!-- rsa end -->

**2.RSAUtil**

**package** com.shop.utils;

**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.RSAPrivateKeySpec;

**import** java.security.spec.RSAPublicKeySpec;

**import** java.util.HashMap;

**import** javax.crypto.Cipher;

**public** **class** RSAUtil {

/\*\*

\* 生成公钥和私钥

\* **@throws** NoSuchAlgorithmException

\*

\*/

**public** **static** HashMap<String, Object> getKeys() **throws** NoSuchAlgorithmException{

HashMap<String, Object> map = **new** HashMap<String, Object>();

KeyPairGenerator keyPairGen = KeyPairGenerator.*getInstance*("RSA", **new** org.bouncycastle.jce.provider.BouncyCastleProvider());

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

\*

\*

\* **@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", **new** org.bouncycastle.jce.provider.BouncyCastleProvider());

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

\* /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", **new** org.bouncycastle.jce.provider.BouncyCastleProvider());

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", **new** org.bouncycastle.jce.provider.BouncyCastleProvider());

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;

}

/\*\*

\* 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", **new** org.bouncycastle.jce.provider.BouncyCastleProvider());

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** **new** StringBuilder(ming).reverse().toString();

}

/\*\*

\* 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** **void** main(String[] args) **throws** Exception{

HashMap<String, Object> map = *getKeys*();

//生成公钥和私钥

RSAPublicKey publicKey = (RSAPublicKey) map.get("public");

RSAPrivateKey privateKey = (RSAPrivateKey) map.get("private");

//模

String modulus = publicKey.getModulus().toString();

System.***out***.println("pubkey modulus="+modulus);

//公钥指数

String public\_exponent = publicKey.getPublicExponent().toString();

System.***out***.println("pubkey exponent="+public\_exponent);

//私钥指数

String private\_exponent = privateKey.getPrivateExponent().toString();

System.***out***.println("private exponent="+private\_exponent);

//明文

String ming = "111";

//使用模和指数生成公钥和私钥

RSAPublicKey pubKey = RSAUtil.*getPublicKey*(modulus, public\_exponent);

RSAPrivateKey priKey = RSAUtil.*getPrivateKey*(modulus, private\_exponent);

//加密后的密文

String mi = RSAUtil.*encryptByPublicKey*(ming, pubKey);

System.***err***.println("mi="+mi);

//解密后的明文

String ming2 = RSAUtil.*decryptByPrivateKey*(mi, priKey);

System.***err***.println("ming2="+ming2);

}

}

**3.请求指数和模的Controller**

**package** com.shop.controller.wsq;

**import** java.security.interfaces.RSAPrivateKey;

**import** java.security.interfaces.RSAPublicKey;

**import** java.util.HashMap;

**import** javax.servlet.http.HttpServletRequest;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.ResponseBody;

**import** com.alibaba.fastjson.JSONObject;

**import** com.shop.utils.PublicUtils;

**import** com.shop.utils.RSAUtil;

@Controller

**public** **class** LoginRsaController {

// 查询需求

@RequestMapping("/loginRsa.ajax")

@ResponseBody

**public** JSONObject wantTender(HttpServletRequest request) **throws** Exception {

// 构建返回数据

JSONObject json = **new** JSONObject();

// 生成RSA密钥对

HashMap<String, Object> map = RSAUtil.*getKeys*();

RSAPublicKey publicKey = (RSAPublicKey) map.get("public");

RSAPrivateKey privateKey = (RSAPrivateKey) map.get("private");

// 将密钥存起

PublicUtils.*privateMap*.put("privateKey", privateKey);

// 指数， 模

String publicKeyExponent = publicKey.getPublicExponent().toString(16);

String publicKeyModulus = publicKey.getModulus().toString(16);

// 将公钥发送给前端

// System.out.println("key参数：" + publicKeyExponent + ":" + publicKeyModulus);

json.put("exp", publicKeyExponent);

json.put("mod", publicKeyModulus);

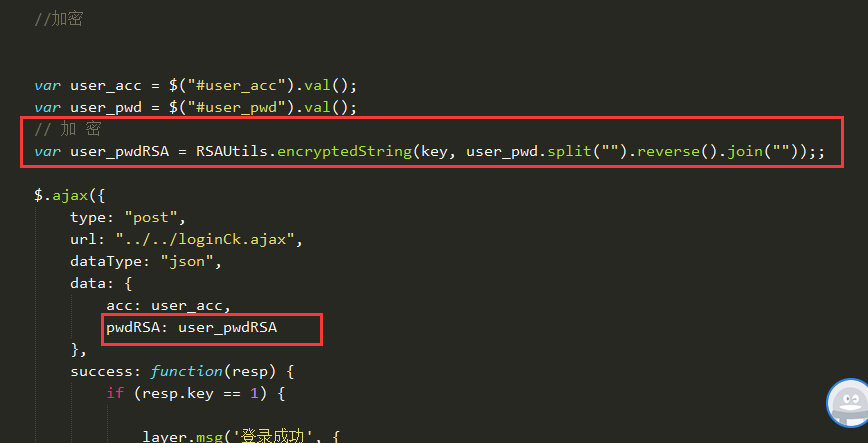
**return** json;

}

}

**4 – js部分，从服务器获取公钥指数和摸**



**5. 将需要加密的内容加密，传至服务器**

**6.在服务器用私钥解码**

