MD5（单向散列算法）的全称是Message-Digest Algorithm 5（信息-摘要算法），经MD2、MD3和MD4发展而来。MD5算法的使用不需要支付任何版权费用。  
  
    MD5功能：  
    输入任意长度的信息，经过处理，输出为128位的信息（数字指纹）；  
    不同的输入得到的不同的结果（唯一性）；  
    根据128位的输出结果不可能反推出输入的信息（不可逆）；   
  
    MD5属不属于加密算法：  
    认为不属于的人是因为他们觉得不能从密文（散列值）反过来得到原文，即没有解密算法，所以这部分人认为MD5只能属于算法，不能称为加密算法；  
    认为属于的人是因为他们觉得经过MD5处理后看不到原文，即已经将原文加密，所以认为MD5属于加密算法；我个人支持后者。  
  
    MD5用途：  
    1、防止被篡改：  
    1）比如发送一个电子文档，发送前，我先得到MD5的输出结果a。然后在对方收到电子文档后，对方也得到一个MD5的输出结果b。如果a与b一样就代表中途未被篡改。2）比如我提供文件下载，为了防止不法分子在安装程序中添加木马，我可以在网站上公布由安装文件得到的MD5输出结果。3）SVN在检测文件是否在CheckOut后被修改过，也是用到了MD5.  
  
    2、防止直接看到明文：  
    现在很多网站在数据库存储用户的密码的时候都是存储用户密码的MD5值。这样就算不法分子得到数据库的用户密码的MD5值，也无法知道用户的密码(其实这样是不安全的，后面我会提到)。（比如在UNIX系统中用户的密码就是以MD5（或其它类似的算法）经加密后存储在文件系统中。当用户登录的时候，系统把用户输入的密码计算成MD5值，然后再去和保存在文件系统中的MD5值进行比较，进而确定输入的密码是否正确。通过这样的步骤，系统在并不知道用户密码的明码的情况下就可以确定用户登录系统的合法性。这不但可以避免用户的密码被具有系统管理员权限的用户知道，而且还在一定程度上增加了密码被破解的难度。）  
  
    3、防止抵赖（数字签名）：  
    这需要一个第三方认证机构。例如A写了一个文件，认证机构对此文件用MD5算法产生摘要信息并做好记录。若以后A说这文件不是他写的，权威机构只需对此文件重新产生摘要信息，然后跟记录在册的摘要信息进行比对，相同的话，就证明是A写的了。这就是所谓的“数字签名”。  
  
    MD5算法过程：  
    对MD5算法简要的叙述可以为：MD5以512位分组来处理输入的信息，且每一分组又被划分为16个32位子分组，经过了一系列的处理后，算法的输出由四个32位分组组成，将这四个32位分组级联后将生成一个128位散列值。  
  
     第一步、填充：如果输入信息的长度(bit)对512求余的结果不等于448，就需要填充使得对512求余的结果等于448。填充的方法是填充一个1和n个0。填充完后，信息的长度就为N\*512+448(bit)；  
  
     第二步、记录信息长度：用64位来存储填充前信息长度。这64位加在第一步结果的后面，这样信息长度就变为N\*512+448+64=(N+1)\*512位。  
  
     第三步、装入标准的幻数（四个整数）：标准的幻数（物理顺序）是（A=(01234567)16，B=(89ABCDEF)16，C=(FEDCBA98)16，D=(76543210)16）。如果在程序中定义应该是（A=0X67452301L，B=0XEFCDAB89L，C=0X98BADCFEL，D=0X10325476L）。有点晕哈，其实想一想就明白了。  
  
     第四步、四轮循环运算：循环的次数是分组的个数（N+1）   
  
     1）将每一512字节细分成16个小组，每个小组64位（8个字节）  
       
     2）先认识四个线性函数(&是与,|是或,~是非,^是异或)

  F(X,Y,Z)=(X&Y)|((~X)&Z)  
  G(X,Y,Z)=(X&Z)|(Y&(~Z))  
  H(X,Y,Z)=X^Y^Z  
  I(X,Y,Z)=Y^(X|(~Z))

    3）设Mj表示消息的第j个子分组（从0到15），<<<s表示循环左移s位，则四种操作为：

  FF(a,b,c,d,Mj,s,ti)表示a=b+((a+F(b,c,d)+Mj+ti)<<<s)  
  GG(a,b,c,d,Mj,s,ti)表示a=b+((a+G(b,c,d)+Mj+ti)<<<s)  
  HH(a,b,c,d,Mj,s,ti)表示a=b+((a+H(b,c,d)+Mj+ti)<<<s)  
  II(a,b,c,d,Mj,s,ti)表示a=b+((a+I(b,c,d)+Mj+ti)<<<s)

