加密，是以某种特殊的算法改变原有的信息数据，使得未授权的用户即使获得了已加密的信息，但因不知解密的方法，仍然无法了解信息的内容。大体上分为**双向加密**和**单向加密**，而双向加密又分为**对称加密**和**非对称加密**(有些资料将加密直接分为对称加密和非对称加密)。   
  
双向加密大体意思就是明文加密后形成密文，可以通过算法还原成明文。而单向加密只是对信息进行了摘要计算，不能通过算法生成明文，单向加密从严格意思上说不能算是加密的一种，应该算是摘要算法吧。具体区分可以参考：   
(本人解释不清呢 …… )   
<http://security.group.iteye.com/group/wiki/1710-one-way-encryption-algorithm>

双向：KeyGenerator，Cipher

对称：

DES、

AES、

非对称：

RSA

,DSA,

单向：MessageDigest

MD5.

SHA(十六进制指纹)

**一、双向加密**   
**(一)、对称加密**   
采用单钥密码系统的加密方法，同一个密钥可以同时用作信息的加密和解密，这种加密方法称为对称加密，也称为单密钥加密。   
需要对加密和解密使用相同密钥的加密算法。由于其速度，对称性加密通常在消息发送方需要加密大量数据时使用。对称性加密也称为密钥加密。   
所谓对称，就是采用这种加密方法的双方使用方式用同样的密钥进行加密和解密。密钥是控制加密及解密过程的指令。   
  
算法是一组规则，规定如何进行加密和解密。因此对称式加密本身不是安全的。 　　   
常用的对称加密有：DES、AES、IDEA、RC2、RC4、SKIPJACK、RC5算法等   
  
对称加密一般java类中中定义成员

//KeyGenerator 提供对称密钥生成器的功能，支持各种算法

private KeyGenerator keygen;

//SecretKey 负责保存对称密钥

private SecretKey deskey;

//Cipher负责完成加密或解密工作

private Cipher c;

//该字节数组负责保存加密的结果

private byte[] cipherByte;

在构造函数中初始化

Security.addProvider(new com.sun.crypto.provider.SunJCE());

//实例化支持DES算法的密钥生成器(算法名称命名需按规定，否则抛出异常)

keygen = KeyGenerator.getInstance("DES");//

//生成密钥

deskey = keygen.generateKey();

//生成Cipher对象,指定其支持的DES算法

c = Cipher.getInstance("DES");

**1. DES**算法为密码体制中的对称密码体制，又被成为美国数据加密标准，是1972年美国IBM公司研制的对称密码 体制加密算法。 明文按64位进行分组, 密钥长64位，密钥事实上是56位参与DES运算（第8、16、24、32、40、48、56、64位是校验位， 使得每个密钥都有奇数个1）分组后的明文组和56位的密钥按位替代或交换的方法形成密文组的加密方法。

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.security.Security;

import javax.crypto.BadPaddingException;

import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.KeyGenerator;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.SecretKey;

public class EncrypDES {

//KeyGenerator 提供对称密钥生成器的功能，支持各种算法

private KeyGenerator keygen;

//SecretKey 负责保存对称密钥

private SecretKey deskey;

//Cipher负责完成加密或解密工作

private Cipher c;

//该字节数组负责保存加密的结果

private byte[] cipherByte;

public EncrypDES() throws NoSuchAlgorithmException, NoSuchPaddingException{

Security.addProvider(new com.sun.crypto.provider.SunJCE());

//实例化支持DES算法的密钥生成器(算法名称命名需按规定，否则抛出异常)

keygen = KeyGenerator.getInstance("DES");

//生成密钥

deskey = keygen.generateKey();

//生成Cipher对象,指定其支持的DES算法

c = Cipher.getInstance("DES");

}

/\*\*

\* 对字符串加密

\*

\* @param str

\* @return

\* @throws InvalidKeyException

\* @throws IllegalBlockSizeException

\* @throws BadPaddingException

\*/

public byte[] Encrytor(String str) throws InvalidKeyException,

IllegalBlockSizeException, BadPaddingException {

// 根据密钥，对Cipher对象进行初始化，ENCRYPT\_MODE表示加密模式

c.init(Cipher.ENCRYPT\_MODE, deskey);

byte[] src = str.getBytes();

// 加密，结果保存进cipherByte

cipherByte = c.doFinal(src);

return cipherByte;

