java 实现 RSA 加密

流程分析：

1、甲方构建密钥对儿，将公钥公布给乙方，将私钥保留。

2、甲方使用私钥加密数据，然后用私钥对加密后的数据签名，发送给乙方签名以及加密后的数据；乙方使用公钥、签名来验证待解密数据是否有效，如果有效使用公钥对数据解密。

3、乙方使用公钥加密数据，向甲方发送经过加密后的数据；甲方获得加密数据，通过私钥解密。

通过 java 代码实现如下：

```
import java.security.Key;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;

import java.util.HashMap;
import java.util.Map;

import javax.crypto.Cipher;

/** *//**
* RSA 安全编码组件
*
* @version 1.0
* @since 1.0
*/
public abstract class RSACoder extends Coder {
    public static final String KEY_ALGORITHM = "RSA";
    public static final String SIGNATURE_ALGORITHM = "MD5withRSA";

    private static final String PUBLIC_KEY = "RSAPublicKey";
    private static final String PRIVATE_KEY = "RSAPrivateKey";

    /** *//**
```

```
     * 用私钥对信息生成数字签名
     *
     * @param data
     *                  加密数据
     * @param privateKey
     *                  私钥
     *
     * @return
     * @throws Exception
     */
    public static String sign(byte[] data, String privateKey) throws Exception {
        // 解密由 base64 编码的私钥
        byte[] keyBytes = decryptBASE64(privateKey);

        // 构造 PKCS8EncodedKeySpec 对象
        PKCS8EncodedKeySpec pkcs8KeySpec = new PKCS8EncodedKeySpec(keyBytes);

        // KEY_ALGORITHM 指定的加密算法
        KeyFactory keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);

        // 取私钥匙对象
        PrivateKey priKey = keyFactory.generatePrivate(pkcs8KeySpec);

        // 用私钥对信息生成数字签名
        Signature signature = Signature.getInstance(SIGNATURE_ALGORITHM);
        signature.initSign(priKey);
        signature.update(data);

        return encryptBASE64(signature.sign());
    }

    /** *//**
     * 校验数字签名
     *
     * @param data
     *                  加密数据
     * @param publicKey
     *                  公钥
     * @param sign
     *                  数字签名
     *
     * @return 校验成功返回 true 失败返回 false
     * @throws Exception
     *
```

```
     */
    public static boolean verify(byte[] data, String publicKey, String sign)
            throws Exception {

        // 解密由 base64 编码的公钥
        byte[] keyBytes = decryptBASE64(publicKey);

        // 构造 X509EncodedKeySpec 对象
        X509EncodedKeySpec keySpec = new X509EncodedKeySpec(keyBytes);

        // KEY_ALGORITHM 指定的加密算法
        KeyFactory keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);

        // 取公钥匙对象
        PublicKey pubKey = keyFactory.generatePublic(keySpec);

        Signature signature = Signature.getInstance(SIGNATURE_ALGORITHM);
        signature.initVerify(pubKey);
        signature.update(data);

        // 验证签名是否正常
        return signature.verify(decryptBASE64(sign));
    }

    /** *//**
     * 解密<br>
* 用私钥解密 http://www.5a520.cn http://www.feng123.com
     *
     * @param data
     * @param key
     * @return
     * @throws Exception
     */
    public static byte[] decryptByPrivateKey(byte[] data, String key)
            throws Exception {
        // 对密钥解密
        byte[] keyBytes = decryptBASE64(key);

        // 取得私钥
        PKCS8EncodedKeySpec pkcs8KeySpec = new PKCS8EncodedKeySpec(keyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);
        Key privateKey = keyFactory.generatePrivate(pkcs8KeySpec);

        // 对数据解密
```

```
        Cipher cipher = Cipher.getInstance(keyFactory.getAlgorithm());
        cipher.init(Cipher.DECRYPT_MODE, privateKey);

        return cipher.doFinal(data);
    }

    /** *//**
     * 解密<br>
     * 用私钥解密
     *
     * @param data
     * @param key
     * @return
     * @throws Exception
     */
    public static byte[] decryptByPublicKey(byte[] data, String key)
            throws Exception {
        // 对密钥解密
        byte[] keyBytes = decryptBASE64(key);

        // 取得公钥
        X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(keyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);
        Key publicKey = keyFactory.generatePublic(x509KeySpec);

        // 对数据解密
        Cipher cipher = Cipher.getInstance(keyFactory.getAlgorithm());
        cipher.init(Cipher.DECRYPT_MODE, publicKey);

        return cipher.doFinal(data);
    }

    /** *//**
     * 加密<br>
     * 用公钥加密
     *
     * @param data
     * @param key
     * @return
     * @throws Exception
     */
    public static byte[] encryptByPublicKey(byte[] data, String key)
            throws Exception {
        // 对公钥解密
```

```
        byte[] keyBytes = decryptBASE64(key);

        // 取得公钥
        X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(keyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);
        Key publicKey = keyFactory.generatePublic(x509KeySpec);

        // 对数据加密
        Cipher cipher = Cipher.getInstance(keyFactory.getAlgorithm());
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);

        return cipher.doFinal(data);
    }

    /** *//**
     * 加密<br>
     * 用私钥加密
     *
     * @param data
     * @param key
     * @return
     * @throws Exception
     */
    public static byte[] encryptByPrivateKey(byte[] data, String key)
            throws Exception {
        // 对密钥解密
        byte[] keyBytes = decryptBASE64(key);

        // 取得私钥
        PKCS8EncodedKeySpec pkcs8KeySpec = new PKCS8EncodedKeySpec(keyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);
        Key privateKey = keyFactory.generatePrivate(pkcs8KeySpec);

        // 对数据加密
        Cipher cipher = Cipher.getInstance(keyFactory.getAlgorithm());
        cipher.init(Cipher.ENCRYPT_MODE, privateKey);

        return cipher.doFinal(data);
    }

    /** *//**
     * 取得私钥
     *
     * @param keyMap
```

```
     * @return
     * @throws Exception
     */
    public static String getPrivateKey(Map<String, Object> keyMap)
            throws Exception {
        Key key = (Key) keyMap.get(PRIVATE_KEY);

        return encryptBASE64(key.getEncoded());
    }

    /** *//**
     * 取得公钥
     *
     * @param keyMap
     * @return
     * @throws Exception
     */
    public static String getPublicKey(Map<String, Object> keyMap)
            throws Exception {
        Key key = (Key) keyMap.get(PUBLIC_KEY);

        return encryptBASE64(key.getEncoded());
    }

    /** *//**
     * 初始化密钥
     *
     * @return
     * @throws Exception
     */
    public static Map<String, Object> initKey() throws Exception {
        KeyPairGenerator keyPairGen = KeyPairGenerator
                .getInstance(KEY_ALGORITHM);
        keyPairGen.initialize(1024);

        KeyPair keyPair = keyPairGen.generateKeyPair();

        // 公钥
        RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();

        // 私钥
        RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();

        Map<String, Object> keyMap = new HashMap<String, Object>(2);
```

```
            keyMap.put(PUBLIC_KEY, publicKey);
            keyMap.put(PRIVATE_KEY, privateKey);
            return keyMap;
        }
}
```