    4）四轮运算

**第一轮**  
a=FF(a,b,c,d,M0,7,0xd76aa478)  
b=FF(d,a,b,c,M1,12,0xe8c7b756)  
c=FF(c,d,a,b,M2,17,0x242070db)  
d=FF(b,c,d,a,M3,22,0xc1bdceee)  
a=FF(a,b,c,d,M4,7,0xf57c0faf)  
b=FF(d,a,b,c,M5,12,0x4787c62a)  
c=FF(c,d,a,b,M6,17,0xa8304613)  
d=FF(b,c,d,a,M7,22,0xfd469501)  
a=FF(a,b,c,d,M8,7,0x698098d8)  
b=FF(d,a,b,c,M9,12,0x8b44f7af)  
c=FF(c,d,a,b,M10,17,0xffff5bb1)  
d=FF(b,c,d,a,M11,22,0x895cd7be)  
a=FF(a,b,c,d,M12,7,0x6b901122)  
b=FF(d,a,b,c,M13,12,0xfd987193)  
c=FF(c,d,a,b,M14,17,0xa679438e)  
d=FF(b,c,d,a,M15,22,0x49b40821)  
  
**第二轮**  
a=GG(a,b,c,d,M1,5,0xf61e2562)  
b=GG(d,a,b,c,M6,9,0xc040b340)  
c=GG(c,d,a,b,M11,14,0x265e5a51)  
d=GG(b,c,d,a,M0,20,0xe9b6c7aa)  
a=GG(a,b,c,d,M5,5,0xd62f105d)  
b=GG(d,a,b,c,M10,9,0x02441453)  
c=GG(c,d,a,b,M15,14,0xd8a1e681)  
d=GG(b,c,d,a,M4,20,0xe7d3fbc8)  
a=GG(a,b,c,d,M9,5,0x21e1cde6)  
b=GG(d,a,b,c,M14,9,0xc33707d6)  
c=GG(c,d,a,b,M3,14,0xf4d50d87)  
d=GG(b,c,d,a,M8,20,0x455a14ed)  
a=GG(a,b,c,d,M13,5,0xa9e3e905)  
b=GG(d,a,b,c,M2,9,0xfcefa3f8)  
c=GG(c,d,a,b,M7,14,0x676f02d9)  
d=GG(b,c,d,a,M12,20,0x8d2a4c8a)  
  
**第三轮**  
a=HH(a,b,c,d,M5,4,0xfffa3942)  
b=HH(d,a,b,c,M8,11,0x8771f681)  
c=HH(c,d,a,b,M11,16,0x6d9d6122)  
d=HH(b,c,d,a,M14,23,0xfde5380c)  
a=HH(a,b,c,d,M1,4,0xa4beea44)  
b=HH(d,a,b,c,M4,11,0x4bdecfa9)  
c=HH(c,d,a,b,M7,16,0xf6bb4b60)  
d=HH(b,c,d,a,M10,23,0xbebfbc70)  
a=HH(a,b,c,d,M13,4,0x289b7ec6)  
b=HH(d,a,b,c,M0,11,0xeaa127fa)  
c=HH(c,d,a,b,M3,16,0xd4ef3085)  
d=HH(b,c,d,a,M6,23,0x04881d05)  
a=HH(a,b,c,d,M9,4,0xd9d4d039)  
b=HH(d,a,b,c,M12,11,0xe6db99e5)  
c=HH(c,d,a,b,M15,16,0x1fa27cf8)  
d=HH(b,c,d,a,M2,23,0xc4ac5665)  
  
**第四轮**  
a=II(a,b,c,d,M0,6,0xf4292244)  
b=II(d,a,b,c,M7,10,0x432aff97)  
c=II(c,d,a,b,M14,15,0xab9423a7)  
d=II(b,c,d,a,M5,21,0xfc93a039)  
a=II(a,b,c,d,M12,6,0x655b59c3)  
b=II(d,a,b,c,M3,10,0x8f0ccc92)  
c=II(c,d,a,b,M10,15,0xffeff47d)  
d=II(b,c,d,a,M1,21,0x85845dd1)  
a=II(a,b,c,d,M8,6,0x6fa87e4f)  
b=II(d,a,b,c,M15,10,0xfe2ce6e0)  
c=II(c,d,a,b,M6,15,0xa3014314)  
d=II(b,c,d,a,M13,21,0x4e0811a1)  
a=II(a,b,c,d,M4,6,0xf7537e82)  
b=II(d,a,b,c,M11,10,0xbd3af235)  
c=II(c,d,a,b,M2,15,0x2ad7d2bb)  
d=II(b,c,d,a,M9,21,0xeb86d391)