}

/\*\*

\* 对字符串解密

\*

\* @param buff

\* @return

\* @throws InvalidKeyException

\* @throws IllegalBlockSizeException

\* @throws BadPaddingException

\*/

public byte[] Decryptor(byte[] buff) throws InvalidKeyException,

IllegalBlockSizeException, BadPaddingException {

// 根据密钥，对Cipher对象进行初始化，DECRYPT\_MODE表示加密模式

c.init(Cipher.DECRYPT\_MODE, deskey);

cipherByte = c.doFinal(buff);

return cipherByte;

}

/\*\*

\* @param args

\* @throws NoSuchPaddingException

\* @throws NoSuchAlgorithmException

\* @throws BadPaddingException

\* @throws IllegalBlockSizeException

\* @throws InvalidKeyException

\*/

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

EncrypDES de1 = new EncrypDES();

String msg ="郭XX-搞笑相声全集";

byte[] encontent = de1.Encrytor(msg);

byte[] decontent = de1.Decryptor(encontent);

System.out.println("明文是:" + msg);

System.out.println("加密后:" + new String(encontent));

System.out.println("解密后:" + new String(decontent));

}

}

**2. 3DES**又称Triple DES，是DES加密算法的一种模式，它使用3条56位的密钥对3DES   
数据进行三次加密。数据加密标准（DES）是美国的一种由来已久的加密标准，它使用对称密钥加密法，并于1981年被ANSI组织规范为ANSI X.3.92。DES使用56位密钥和密码块的方法，而在密码块的方法中，文本被分成64位大小的文本块然后再进行加密。比起最初的DES，3DES更为 安全。 　　   
3DES（即Triple DES）是DES向AES过渡的加密算法（1999年，NIST将3-DES指定为过渡的加密标准），是DES的一个更安全的变形。它以DES为基本模块，通过组合分组方法设计出分组加密算法，其具体实现如下：   
设Ek()和Dk()代表DES算法的加密和解密过程，K代表DES算法使用的密钥，P代表明文，C代表密文，   
这样， 　　   
3DES加密过程为：C=Ek3(Dk2(Ek1(P)))   
3DES解密过程为：P=Dk1((EK2(Dk3(C)))

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.security.Security;

import javax.crypto.BadPaddingException;

import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.KeyGenerator;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.SecretKey;

public class EncrypDES3 {

// KeyGenerator 提供对称密钥生成器的功能，支持各种算法

private KeyGenerator keygen;

// SecretKey 负责保存对称密钥

private SecretKey deskey;

// Cipher负责完成加密或解密工作

private Cipher c;

// 该字节数组负责保存加密的结果

private byte[] cipherByte;

public EncrypDES3() throws NoSuchAlgorithmException, NoSuchPaddingException {

Security.addProvider(new com.sun.crypto.provider.SunJCE());

// 实例化支持DES算法的密钥生成器(算法名称命名需按规定，否则抛出异常)

keygen = KeyGenerator.getInstance("DESede");

// 生成密钥

deskey = keygen.generateKey();

// 生成Cipher对象,指定其支持的DES算法

c = Cipher.getInstance("DESede");

}

/\*\*

\* 对字符串加密

\*

\* @param str

\* @return

\* @throws InvalidKeyException

\* @throws IllegalBlockSizeException

\* @throws BadPaddingException

\*/

public byte[] Encrytor(String str) throws InvalidKeyException,

IllegalBlockSizeException, BadPaddingException {

// 根据密钥，对Cipher对象进行初始化，ENCRYPT\_MODE表示加密模式

c.init(Cipher.ENCRYPT\_MODE, deskey);

byte[] src = str.getBytes();

// 加密，结果保存进cipherByte

cipherByte = c.doFinal(src);

return cipherByte;

}

/\*\*

\* 对字符串解密

\*

\* @param buff

\* @return

\* @throws InvalidKeyException

\* @throws IllegalBlockSizeException

\* @throws BadPaddingException

\*/

public byte[] Decryptor(byte[] buff) throws InvalidKeyException,

IllegalBlockSizeException, BadPaddingException {

// 根据密钥，对Cipher对象进行初始化，DECRYPT\_MODE表示加密模式

c.init(Cipher.DECRYPT\_MODE, deskey);

cipherByte = c.doFinal(buff);

return cipherByte;

