Java中4大基本加密算法

加密算法 BASE64 MD5 SHA HMAC

BASE64

Base64是网络上最常见的用于传输8Bit字节代码的编码方式之一，大家可以查看RFC2045～RFC2049，上面有MIME的详细规范。Base64编码可用于在HTTP环境下传递较长的标识信息。例如，在Java Persistence系统Hibernate中，就采用了Base64来将一个较长的唯一标识符（一般为128-bit的UUID）编码为一个字符串，用作HTTP表单和HTTP GET URL中的参数。在其他应用程序中，也常常需要把二进制数据编码为适合放在URL（包括隐藏表单域）中的形式。此时，采用Base64编码具有不可读性，即所编码的数据不会被人用肉眼所直接看到。

java代码实现

```
package com.cn.单向加密;

import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;
/*
BASE64的加密解密是双向的，可以求反解。
BASE64Encoder和BASE64Decoder是非官方JDK实现类。虽然可以在JDK里能找到并使用，但是在API里查不到。
JRE 中 sun 和 com.sun 开头包的类都是未被文档化的，他们属于 java，javax 类库的基础，其中的实现大多数与底层平台有关，
一般来说是不推荐使用的。
BASE64 严格地说，属于编码格式，而非加密算法
主要就是BASE64Encoder、BASE64Decoder两个类，我们只需要知道使用对应的方法即可。
另，BASE加密后产生的字节位数是8的倍数，如果不够位数以=符号填充。
BASE64
按照RFC2045的定义，Base64被定义为：Base64内容传送编码被设计用来把任意序列的8位字节描述为一种不易被人直接识别的形式。
（The Base64 Content-Transfer-Encoding is designed to represent arbitrary sequences of octets in a form that need not be humanly readable.）
常见于邮件、http加密，截取http信息，你就会发现登录操作的用户名、密码字段通过BASE64加密的。
*/

public class BASE64 {
    /**
     * BASE64解密
```

```
     *
     * @param key
     * @return
     * @throws Exception
     */
    public static byte[] decryptBASE64(String key) throws
Exception {
        return (new BASE64Decoder()).decodeBuffer(key);
    }

    /**
     * BASE64加密
     *
     * @param key
     * @return
     * @throws Exception
     */
    public static String encryptBASE64(byte[] key) throws
Exception {
        return (new BASE64Encoder()).encodeBuffer(key);
    }
    public static void main(String[] args) {
     String  str="12345678";
        try {
        String  result1= BASE64.encryptBASE64(str.getBytes());
         System.out.println("result1=====加密数据
==========="+result1);
          byte  result2[]= BASE64.decryptBASE64(result1);
          String  str2=new String(result2);
          System.out.println("str2========解密数据========"+str2);
    } catch (Exception e) {
        e.printStackTrace();
    }
    }
}
```

MD5

MD5即Message-Digest Algorithm 5（信息-摘要算法5），用于确保信息传输完整一致。是计算机广泛使用的杂凑算法之一（又译摘要算法、哈希算法），主流编程语言普遍已有MD5实现。将数据（如汉字）运算为另一固定长度值，是杂凑算法的基础原理，MD5的前身有MD2、MD3和MD4。广泛用于加密和解密技术，常用于文件校验。校验？不管文件多大，经过MD5后都能生成唯一的MD5值。好比现在的ISO校验，都是MD5校验。怎么用？当然是把ISO经过MD5后产生MD5的值。一般下载linux-ISO的朋友都见过下载链接旁边放着MD5的串。就是用来验证文件是否一致的。