    5）每轮循环后，将A，B，C，D分别加上a，b，c，d，然后进入下一循环。  
   
    如果上面的过程用JAVA代码来实现的话，代码如下：

IMG_256package woxingwosu;  
IMG_257  
IMG_258  
IMG_259/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
IMG_260 \* MD5 算法  
IMG_261 \* @author 我行我素  
IMG_262 \* @Date 2007-07-01  
IMG_263\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
IMG_264public class MD5 {  
IMG_265  
IMG_266    static final String hexs[]={"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"};  
IMG_267    //标准的幻数  
IMG_268    private static final long A=0x67452301L;  
IMG_269    private static final long B=0xefcdab89L;  
IMG_270    private static final long C=0x98badcfeL;  
IMG_271    private static final long D=0x10325476L;  
IMG_272  
IMG_273      
IMG_274    //下面这些S11-S44实际上是一个4\*4的矩阵，在四轮循环运算中用到  
IMG_275    static final int S11 = 7;  
IMG_276    static final int S12 = 12;  
IMG_277    static final int S13 = 17;  
IMG_278    static final int S14 = 22;  
IMG_279  
IMG_280    static final int S21 = 5;  
IMG_281    static final int S22 = 9;  
IMG_282    static final int S23 = 14;  
IMG_283    static final int S24 = 20;  
IMG_284  
IMG_285    static final int S31 = 4;  
IMG_286    static final int S32 = 11;  
IMG_287    static final int S33 = 16;  
IMG_288    static final int S34 = 23;  
IMG_289  
IMG_290    static final int S41 = 6;  
IMG_291    static final int S42 = 10;  
IMG_292    static final int S43 = 15;  
IMG_293    static final int S44 = 21;  
IMG_294      
IMG_295    //java不支持无符号的基本数据（unsigned）  
IMG_296    private long [] result={A,B,C,D};//存储hash结果，共4×32=128位，初始化值为（幻数的级联）  
IMG_297      
IMG_298    public static void main(String []args){  
IMG_299        MD5 md=new MD5();  
IMG_300        System.out.println("md5(abc)="+md.digest("abc"));  
IMG_301    }  
IMG_302      
IMG_303    private String digest(String inputStr){  
IMG_304        byte [] inputBytes=inputStr.getBytes();  
IMG_305        int byteLen=inputBytes.length;//长度（字节）  
IMG_306        int groupCount=0;//完整分组的个数  
IMG_307        groupCount=byteLen/64;//每组512位（64字节）  
IMG_308        long []groups=null;//每个小组(64字节)再细分后的16个小组(4字节)  
IMG_309          
IMG_310        //处理每一个完整 分组  
IMG_311        for(int step=0;step<groupCount;step++){  
IMG_312            groups=divGroup(inputBytes,step\*64);  
IMG_313            trans(groups);//处理分组，核心算法  
IMG_314        }  
IMG_315          
IMG_316        //处理完整分组后的尾巴  
IMG_317        int rest=byteLen%64;//512位分组后的余数  
IMG_318        byte [] tempBytes=new byte[64];  
IMG_319        if(rest<=56){  
IMG_320            for(int i=0;i<rest;i++)  
IMG_321                tempBytes[i]=inputBytes[byteLen-rest+i];  
IMG_322            if(rest<56){  
IMG_323                tempBytes[rest]=(byte)(1<<7);  
IMG_324                for(int i=1;i<56-rest;i++)  
IMG_325                    tempBytes[rest+i]=0;  
IMG_326            }  
IMG_327            long len=(long)(byteLen<<3);  
IMG_328            for(int i=0;i<8;i++){  
IMG_329                tempBytes[56+i]=(byte)(len&0xFFL);  
IMG_330                len=len>>8;  
IMG_331            }  
IMG_332            groups=divGroup(tempBytes,0);  
IMG_333            trans(groups);//处理分组  
IMG_334        }else{  
IMG_335            for(int i=0;i<rest;i++)  
IMG_336                tempBytes[i]=inputBytes[byteLen-rest+i];  
IMG_337            tempBytes[rest]=(byte)(1<<7);  
IMG_338            for(int i=rest+1;i<64;i++)  
IMG_339                tempBytes[i]=0;  
IMG_340            groups=divGroup(tempBytes,0);  
IMG_341            trans(groups);//处理分组  
IMG_342              
IMG_343            for(int i=0;i<56;i++)  
IMG_344                tempBytes[i]=0;  
IMG_345            long len=(long)(byteLen<<3);  
IMG_346            for(int i=0;i<8;i++){  
IMG_347                tempBytes[56+i]=(byte)(len&0xFFL);  
IMG_348                len=len>>8;  
IMG_349            }  
IMG_350            groups=divGroup(tempBytes,0);  
IMG_351            trans(groups);//处理分组  
IMG_352        }  
IMG_353          
IMG_354        //将Hash值转换成十六进制的字符串  
IMG_355        String resStr="";  
IMG_356        long temp=0;  
IMG_357        for(int i=0;i<4;i++){  
IMG_358            for(int j=0;j<4;j++){  
IMG_359                temp=result[i]&0x0FL;  
IMG_360                String a=hexs[(int)(temp)];  
IMG_361                result[i]=result[i]>>4;  
IMG_362                temp=result[i]&0x0FL;  
IMG_363                resStr+=hexs[(int)(temp)]+a;  