}

/\*\*

\* @param args

\* @throws NoSuchPaddingException

\* @throws NoSuchAlgorithmException

\* @throws BadPaddingException

\* @throws IllegalBlockSizeException

\* @throws InvalidKeyException

\*/

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

EncrypDES3 des = new EncrypDES3();

String msg ="郭XX-搞笑相声全集";

byte[] encontent = des.Encrytor(msg);

byte[] decontent = des.Decryptor(encontent);

System.out.println("明文是:" + msg);

System.out.println("加密后:" + new String(encontent));

System.out.println("解密后:" + new String(decontent));

}

}

**3. AES**密码学中的高级加密标准（Advanced Encryption Standard，AES），又称  高级加密标准   
Rijndael加密法，是美国联邦政府采用的一种区块加密标准。这个标准用来替代原先的DES，已经被多方分析且广为全世界所使用。经过五年的 甄选流程，高级加密标准由美国国家标准与技术研究院（NIST）于2001年11月26日发布于FIPS PUB 197，并在2002年5月26日成为有效的标准。2006年，高级加密标准已然成为对称密钥加密中最流行的算法之一。 　　该算法为比利时密码学家Joan Daemen和Vincent Rijmen所设计，结合两位作者的名字，以Rijndael之命名之，投稿高级加密标准的甄选流程。（Rijdael的发音近于 "Rhinedoll"。）

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.security.Security;

import javax.crypto.BadPaddingException;

import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.KeyGenerator;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.SecretKey;

public class EncrypAES {

//KeyGenerator 提供对称密钥生成器的功能，支持各种算法

private KeyGenerator keygen;

//SecretKey 负责保存对称密钥

private SecretKey deskey;

//Cipher负责完成加密或解密工作

private Cipher c;

//该字节数组负责保存加密的结果

private byte[] cipherByte;

public EncrypAES() throws NoSuchAlgorithmException, NoSuchPaddingException{

Security.addProvider(new com.sun.crypto.provider.SunJCE());

//实例化支持DES算法的密钥生成器(算法名称命名需按规定，否则抛出异常)

keygen = KeyGenerator.getInstance("AES");

//生成密钥

deskey = keygen.generateKey();

//生成Cipher对象,指定其支持的DES算法

c = Cipher.getInstance("AES");

}

/\*\*

\* 对字符串加密

\*

\* @param str

\* @return

\* @throws InvalidKeyException

\* @throws IllegalBlockSizeException

\* @throws BadPaddingException

\*/

public byte[] Encrytor(String str) throws InvalidKeyException,

IllegalBlockSizeException, BadPaddingException {

// 根据密钥，对Cipher对象进行初始化，ENCRYPT\_MODE表示加密模式

c.init(Cipher.ENCRYPT\_MODE, deskey);

byte[] src = str.getBytes();

// 加密，结果保存进cipherByte

cipherByte = c.doFinal(src);

return cipherByte;

}

/\*\*

\* 对字符串解密

\*

\* @param buff

\* @return

\* @throws InvalidKeyException

\* @throws IllegalBlockSizeException

\* @throws BadPaddingException

\*/

public byte[] Decryptor(byte[] buff) throws InvalidKeyException,

IllegalBlockSizeException, BadPaddingException {

// 根据密钥，对Cipher对象进行初始化，DECRYPT\_MODE表示加密模式

c.init(Cipher.DECRYPT\_MODE, deskey);

cipherByte = c.doFinal(buff);

return cipherByte;

}

/\*\*

\* @param args

\* @throws NoSuchPaddingException

\* @throws NoSuchAlgorithmException

\* @throws BadPaddingException

\* @throws IllegalBlockSizeException

\* @throws InvalidKeyException

\*/

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

EncrypAES de1 = new EncrypAES();

String msg ="郭XX-搞笑相声全集";

byte[] encontent = de1.Encrytor(msg);

byte[] decontent = de1.Decryptor(encontent);

System.out.println("明文是:" + msg);

System.out.println("加密后:" + new String(encontent));

System.out.println("解密后:" + new String(decontent));

}

}

**(二)、非对称加密**   
1976年，美国学者Dime和Henman为解决信息公开传送和密钥管理问题，提出一种新的密钥交换协议，允许在不安全的媒体上的通讯双方交换 信息，安全地达成一致的密钥，这就是“公开密钥系统”。相对于“对称加密算法”这种方法也叫做“非对称加密算法”。 与对称加密算法不同，非对称加密算法需要两个密钥：公开密钥（publickey）和私有密钥   
（privatekey）。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密；如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。因为加密和解密使用的是两个不同的密钥，所以这种算法叫作非对称加密算法。   
  
