概述

在下载很多软件，特别是大型软件园的时候，一般都会有一个校验值。如下图

这里写图片描述

有些时候网络原因下载到破损的文件。更有甚至下载的软件园被篡改，加入一些其他东西。那么就需要校验来验证下载的软
法就发挥作用了.

消息摘要算法的分类

消息摘要算法主要分为三大类：md（message digest，消息摘要算法），SHA（secure hash algorithm，安全散列算法）
code，消息认证码算法）

基于这些消息摘要算法衍生出RipeMD系列、Tiger、GOST3411、Whirlpool算法。

MD算法

MD5算法是消息摘要算法的首要代表，其前身有MD2、MD3、MD4算法，MD5算法由MD4、MD3、MD2算法改进而来。不
要获取一个随机长度的信息并产生一个128位的信息摘要，如果把这个128位的二进制摘要信息换成16进制，可以得到一个

SHA算法

SHA算法（安全散列算法）是消息摘要算法的一种，被广泛认可为MD5算法的继任者。SHA算法家族目前有SHA-1、SHA-
SHA-512五种算法，通常后面的算法被统称为SHA-2算法。

SHA算法是在MD4算法的基础上演进而来，SHA与MD算法的不同之处主要在于摘要长度，MD算法的摘要长度为128位，而
性更高。

长度对比

算法	摘要长度
SHA-1	160
SHA-256	256
SHA-384	384
SHA-512	512
SHA-224	224

MAC算法

Mac(message Authentication code，消息认证码算法)是含有密钥散列函数算法，兼容了MD和SHA算法的特性，并在此基
HMAC（keyed-hash message Authentication code）。

MAC算法主要集合了MD和SHA两个系列消息算法。MD系列有HmacMD2、HmacMD4和HmacMD5三种算法。SHA系列算
HmacSHA224、HmacSHA256、HmacSHA384、HmacSHA512五种算法。

经MAC算法得到的摘要值也可以十六进制编码表示，其摘要长度与参与实现的算法摘要长度相同，例如HmacSHA1算法的
一样，都是160位二进制，换算成十六进制编码为40位。

java中使用MD5、SHA、HMAC算法

算法类

```
import java.security.MessageDigest;

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

import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;

/**
 * 基础加密组件
 *
 * @author 梁栋
 * @version 1.0
 * @since 1.0
 */
public abstract class Coder {
    public static final String KEY_SHA = "SHA";
    public static final String KEY_MD5 = "MD5";

    /**
     * MAC算法可选以下多种算法
     *
     * <pre>
     * HmacMD5
     * HmacSHA1
     * HmacSHA256
     * HmacSHA384
     * HmacSHA512
     * </pre>
     */
    public static final String KEY_MAC = "HmacMD5";

    /**
     * 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);
    }

    /**
     * MD5加密
     *
     * @param data
     * @return
     * @throws Exception
     */
    public static byte[] encryptMD5(byte[] data) throws Exception {

        MessageDigest md5 = MessageDigest.getInstance(KEY_MD5);
        md5.update(data);
```

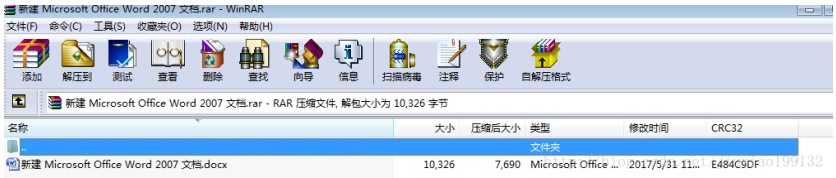
```
        return md5.digest();

    }

    /**
     * SHA加密
     *
     * @param data
     * @return
     * @throws Exception
     */
    public static byte[] encryptSHA(byte[] data) throws Exception {

        MessageDigest sha = MessageDigest.getInstance(KEY_SHA);
        sha.update(data);

        return sha.digest();

    }

    /**
     * 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 encryptBASE64(secretKey.getEncoded());
    }

    /**
     * HMAC加密
     *
     * @param data
     * @param key
     * @return
     * @throws Exception
     */
    public static byte[] encryptHMAC(byte[] data, String key) throws Exception {

        SecretKey secretKey = new SecretKeySpec(decryptBASE64(key), KEY_MAC);
        Mac mac = Mac.getInstance(secretKey.getAlgorithm());
        mac.init(secretKey);

        return mac.doFinal(data);

    }
}
```

测试类

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

import org.apache.commons.codec.binary.Hex;
import org.junit.Test;

public class CoderTest {

    @Test
    public void test() throws Exception {
        String inputStr = "简单加密";
        System.err.println("原文:\n" + inputStr);

        byte[] inputData = inputStr.getBytes();
        String code = Coder.encryptBASE64(inputData);

        System.err.println("BASE64加密后:\n" + code);
```

```
        byte[] output = Coder.decryptBASE64(code);

        String outputStr = new String(output);

        System.err.println("BASE64解密后:\n" + outputStr);

        // 验证BASE64加密解密一致性
        assertEquals(inputStr, outputStr);

        // 验证MD5对于同一内容加密是否一致
        assertArrayEquals(Coder.encryptMD5(inputData), Coder
                .encryptMD5(inputData));

        // 验证SHA对于同一内容加密是否一致
        assertArrayEquals(Coder.encryptSHA(inputData), Coder
                .encryptSHA(inputData));

        String key = Coder.initMacKey();
        System.err.println("Mac密钥:\n" + key);

        // 验证HMAC对于同一内容，同一密钥加密是否一致
        assertArrayEquals(Coder.encryptHMAC(inputData, key), Coder.encryptHMAC(
                inputData, key));

        System.err.println("MD5:\n" + Hex.encodeHexString(Coder.encryptMD5(inputData)));

        System.err.println("SHA:\n" + Hex.encodeHexString(Coder.encryptSHA(inputData)));

        System.err.println("HMAC:\n" + Hex.encodeHexString(Coder.encryptHMAC(inputData, inputStr)));
    }
}
```

CRC算法



这里写图片描述

crc(cyclic redundancy check，循环冗余校验，又称作，奇偶校验码）是一种根据网络数据包或电脑文件等数据产生简短主要用来检测或校验数据传输或者保存后可能出现的错误。它是利用除法及余数的原理来作错误侦测的。
在数据传输过程中，无论传输系统的设计再怎么完美，差错总会存在，这种差错可能会导致在链路上传输的一个或者多个1，或者1变为0），从而接受方接收到错误的数据。为尽量提高接受方收到数据的正确率，在接收方接收数据之前需要对数据结果为正确时接收方才真正收下数据。检测的方式有多种，常见的有奇偶校验、因特网校验和循环冗余校验等。

java中crc32算法

经过crc32算法处理后可以得到一个8位十六进制字符串

```
public static void main(String[] args) {

        String str = "测试-crC32";
        CRC32 c = new CRC32();
        c.update(str.getBytes());
        String hex = Long.toHexString(c.getValue());
        System.out.println("原文:\n" + str);
        System.out.println("crc32:\n" + hex);
    }
```