IMG_364                result[i]=result[i]>>4;  
IMG_365            }  
IMG_366        }  
IMG_367        return resStr;  
IMG_368    }  
IMG_369      
IMG_370    /\*\*  
IMG_371     \* 从inputBytes的index开始取512位，作为新的分组  
IMG_372     \* 将每一个512位的分组再细分成16个小组，每个小组64位（8个字节）  
IMG_373     \* @param inputBytes  
IMG_374     \* @param index  
IMG_375     \* @return  
IMG_376     \*/  
IMG_377    private static long[] divGroup(byte[] inputBytes,int index){  
IMG_378        long [] temp=new long[16];  
IMG_379        for(int i=0;i<16;i++){  
IMG_380            temp[i]=b2iu(inputBytes[4\*i+index])|  
IMG_381                (b2iu(inputBytes[4\*i+1+index]))<<8|  
IMG_382                (b2iu(inputBytes[4\*i+2+index]))<<16|  
IMG_383                (b2iu(inputBytes[4\*i+3+index]))<<24;  
IMG_384        }  
IMG_385        return temp;  
IMG_386    }  
IMG_387      
IMG_388    /\*\*  
IMG_389     \* 这时不存在符号位（符号位存储不再是代表正负），所以需要处理一下  
IMG_390     \* @param b  
IMG_391     \* @return  
IMG_392     \*/  
IMG_393    public static long b2iu(byte b){  
IMG_394        return b < 0 ? b & 0x7F + 128 : b;  
IMG_395     }  
IMG_396      
IMG_397    /\*\*  
IMG_398     \* 主要的操作，四轮循环  
IMG_399     \* @param groups[]--每一个分组512位（64字节）  
IMG_400     \*/  
IMG_401    private void trans(long[] groups) {  
IMG_402        long a = result[0], b = result[1], c = result[2], d = result[3];  
IMG_403        /\*第一轮\*/  
IMG_404        a = FF(a, b, c, d, groups[0], S11, 0xd76aa478L); /\* 1 \*/  
IMG_405        d = FF(d, a, b, c, groups[1], S12, 0xe8c7b756L); /\* 2 \*/  
IMG_406        c = FF(c, d, a, b, groups[2], S13, 0x242070dbL); /\* 3 \*/  
IMG_407        b = FF(b, c, d, a, groups[3], S14, 0xc1bdceeeL); /\* 4 \*/  
IMG_408        a = FF(a, b, c, d, groups[4], S11, 0xf57c0fafL); /\* 5 \*/  
IMG_409        d = FF(d, a, b, c, groups[5], S12, 0x4787c62aL); /\* 6 \*/  
IMG_410        c = FF(c, d, a, b, groups[6], S13, 0xa8304613L); /\* 7 \*/  
IMG_411        b = FF(b, c, d, a, groups[7], S14, 0xfd469501L); /\* 8 \*/  
IMG_412        a = FF(a, b, c, d, groups[8], S11, 0x698098d8L); /\* 9 \*/  
IMG_413        d = FF(d, a, b, c, groups[9], S12, 0x8b44f7afL); /\* 10 \*/  
IMG_414        c = FF(c, d, a, b, groups[10], S13, 0xffff5bb1L); /\* 11 \*/  
IMG_415        b = FF(b, c, d, a, groups[11], S14, 0x895cd7beL); /\* 12 \*/  
IMG_416        a = FF(a, b, c, d, groups[12], S11, 0x6b901122L); /\* 13 \*/  
IMG_417        d = FF(d, a, b, c, groups[13], S12, 0xfd987193L); /\* 14 \*/  
IMG_418        c = FF(c, d, a, b, groups[14], S13, 0xa679438eL); /\* 15 \*/  
IMG_419        b = FF(b, c, d, a, groups[15], S14, 0x49b40821L); /\* 16 \*/  
IMG_420  
IMG_421        /\*第二轮\*/  
IMG_422        a = GG(a, b, c, d, groups[1], S21, 0xf61e2562L); /\* 17 \*/  
IMG_423        d = GG(d, a, b, c, groups[6], S22, 0xc040b340L); /\* 18 \*/  
IMG_424        c = GG(c, d, a, b, groups[11], S23, 0x265e5a51L); /\* 19 \*/  
IMG_425        b = GG(b, c, d, a, groups[0], S24, 0xe9b6c7aaL); /\* 20 \*/  
IMG_426        a = GG(a, b, c, d, groups[5], S21, 0xd62f105dL); /\* 21 \*/  
IMG_427        d = GG(d, a, b, c, groups[10], S22, 0x2441453L); /\* 22 \*/  
IMG_428        c = GG(c, d, a, b, groups[15], S23, 0xd8a1e681L); /\* 23 \*/  
IMG_429        b = GG(b, c, d, a, groups[4], S24, 0xe7d3fbc8L); /\* 24 \*/  
IMG_430        a = GG(a, b, c, d, groups[9], S21, 0x21e1cde6L); /\* 25 \*/  
IMG_431        d = GG(d, a, b, c, groups[14], S22, 0xc33707d6L); /\* 26 \*/  
IMG_432        c = GG(c, d, a, b, groups[3], S23, 0xf4d50d87L); /\* 27 \*/  
IMG_433        b = GG(b, c, d, a, groups[8], S24, 0x455a14edL); /\* 28 \*/  
IMG_434        a = GG(a, b, c, d, groups[13], S21, 0xa9e3e905L); /\* 29 \*/  
IMG_435        d = GG(d, a, b, c, groups[2], S22, 0xfcefa3f8L); /\* 30 \*/  
IMG_436        c = GG(c, d, a, b, groups[7], S23, 0x676f02d9L); /\* 31 \*/  
IMG_437        b = GG(b, c, d, a, groups[12], S24, 0x8d2a4c8aL); /\* 32 \*/  
IMG_438  
IMG_439        /\*第三轮\*/  
IMG_440        a = HH(a, b, c, d, groups[5], S31, 0xfffa3942L); /\* 33 \*/  
IMG_441        d = HH(d, a, b, c, groups[8], S32, 0x8771f681L); /\* 34 \*/  
IMG_442        c = HH(c, d, a, b, groups[11], S33, 0x6d9d6122L); /\* 35 \*/  
IMG_443        b = HH(b, c, d, a, groups[14], S34, 0xfde5380cL); /\* 36 \*/  
IMG_444        a = HH(a, b, c, d, groups[1], S31, 0xa4beea44L); /\* 37 \*/  
IMG_445        d = HH(d, a, b, c, groups[4], S32, 0x4bdecfa9L); /\* 38 \*/  