**1. RSA** 公钥加密算法是1977年由Ron Rivest、Adi Shamirh和LenAdleman在（美国麻省理工学院）开发的。RSA取名来自开发他们三者的名字。RSA是目前最有影响力的公钥加密算法，它能够 抵抗到目前为止已知的所有密码攻击，已被ISO推荐为公钥数据加密标准。RSA算法基于一个十分简单的数论事实：将两个大素数相乘十分容易，但那时想要对 其乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。

import java.security.InvalidKeyException;

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.NoSuchAlgorithmException;

import java.security.interfaces.RSAPrivateKey;

import java.security.interfaces.RSAPublicKey;

import javax.crypto.BadPaddingException;

import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.NoSuchPaddingException;

public class EncrypRSA {

/\*\*

\* 加密

\* @param publicKey

\* @param srcBytes

\* @return

\* @throws NoSuchAlgorithmException

\* @throws NoSuchPaddingException

\* @throws InvalidKeyException

\* @throws IllegalBlockSizeException

\* @throws BadPaddingException

\*/

protected byte[] encrypt(RSAPublicKey publicKey,byte[] srcBytes) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException{

if(publicKey!=null){

//Cipher负责完成加密或解密工作，基于RSA

Cipher cipher = Cipher.getInstance("RSA");

//根据公钥，对Cipher对象进行初始化

cipher.init(Cipher.ENCRYPT\_MODE, publicKey);

byte[] resultBytes = cipher.doFinal(srcBytes);

return resultBytes;

}

return null;

}

/\*\*

\* 解密

\* @param privateKey

\* @param srcBytes

\* @return

\* @throws NoSuchAlgorithmException

\* @throws NoSuchPaddingException

\* @throws InvalidKeyException

\* @throws IllegalBlockSizeException

\* @throws BadPaddingException

\*/

protected byte[] decrypt(RSAPrivateKey privateKey,byte[] srcBytes) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException{

if(privateKey!=null){

//Cipher负责完成加密或解密工作，基于RSA

Cipher cipher = Cipher.getInstance("RSA");

//根据私钥，对Cipher对象进行初始化

cipher.init(Cipher.DECRYPT\_MODE, privateKey);

byte[] resultBytes = cipher.doFinal(srcBytes);

return resultBytes;

}

return null;

}

/\*\*

\* @param args

\* @throws NoSuchAlgorithmException

\* @throws BadPaddingException

\* @throws IllegalBlockSizeException

\* @throws NoSuchPaddingException

\* @throws InvalidKeyException

\*/

public static void main(String[] args) throws NoSuchAlgorithmException, InvalidKeyException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException {

EncrypRSA rsa = new EncrypRSA();

String msg = "郭XX-精品相声";

//KeyPairGenerator类用于生成公钥和私钥对，基于RSA算法生成对象

KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("RSA");

//初始化密钥对生成器，密钥大小为1024位

keyPairGen.initialize(1024);

//生成一个密钥对，保存在keyPair中

KeyPair keyPair = keyPairGen.generateKeyPair();

//得到私钥

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

//得到公钥

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

//用公钥加密

byte[] srcBytes = msg.getBytes();

byte[] resultBytes = rsa.encrypt(publicKey, srcBytes);

//用私钥解密

byte[] decBytes = rsa.decrypt(privateKey, resultBytes);

System.out.println("明文是:" + msg);

System.out.println("加密后是:" + new String(resultBytes));

System.out.println("解密后是:" + new String(decBytes));

}

}

**2. DSA**   
Digital Signature Algorithm (DSA)是Schnorr和ElGamal签名算法的变种，被美国NIST作为DSS(DigitalSignature Standard)。(感觉有点复杂，没有附代码)   
详见[http://63938525.iteye.com/blog/1051565](http://63938525.iteye.com/blog/1051565" \t "_blank)   
  
**(三)、题外话 MySQL加密解密函数**   
MySQL有两个函数来支持这种类型的加密，分别叫做ENCODE()和DECODE()。   
下面是一个简单的实例：

mysql> INSERT INTO users (username,password) VALUES ('joe',ENCODE('guessme','abr'));