java代码实现

```
package com.cn.单向加密;

import java.math.BigInteger;
import java.security.MessageDigest;
/*
MD5(Message Digest algorithm 5，信息摘要算法)
通常我们不直接使用上述MD5加密。通常将MD5产生的字节数组交给BASE64再加密一把，
得到相应的字符串
Digest:汇编
*/
public class MD5 {
    public static final String KEY_MD5 = "MD5";
    public static  String  getResult(String inputStr)
    {
        System.out.println("=======加密前的数据:"+inputStr);
        BigInteger bigInteger=null;
        try {
         MessageDigest md = MessageDigest.getInstance(KEY_MD5);
         byte[] inputData = inputStr.getBytes();
         md.update(inputData);
         bigInteger = new BigInteger(md.digest());
        } catch (Exception e) {e.printStackTrace();}
        System.out.println("MD5加密后:" + bigInteger.toString(16));
        return bigInteger.toString(16);
    }
    public static void main(String args[])
    {
        try {
             String inputStr = "简单加密8888888888888888888";
             getResult(inputStr);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- **MD5算法具有以下特点**
  1.压缩性：任意长度的数据，算出的MD5值长度都是固定的。
  2.容易计算：从元数据计算出的MD5值很容易。
  3.抗修改性：对元数据进行任何改动，哪怕只修改一个字节，所得到的MD5值都有很大区别。
  4.弱抗碰撞：已知原数据和其MD5值，想找到一个具有相同MD5值的数据（即伪造数据）是非常困难的。
  5.强抗碰撞：想找到两个不同的数据，使它们具有相同的MD5值，是非常困难的。
  MD5的作用是让大容量信息在用数字签名软件签署私人密钥前被”压缩”成一种保密的格式（就是把一个任意长度的字节串变换成一定长的十六进制数字串）。

SHA

安全哈希算法（Secure Hash Algorithm）主要适用于数字签名标准（Digital Signature Standard DSS）里面定义的数字签名算法（Digital Signature Algorithm DSA）。对于长度小于2^64位的消息，SHA1会产生一个160位的消息摘要。该算法经过加密专家多年来的发展和改进已日益完善，并被广泛使用。该算法的思想是接收一段明文，然后以一种不可逆的方式将它转换成一段（通常更小）密文，也可以简单的理解为取一串输入码（称为预映射或信息），并把它们转化为长度较短、位数固定的输出序列即散列值（也称为信息摘要或信息认证代码）的过程。散列函数值可以说是对明文的一种”指纹”或是”摘要”所以对散列值的数字签名就可以视为对此明文的数字签名。

java代码实现

```
package com.cn.单向加密;

import java.math.BigInteger;
import java.security.MessageDigest;

/*
SHA(Secure Hash Algorithm，安全散列算法），数字签名等密码学应用中重要的工具，
被广泛地应用于电子商务等信息安全领域。虽然，SHA与MD5通过碰撞法都被破解了，
但是SHA仍然是公认的安全加密算法，较之MD5更为安全*/
public class SHA {
     public static final String KEY_SHA = "SHA";
    public static  String  getResult(String inputStr)
    {
        BigInteger sha =null;
        System.out.println("=======加密前的数据:"+inputStr);
        byte[] inputData = inputStr.getBytes();
        try {
             MessageDigest messageDigest =
MessageDigest.getInstance(KEY_SHA);
             messageDigest.update(inputData);
             sha = new BigInteger(messageDigest.digest());
             System.out.println("SHA加密后:" + sha.toString(32));
        } catch (Exception e) {e.printStackTrace();}
        return sha.toString(32);
    }
    public static void main(String args[])
    {
        try {
             String inputStr = "简单加密";
             getResult(inputStr);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