IMG_446        c = HH(c, d, a, b, groups[7], S33, 0xf6bb4b60L); /\* 39 \*/  
IMG_447        b = HH(b, c, d, a, groups[10], S34, 0xbebfbc70L); /\* 40 \*/  
IMG_448        a = HH(a, b, c, d, groups[13], S31, 0x289b7ec6L); /\* 41 \*/  
IMG_449        d = HH(d, a, b, c, groups[0], S32, 0xeaa127faL); /\* 42 \*/  
IMG_450        c = HH(c, d, a, b, groups[3], S33, 0xd4ef3085L); /\* 43 \*/  
IMG_451        b = HH(b, c, d, a, groups[6], S34, 0x4881d05L); /\* 44 \*/  
IMG_452        a = HH(a, b, c, d, groups[9], S31, 0xd9d4d039L); /\* 45 \*/  
IMG_453        d = HH(d, a, b, c, groups[12], S32, 0xe6db99e5L); /\* 46 \*/  
IMG_454        c = HH(c, d, a, b, groups[15], S33, 0x1fa27cf8L); /\* 47 \*/  
IMG_455        b = HH(b, c, d, a, groups[2], S34, 0xc4ac5665L); /\* 48 \*/  
IMG_456  
IMG_457        /\*第四轮\*/  
IMG_458        a = II(a, b, c, d, groups[0], S41, 0xf4292244L); /\* 49 \*/  
IMG_459        d = II(d, a, b, c, groups[7], S42, 0x432aff97L); /\* 50 \*/  
IMG_460        c = II(c, d, a, b, groups[14], S43, 0xab9423a7L); /\* 51 \*/  
IMG_461        b = II(b, c, d, a, groups[5], S44, 0xfc93a039L); /\* 52 \*/  
IMG_462        a = II(a, b, c, d, groups[12], S41, 0x655b59c3L); /\* 53 \*/  
IMG_463        d = II(d, a, b, c, groups[3], S42, 0x8f0ccc92L); /\* 54 \*/  
IMG_464        c = II(c, d, a, b, groups[10], S43, 0xffeff47dL); /\* 55 \*/  
IMG_465        b = II(b, c, d, a, groups[1], S44, 0x85845dd1L); /\* 56 \*/  
IMG_466        a = II(a, b, c, d, groups[8], S41, 0x6fa87e4fL); /\* 57 \*/  
IMG_467        d = II(d, a, b, c, groups[15], S42, 0xfe2ce6e0L); /\* 58 \*/  
IMG_468        c = II(c, d, a, b, groups[6], S43, 0xa3014314L); /\* 59 \*/  
IMG_469        b = II(b, c, d, a, groups[13], S44, 0x4e0811a1L); /\* 60 \*/  
IMG_470        a = II(a, b, c, d, groups[4], S41, 0xf7537e82L); /\* 61 \*/  
IMG_471        d = II(d, a, b, c, groups[11], S42, 0xbd3af235L); /\* 62 \*/  
IMG_472        c = II(c, d, a, b, groups[2], S43, 0x2ad7d2bbL); /\* 63 \*/  
IMG_473        b = II(b, c, d, a, groups[9], S44, 0xeb86d391L); /\* 64 \*/  
IMG_474  
IMG_475        /\*加入到之前计算的结果当中\*/  
IMG_476        result[0] += a;  
IMG_477        result[1] += b;  
IMG_478        result[2] += c;  
IMG_479        result[3] += d;  
IMG_480        result[0]=result[0]&0xFFFFFFFFL;  
IMG_481        result[1]=result[1]&0xFFFFFFFFL;  
IMG_482        result[2]=result[2]&0xFFFFFFFFL;  
IMG_483        result[3]=result[3]&0xFFFFFFFFL;  
IMG_484    }  
IMG_485      
IMG_486    /\*\*  
IMG_487     \* 下面是处理要用到的线性函数  
IMG_488     \*/  
IMG_489    private static long F(long x, long y, long z) {  
IMG_490        return (x & y) | ((~x) & z);  
IMG_491    }  
IMG_492  
IMG_493    private static long G(long x, long y, long z) {  
IMG_494        return (x & z) | (y & (~z));  
IMG_495    }  
IMG_496  
IMG_497    private static long H(long x, long y, long z) {  
IMG_498        return x ^ y ^ z;  
IMG_499    }  
IMG_500  
IMG_501    private static long I(long x, long y, long z) {  
IMG_502        return y ^ (x | (~z));  
IMG_503    }  
IMG_504  
IMG_505    private static long FF(long a, long b, long c, long d, long x, long s,  
IMG_506            long ac) {  
IMG_507        a += (F(b, c, d)&0xFFFFFFFFL) + x + ac;  
IMG_508        a = ((a&0xFFFFFFFFL)<< s) | ((a&0xFFFFFFFFL) >>> (32 - s));  
IMG_509        a += b;  
IMG_510        return (a&0xFFFFFFFFL);  
IMG_511    }  
IMG_512  
IMG_513    private static long GG(long a, long b, long c, long d, long x, long s,  
IMG_514            long ac) {  
IMG_515        a += (G(b, c, d)&0xFFFFFFFFL) + x + ac;  
IMG_516        a = ((a&0xFFFFFFFFL) << s) | ((a&0xFFFFFFFFL) >>> (32 - s));  
IMG_517        a += b;  
IMG_518        return (a&0xFFFFFFFFL);  
IMG_519    }  
IMG_520  
IMG_521    private static long HH(long a, long b, long c, long d, long x, long s,  
IMG_522            long ac) {  
IMG_523        a += (H(b, c, d)&0xFFFFFFFFL) + x + ac;  
IMG_524        a = ((a&0xFFFFFFFFL) << s) | ((a&0xFFFFFFFFL) >>> (32 - s));  
IMG_525        a += b;  
IMG_526        return (a&0xFFFFFFFFL);  
IMG_527    }  
IMG_528  
IMG_529    private static long II(long a, long b, long c, long d, long x, long s,  
IMG_530            long ac) {  
IMG_531        a += (I(b, c, d)&0xFFFFFFFFL) + x + ac;  
IMG_532        a = ((a&0xFFFFFFFFL) << s) | ((a&0xFFFFFFFFL) >>> (32 - s));  
IMG_533        a += b;  
IMG_534        return (a&0xFFFFFFFFL);  
IMG_535    }  
IMG_536}  
IMG_537