Query OK， 1 row affected （0.14 sec）

其中，Joe的密码是guessme，它通过密钥abracadabra被加密。要注意的是，加密完的结果是一个二进制字符串，如下所示：   
  
提示：虽然ENCODE()和DECODE()这两个函数能够满足大多数的要求，但是有的时候您希望使用强度更高的加密手段。在这种情况下，您可以使用AES\_ENCRYPT()和AES\_DECRYPT()函数，它们的工作方式是相同的，但是加密强度更高。   
  
  
单向加密与双向加密不同，一旦数据被加密就没有办法颠倒这一过程。因此密码的验证包括对用户输入内容的重新加密，并将它与保存的密文进行比对，看 是否匹配。一种简单的单向加密方式是MD5校验码。MySQL的MD5（）函数会为您的数据创建一个“指纹”并将它保存起来，供验证测试使用。下面就是如 何使用它的一个简单例子：

mysql> INSERT INTO users (username,password) VALUES ('joe',MD5('guessme'));

Query OK， 1 row affected （0.00 sec）

或者，您考虑一下使用ENCRYPT（）函数，它使用系统底层的crypt（）系统调用来完成加密。这个函数有两个参数：一个是要被加密的字符串，另一个 是双（或者多）字符的“salt”。它然后会用salt加密字符串；这个salt然后可以被用来再次加密用户输入的内容，并将它与先前加密的字符串进行比 对。下面一个例子说明了如何使用它：

mysql> INSERT INTO users (username,password) VALUES('joe'， ENCRYPT('guessme','ab'));

Query OK， 1 row affected （0.00 sec）

提示：ENCRYPT()只能用在UNIX、LINIX系统上，因为它需要用到底层的crypt()库。

**二、单向加密(信息摘要)**   
Java一般需要获取对象MessageDigest来实现单项加密(信息摘要)。   
**1. 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。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）。   
除了MD5以外，其中比较有名的还有sha-1、RIPEMD以及Haval等

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

public class EncrypMD5 {

public byte[] eccrypt(String info) throws NoSuchAlgorithmException{

//根据MD5算法生成MessageDigest对象

MessageDigest md5 = MessageDigest.getInstance("MD5");

byte[] srcBytes = info.getBytes();

//使用srcBytes更新摘要

md5.update(srcBytes);

//完成哈希计算，得到result

byte[] resultBytes = md5.digest();

return resultBytes;

}

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

String msg = "郭XX-精品相声技术";

EncrypMD5 md5 = new EncrypMD5();

byte[] resultBytes = md5.eccrypt(msg);

System.out.println("密文是：" + new String(resultBytes));

System.out.println("明文是：" + msg);

}

}

**2. SHA** 是一种数据加密算法，该算法经过加密专家多年来的发展和改进已日益完善，现在已成为公认的最安全的散列算法之一，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。散列函数值可以说时对明文的一种“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。

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

public class EncrypSHA {

public byte[] eccrypt(String info) throws NoSuchAlgorithmException{

MessageDigest md5 = MessageDigest.getInstance("SHA");

byte[] srcBytes = info.getBytes();

//使用srcBytes更新摘要

md5.update(srcBytes);

//完成哈希计算，得到result

byte[] resultBytes = md5.digest();

return resultBytes;

}

/\*\*

\* @param args

\* @throws NoSuchAlgorithmException

\*/

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

String msg = "郭XX-精品相声技术";

EncrypSHA sha = new EncrypSHA();

byte[] resultBytes = sha.eccrypt(msg);

System.out.println("明文是：" + msg);

System.out.println("密文是：" + new String(resultBytes));

}

}

附件中是以上几种的源代码，附带额外的两种使用方式。   
  
增加一种关于文件的哈希算法源代码：

import java.io.FileInputStream;

import java.io.InputStream;

import java.security.MessageDigest;

public class FileHashUtil {

public static final char[] hexChar = {

'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f' };

public static final String[] hashTypes = new String[] { "MD2", "MD5", "SHA1", "SHA-256", "SHA-384", "SHA-512" };

public void MD5File(String fileName) throws Exception{

//String fileName = args[0];

System.out.println("需要获取hash的文件为：　" + fileName);

java.util.List<MessageDigest> mds = new java.util.ArrayList<MessageDigest>();

for (String hashType : hashTypes) {

MessageDigest md = MessageDigest.getInstance(hashType);

mds.add(md);