SHA-1与MD5的比较

二者均由MD4导出，SHA-1和MD5彼此很相似。相应的，他们的强度和其他特性也是相似，但还有以下几点不同：

l 对`强行攻击的安全性`：最显著和最重要的区别是SHA-1摘要比MD5摘要长32 位。使用强行技术，产生任何一个报文使其摘要等于给定报摘要的难度对MD5是2^128数量级的操作，而对SHA-1则是2^160数量级的操作。这样，SHA-1对强行攻击有更大的强度。

l 对`密码分析的安全性`：由于MD5的设计，易受密码分析的攻击，SHA-1显得不易受这样的攻击。

l `速度`：在相同的硬件上，SHA-1的运行速度比MD5慢。

HMAC

HMAC(`Hash Message Authentication Code`，散列消息鉴别码，基于密钥的Hash算法的认证协议。消息鉴别码实现鉴别的原理是，用公开函数和密钥产生一个固定长度的值作为认证标识，用这个标识鉴别消息的完整性。使用一个密钥生成一个固定大小的小数据块，即MAC，并将其加入到消息中，然后传输。接收方利用与发送方共享的密钥进行鉴别认证等。

```
package com.cn.单向加密;

import javax.crypto.KeyGenerator;
import javax.crypto.Mac;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;

import com.cn.comm.Tools;
/*
HMAC
HMAC(Hash Message Authentication Code，散列消息鉴别码，基于密钥的Hash算法的认证协议。
消息鉴别码实现鉴别的原理是，用公开函数和密钥产生一个固定长度的值作为认证标识，用这个标识鉴别消息的完整性。
使用一个密钥生成一个固定大小的小数据块，
即MAC，并将其加入到消息中，然后传输。接收方利用与发送方共享的密钥进行鉴别认证等。*/

/**
 * 基础加密组件
 */
public abstract class HMAC {
    public static final String KEY_MAC = "HmacMD5";

    /**
     * 初始化HMAC密钥
     * @return
     * @throws Exception
     */
    public static String initMacKey() throws Exception {
        KeyGenerator keyGenerator =
```

```
KeyGenerator.getInstance(KEY_MAC);
        SecretKey secretKey = keyGenerator.generateKey();
        return BASE64.encryptBASE64(secretKey.getEncoded());
    }
    /**
     * HMAC加密  ：主要方法
     *
     * @param data
     * @param key
     * @return
     * @throws Exception
     */
    public static String encryptHMAC(byte[] data, String key)
throws Exception {

        SecretKey secretKey = new
SecretKeySpec(BASE64.decryptBASE64(key), KEY_MAC);
        Mac mac = Mac.getInstance(secretKey.getAlgorithm());
        mac.init(secretKey);
        return new String(mac.doFinal(data));
    }

    public static  String  getResult1(String inputStr)
    {
        String path=Tools.getClassPath();
        String fileSource=path+"/file/HMAC_key.txt";
        System.out.println("=======加密前的数据:"+inputStr);
        String  result=null;
        try {
            byte[] inputData = inputStr.getBytes();
            String key = HMAC.initMacKey(); /*产生密钥*/
            System.out.println("Mac密钥:===" + key);
            /*将密钥写文件*/
            Tools.WriteMyFile(fileSource,key);
            result= HMAC.encryptHMAC(inputData, key);
            System.out.println("HMAC加密后:===" + result);
        } catch (Exception e) {e.printStackTrace();}
       return result.toString();
    }

    public static  String  getResult2(String inputStr)
    {
        System.out.println("=======加密前的数据:"+inputStr);
         String path=Tools.getClassPath();
         String fileSource=path+"/file/HMAC_key.txt";
         String key=null;;
        try {
             /*将密钥从文件中读取*/
            key=Tools.ReadMyFile(fileSource);
```

```
            System.out.println("getResult2密钥:===" + key);
        } catch (Exception e1) {
            e1.printStackTrace();}
        String  result=null;
        try {
            byte[] inputData = inputStr.getBytes();
            /*对数据进行加密*/
            result= HMAC.encryptHMAC(inputData, key);
            System.out.println("HMAC加密后:===" + result);
        } catch (Exception e) {e.printStackTrace();}
       return result.toString();
    }
    public static void main(String args[])
    {
        try {
            String inputStr = "简单加密";
            /*使用同一密钥：对数据进行加密：查看两次加密的结果是否一样*/
            getResult1(inputStr);
            getResult2(inputStr);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```