MD5安全性：  
    普遍认为MD5是很安全，因为暴力破解的时间是一般人无法接受的。实际上如果把用户的密码MD5处理后再存储到数据库，其实是很不安全的。因为用户的密码是比较短的，而且很多用户的密码都使用生日，手机号码，身份证号码，电话号码等等。或者使用常用的一些吉利的数字，或者某个英文单词。如果我把常用的密码先MD5处理，把数据存储起来，然后再跟你的MD5结果匹配，这时我就有可能得到明文。比如某个MD5破解网站<http://www.cmd5.com/default.aspx>，我把其网站下的公告复制如下

  md5破解、动网论坛密码破解等不再需要用穷举法，本站共有md5记录235亿条，还在不断增长中，已包含10位及10位以下数字、7位字母、部分7位字母＋数字,全部6位及以下字母加数字等组合，并针对国内用户做了大量优化，例如已经包含所有手机号码、全国部分大中城市固定电话号码、百家姓、常用拼音等大量组合，另加入了某大型网站真实会员密码数据10万条。本站数据量大，查询速度快，同时支持16位及32位密码查询。通过对10万会员的真实动网论坛样本数据的测试，本站对于动网论坛密码的命中率达到83%。  
  本站4T的硬盘已经上线，正在生成数据，预计需要2个月左右时间，到时候本站能查询到12位数字和9位字母。

   你可以用你的生日，手机号码去测试一下。  
   我觉得只需要将上面我写的MD5的标准幻数A,B,C,D的值修改一下，修改后也不是MD5算法了，因为不能保证唯一性。这样就算别人得到32位的值，他如果不知道幻数的值是无法还原明文的。就算得到了幻数，也是很难破解的。  
  
     JAVA实现MD5  
    在java中实现MD5是很简单的，在包java.security有个类MessageDigest。官方文档如下

    MessageDigest 类为应用程序提供信息摘要算法的功能，如 MD5 或 SHA 算法。信息摘要是安全的单向哈希函数，它接收任意大小的数据，输出固定长度的哈希值。   
  
    MessageDigest 对象开始被初始化。该对象通过使用 update 方法处理数据。任何时候都可以调用 reset 方法重置摘要。一旦所有需要更新的数据都已经被更新了，应该调用 digest 方法之一完成哈希计算。   
  
    对于给定数量的更新数据，digest 方法只能被调用一次。digest 被调用后，MessageDigest 对象被重新设置成其初始状态。

   使用MessageDigest很简单，例如

IMG_538package woxingwosu;  
IMG_539/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
IMG_540 \* MD5 算法  
IMG_541 \* @author 我行我素  
IMG_542 \* @Date 2007-07-06  
IMG_543\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
IMG_544import java.security.MessageDigest;  
IMG_545  
IMG_546public class MyMD5 {  
IMG_547  
IMG_548    static char[] hex = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};   
IMG_549    public static void main(String[] args) {  
IMG_550        try{  
IMG_551            MessageDigest md5 = MessageDigest.getInstance("MD5");//申明使用MD5算法  
IMG_552            md5.update("a".getBytes());//  
IMG_553            System.out.println("md5(a)="+byte2str(md5.digest()));  
IMG_554            md5.update("a".getBytes());  
IMG_555            md5.update("bc".getBytes());  
IMG_556            System.out.println("md5(abc)="+byte2str(md5.digest()));  
IMG_557        }catch(Exception e){  
IMG_558            e.printStackTrace();  
IMG_559        }  
IMG_560    }  
IMG_561      
IMG_562    /\*\*  
IMG_563     \* 将字节数组转换成十六进制字符串  
IMG_564     \* @param bytes  
IMG_565     \* @return  
IMG_566     \*/  
IMG_567    private static String byte2str(byte []bytes){  
IMG_568        int len = bytes.length;     
IMG_569        StringBuffer result = new StringBuffer();      
IMG_570        for (int i = 0; i < len; i++) {     
IMG_571            byte byte0 = bytes[i];     
IMG_572            result.append(hex[byte0 >>> 4 & 0xf]);     
IMG_573            result.append(hex[byte0 & 0xf]);     
IMG_574        }  
IMG_575        return result.toString();  
IMG_576    }  
IMG_577}   
IMG_578