}

InputStream fis = null;

try {

fis = new FileInputStream(fileName);

byte[] buffer = new byte[1024];

int numRead = 0;

while ((numRead = fis.read(buffer)) > 0) {

for (MessageDigest md : mds) {

md.update(buffer, 0, numRead);

}

}

} catch (Exception ex) {

ex.printStackTrace();

} finally {

if (fis != null) {

fis.close();

}

}

for (MessageDigest md : mds) {

System.out.println(md.getAlgorithm() + " == " + toHexString(md.digest()));

}

}

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

String[] fileName = new String[] {"D:/hapfish/ShellFolder.java","D:/hapfish/ShellFolder - 副本.java",

"E:/ShellFolder - 副本.java","E:/ShellFolder.txt","D:/hapfish/ShellFolder.jpg",

"E:/ShellFolder增加字符.txt","D:/hapfish/birosoft.jar"};

FileHashUtil files = new FileHashUtil();

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

files.MD5File(fileName[i]);

}

}

public static String toHexString(byte[] b) {

StringBuilder sb = new StringBuilder(b.length \* 2);

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

sb.append(hexChar[(b[i] & 0xf0) >>> 4]);

sb.append(hexChar[b[i] & 0x0f]);

}

return sb.toString();

}

}

运行说明

"D:/hapfish/ShellFolder.java",

"D:/hapfish/ShellFolder - 副本.java",

"E:/ShellFolder - 副本.java",

"E:/ShellFolder.txt",

"D:/hapfish/ShellFolder.jpg",

以上五个文件是同一文件经过复制、改扩展名的，最后计算哈希结果是一致的。

"E:/ShellFolder增加字符.txt" 增加了几个字符串，就不一样了

"D:/hapfish/birosoft.jar" 完全不相关的另外一个文件

运行结果：

需要获取hash的文件为：　D:/hapfish/ShellFolder.java

MD2 == 3a755a99c5e407005cd45ebd856b4649

MD5 == 5d08d440fa911d1e418c69a90b83cd86

SHA1 == 522c8c4f4ff1dd669e251c2ab854c3033a51ca63

SHA-256 == d1feb0c73c10a759e88bd240cb9d56d0598b4ff83a0704c6679f7ba12f6c4d99

SHA-384 == 8f8c9da4cd7241c58af3c52b49199033f2dcf3d67f421753999f87511618d9ea2d738e8c16b9b68a7572d06108ff10f6

SHA-512 == 4711579daee3ddacbaea189310348956cb43bcaaf0099f3be047b06f16c1a20a6b71ee3a4ee018128d647e9f2ef0d644747672238e49a8da3d0cd26dfe597458

需要获取hash的文件为：　D:/hapfish/ShellFolder - 副本.java

MD2 == 3a755a99c5e407005cd45ebd856b4649

MD5 == 5d08d440fa911d1e418c69a90b83cd86

SHA1 == 522c8c4f4ff1dd669e251c2ab854c3033a51ca63

SHA-256 == d1feb0c73c10a759e88bd240cb9d56d0598b4ff83a0704c6679f7ba12f6c4d99

SHA-384 == 8f8c9da4cd7241c58af3c52b49199033f2dcf3d67f421753999f87511618d9ea2d738e8c16b9b68a7572d06108ff10f6

SHA-512 == 4711579daee3ddacbaea189310348956cb43bcaaf0099f3be047b06f16c1a20a6b71ee3a4ee018128d647e9f2ef0d644747672238e49a8da3d0cd26dfe597458

需要获取hash的文件为：　E:/ShellFolder - 副本.java

MD2 == 3a755a99c5e407005cd45ebd856b4649

MD5 == 5d08d440fa911d1e418c69a90b83cd86

SHA1 == 522c8c4f4ff1dd669e251c2ab854c3033a51ca63

SHA-256 == d1feb0c73c10a759e88bd240cb9d56d0598b4ff83a0704c6679f7ba12f6c4d99

SHA-384 == 8f8c9da4cd7241c58af3c52b49199033f2dcf3d67f421753999f87511618d9ea2d738e8c16b9b68a7572d06108ff10f6

SHA-512 == 4711579daee3ddacbaea189310348956cb43bcaaf0099f3be047b06f16c1a20a6b71ee3a4ee018128d647e9f2ef0d644747672238e49a8da3d0cd26dfe597458