再给出一个测试类：

```
import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

import java.util.Map;

/** *//**
*
* @version 1.0
* @since 1.0
*/
public class RSACoderTest {
        private String publicKey;
        private String privateKey;

        @Before
        public void setUp() throws Exception {
            Map<String, Object> keyMap = RSACoder.initKey();

            publicKey = RSACoder.getPublicKey(keyMap);
            privateKey = RSACoder.getPrivateKey(keyMap);
            System.err.println("公钥: \n\r" + publicKey);
            System.err.println("私钥：  \n\r" + privateKey);
        }

        @Test
        public void test() throws Exception {
            System.err.println("公钥加密——私钥解密");
            String inputStr = "abc";
            byte[] data = inputStr.getBytes();

            byte[] encodedData = RSACoder.encryptByPublicKey(data, publicKey);

            byte[] decodedData = RSACoder.decryptByPrivateKey(encodedData,
```

```
                privateKey);

        String outputStr = new String(decodedData);
        System.err.println("加密前: " + inputStr + "\n\r" + "解密后: " + outputStr);
        assertEquals(inputStr, outputStr);

    }

    @Test
    public void testSign() throws Exception {
        System.err.println("私钥加密——公钥解密");
        String inputStr = "sign";
        byte[] data = inputStr.getBytes();

        byte[] encodedData = RSACoder.encryptByPrivateKey(data, privateKey);

        byte[] decodedData = RSACoder
                .decryptByPublicKey(encodedData, publicKey);

        String outputStr = new String(decodedData);
        System.err.println("加密前: " + inputStr + "\n\r" + "解密后: " + outputStr);
        assertEquals(inputStr, outputStr);

        System.err.println("私钥签名——公钥验证签名");
        // 产生签名
        String sign = RSACoder.sign(encodedData, privateKey);
        System.err.println("签名:\r" + sign);

        // 验证签名
        boolean status = RSACoder.verify(encodedData, publicKey, sign);
        System.err.println("状态:\r" + status);
        assertTrue(status);

    }

}
```