  另外附上其他版本的MD5算法的实现(来自网络)  
    1)JS版的MD5(调用方法：md5(明文))

IMG_579/\*  
IMG_580 \* A JavaScript implementation of the RSA Data Security, Inc. MD5 Message  
IMG_581 \* Digest Algorithm, as defined in RFC 1321.  
IMG_582 \* Version 2.1 Copyright (C) Paul Johnston 1999 - 2002.  
IMG_583 \* Other contributors: Greg Holt, Andrew Kepert, Ydnar, Lostinet  
IMG_584 \* Distributed under the BSD License  
IMG_585 \* See http://pajhome.org.uk/crypt/md5 for more info.  
IMG_586 \*/  
IMG_587  
IMG_588/\*  
IMG_589 \* Configurable variables. You may need to tweak these to be compatible with  
IMG_590 \* the server-side, but the defaults work in most cases.  
IMG_591 \*/  
IMG_592var hexcase = 1;  /\* hex output format. 0 - lowercase; 1 - uppercase        \*/  
IMG_593var b64pad  = ""; /\* base-64 pad character. "=" for strict RFC compliance   \*/  
IMG_594var chrsz   = 8;  /\* bits per input character. 8 - ASCII; 16 - Unicode      \*/  
IMG_595  
IMG_596/\*  
IMG_597 \* These are the functions you'll usually want to call  
IMG_598 \* They take string arguments and return either hex or base-64 encoded strings  
IMG_599 \*/  
IMG_600function hex\_md5(s){ return binl2hex(core\_md5(str2binl(s), s.length \* chrsz));}  
IMG_601function str\_md5(s){ return binl2str(core\_md5(str2binl(s), s.length \* chrsz));}  
IMG_602function hex\_hmac\_md5(key, data) { return binl2hex(core\_hmac\_md5(key, data)); }  
IMG_603function str\_hmac\_md5(key, data) { return binl2str(core\_hmac\_md5(key, data)); }  
IMG_604  
IMG_605/\*  
IMG_606 \* Calculate the MD5 of an array of little-endian words, and a bit length  
IMG_607 \*/  
IMG_608function core\_md5(x, len)  
IMG_609{  
IMG_610  /\* append padding \*/  
IMG_611  x[len >> 5] |= 0x80 << ((len) % 32);  
IMG_612  x[(((len + 64) >>> 9) << 4) + 14] = len;  
IMG_613  
IMG_614  var a =  1732584193;  
IMG_615  var b = -271733879;  
IMG_616  var c = -1732584194;  
IMG_617  var d =  271733878;  
IMG_618  
IMG_619  for(var i = 0; i < x.length; i += 16)  
IMG_620  {  
IMG_621    var olda = a;  
IMG_622    var oldb = b;  
IMG_623    var oldc = c;  
IMG_624    var oldd = d;  
IMG_625  
IMG_626    a = md5\_ff(a, b, c, d, x[i+ 0], 7 , -680876936);  
IMG_627    d = md5\_ff(d, a, b, c, x[i+ 1], 12, -389564586);  
IMG_628    c = md5\_ff(c, d, a, b, x[i+ 2], 17,  606105819);  
IMG_629    b = md5\_ff(b, c, d, a, x[i+ 3], 22, -1044525330);  
IMG_630    a = md5\_ff(a, b, c, d, x[i+ 4], 7 , -176418897);  
IMG_631    d = md5\_ff(d, a, b, c, x[i+ 5], 12,  1200080426);  
IMG_632    c = md5\_ff(c, d, a, b, x[i+ 6], 17, -1473231341);  
IMG_633    b = md5\_ff(b, c, d, a, x[i+ 7], 22, -45705983);  
IMG_634    a = md5\_ff(a, b, c, d, x[i+ 8], 7 ,  1770035416);  
IMG_635    d = md5\_ff(d, a, b, c, x[i+ 9], 12, -1958414417);  
IMG_636    c = md5\_ff(c, d, a, b, x[i+10], 17, -42063);  
IMG_637    b = md5\_ff(b, c, d, a, x[i+11], 22, -1990404162);  
IMG_638    a = md5\_ff(a, b, c, d, x[i+12], 7 ,  1804603682);  
IMG_639    d = md5\_ff(d, a, b, c, x[i+13], 12, -40341101);  
IMG_640    c = md5\_ff(c, d, a, b, x[i+14], 17, -1502002290);  
IMG_641    b = md5\_ff(b, c, d, a, x[i+15], 22,  1236535329);  
IMG_642  
IMG_643    a = md5\_gg(a, b, c, d, x[i+ 1], 5 , -165796510);  