需要获取hash的文件为：　E:/ShellFolder.txt

MD2 == 3a755a99c5e407005cd45ebd856b4649

MD5 == 5d08d440fa911d1e418c69a90b83cd86

SHA1 == 522c8c4f4ff1dd669e251c2ab854c3033a51ca63

SHA-256 == d1feb0c73c10a759e88bd240cb9d56d0598b4ff83a0704c6679f7ba12f6c4d99

SHA-384 == 8f8c9da4cd7241c58af3c52b49199033f2dcf3d67f421753999f87511618d9ea2d738e8c16b9b68a7572d06108ff10f6

SHA-512 == 4711579daee3ddacbaea189310348956cb43bcaaf0099f3be047b06f16c1a20a6b71ee3a4ee018128d647e9f2ef0d644747672238e49a8da3d0cd26dfe597458

需要获取hash的文件为：　D:/hapfish/ShellFolder.jpg

MD2 == 3a755a99c5e407005cd45ebd856b4649

MD5 == 5d08d440fa911d1e418c69a90b83cd86

SHA1 == 522c8c4f4ff1dd669e251c2ab854c3033a51ca63

SHA-256 == d1feb0c73c10a759e88bd240cb9d56d0598b4ff83a0704c6679f7ba12f6c4d99

SHA-384 == 8f8c9da4cd7241c58af3c52b49199033f2dcf3d67f421753999f87511618d9ea2d738e8c16b9b68a7572d06108ff10f6

SHA-512 == 4711579daee3ddacbaea189310348956cb43bcaaf0099f3be047b06f16c1a20a6b71ee3a4ee018128d647e9f2ef0d644747672238e49a8da3d0cd26dfe597458

需要获取hash的文件为：　E:/ShellFolder增加字符.txt

MD2 == f2717c24c6c0e110457bd17221c9ca6c

MD5 == c49e353a7c4c26bd7ccb5e90917c230f

SHA1 == 477c8a9e465bfaa4be42d35c032a17f7e6b42b97

SHA-256 == 9fa18adaf242ebcdc6563922d84c2a163c82e1a24db2eb2b73978ed1f354a8a3

SHA-384 == 4eee8f8e6d64d21c15dc01fa049f4d12a3b8e1d94d87763fe0bea75ab5ea8432fa8251289ece45ee39fe3d36b3c3020c

SHA-512 == e852ec0ff77250be497389d2f5a1818c18bb66106b9905c4ee26fe0d256eb3b77e0ce9a28a84e4b67e4332ba37ec3aa7518148e3a682318c0fc34c391f45c201

需要获取hash的文件为：　D:/hapfish/birosoft.jar

MD2 == 38c5e1404718916dec59c33cafc909b3

MD5 == dc3e2cc4fb3949cf3660e0f5f8c3fba3

SHA1 == cde3dc25498afc5a563af0bb0eb54dc45f71bb28

SHA-256 == adf6a961c70c6ea677dff066fc5d896fb0beb4dd442ca0eb619ae1d1b04291e5

SHA-384 == fe7c6b754893c53ebd82bb53703fb5cc32115c9a38f98072f73def90729b271ee3c5c78e258bd9ff5ee5476193c2178b

SHA-512 == a15376f327256a6e049dfbdc5c2ad3a98bffccc6fa92ee01ff53db6b04471ca0f45ca28f76ff4a6911b57825afa046671299141f2499d71f1dac618c92385491

最后，把运行结果贴出来有点占空间，主要为了说明表述自己的猜想。一般来说同一哈希算法对同一文件(镜像、扩展名被修改)所产生的结果应该是一致的。   
  
因此有个猜想，在baidu文库、腾讯的群共享上传时，先会判断是否有相同文件，从某种可能上来说也采用了对文件的哈希算法，毕竟从本地运算一个哈希算法后获得的数值要比把整个文件传过去比较实惠得多。而且字符串的比较也是很方便的。   
  
对于某一种哈希算法，存在一种可能：就是两个不同的文件，计算出来的哈希值可能是一样的。当然为了保险，可以用两种甚至更多的哈希算法，只有在每种算法获得的哈希值都相同时，才能判断是同一个文件。   
如果我们也对用户上传的文件进行哈希计算的话，就可以节省资源，同样的文件按理说可以减少上传次数……