IMG_644    d = md5\_gg(d, a, b, c, x[i+ 6], 9 , -1069501632);  
IMG_645    c = md5\_gg(c, d, a, b, x[i+11], 14,  643717713);  
IMG_646    b = md5\_gg(b, c, d, a, x[i+ 0], 20, -373897302);  
IMG_647    a = md5\_gg(a, b, c, d, x[i+ 5], 5 , -701558691);  
IMG_648    d = md5\_gg(d, a, b, c, x[i+10], 9 ,  38016083);  
IMG_649    c = md5\_gg(c, d, a, b, x[i+15], 14, -660478335);  
IMG_650    b = md5\_gg(b, c, d, a, x[i+ 4], 20, -405537848);  
IMG_651    a = md5\_gg(a, b, c, d, x[i+ 9], 5 ,  568446438);  
IMG_652    d = md5\_gg(d, a, b, c, x[i+14], 9 , -1019803690);  
IMG_653    c = md5\_gg(c, d, a, b, x[i+ 3], 14, -187363961);  
IMG_654    b = md5\_gg(b, c, d, a, x[i+ 8], 20,  1163531501);  
IMG_655    a = md5\_gg(a, b, c, d, x[i+13], 5 , -1444681467);  
IMG_656    d = md5\_gg(d, a, b, c, x[i+ 2], 9 , -51403784);  
IMG_657    c = md5\_gg(c, d, a, b, x[i+ 7], 14,  1735328473);  
IMG_658    b = md5\_gg(b, c, d, a, x[i+12], 20, -1926607734);  
IMG_659  
IMG_660    a = md5\_hh(a, b, c, d, x[i+ 5], 4 , -378558);  
IMG_661    d = md5\_hh(d, a, b, c, x[i+ 8], 11, -2022574463);  
IMG_662    c = md5\_hh(c, d, a, b, x[i+11], 16,  1839030562);  
IMG_663    b = md5\_hh(b, c, d, a, x[i+14], 23, -35309556);  
IMG_664    a = md5\_hh(a, b, c, d, x[i+ 1], 4 , -1530992060);  
IMG_665    d = md5\_hh(d, a, b, c, x[i+ 4], 11,  1272893353);  
IMG_666    c = md5\_hh(c, d, a, b, x[i+ 7], 16, -155497632);  
IMG_667    b = md5\_hh(b, c, d, a, x[i+10], 23, -1094730640);  
IMG_668    a = md5\_hh(a, b, c, d, x[i+13], 4 ,  681279174);  
IMG_669    d = md5\_hh(d, a, b, c, x[i+ 0], 11, -358537222);  
IMG_670    c = md5\_hh(c, d, a, b, x[i+ 3], 16, -722521979);  
IMG_671    b = md5\_hh(b, c, d, a, x[i+ 6], 23,  76029189);  
IMG_672    a = md5\_hh(a, b, c, d, x[i+ 9], 4 , -640364487);  
IMG_673    d = md5\_hh(d, a, b, c, x[i+12], 11, -421815835);  
IMG_674    c = md5\_hh(c, d, a, b, x[i+15], 16,  530742520);  
IMG_675    b = md5\_hh(b, c, d, a, x[i+ 2], 23, -995338651);  
IMG_676  
IMG_677    a = md5\_ii(a, b, c, d, x[i+ 0], 6 , -198630844);  
IMG_678    d = md5\_ii(d, a, b, c, x[i+ 7], 10,  1126891415);  
IMG_679    c = md5\_ii(c, d, a, b, x[i+14], 15, -1416354905);  
IMG_680    b = md5\_ii(b, c, d, a, x[i+ 5], 21, -57434055);  
IMG_681    a = md5\_ii(a, b, c, d, x[i+12], 6 ,  1700485571);  
IMG_682    d = md5\_ii(d, a, b, c, x[i+ 3], 10, -1894986606);  
IMG_683    c = md5\_ii(c, d, a, b, x[i+10], 15, -1051523);  
IMG_684    b = md5\_ii(b, c, d, a, x[i+ 1], 21, -2054922799);  
IMG_685    a = md5\_ii(a, b, c, d, x[i+ 8], 6 ,  1873313359);  
IMG_686    d = md5\_ii(d, a, b, c, x[i+15], 10, -30611744);  
IMG_687    c = md5\_ii(c, d, a, b, x[i+ 6], 15, -1560198380);  
IMG_688    b = md5\_ii(b, c, d, a, x[i+13], 21,  1309151649);  
IMG_689    a = md5\_ii(a, b, c, d, x[i+ 4], 6 , -145523070);  
IMG_690    d = md5\_ii(d, a, b, c, x[i+11], 10, -1120210379);  
IMG_691    c = md5\_ii(c, d, a, b, x[i+ 2], 15,  718787259);  
IMG_692    b = md5\_ii(b, c, d, a, x[i+ 9], 21, -343485551);  
IMG_693  
IMG_694    a = safe\_add(a, olda);  
IMG_695    b = safe\_add(b, oldb);  
IMG_696    c = safe\_add(c, oldc);  
IMG_697    d = safe\_add(d, oldd);  
IMG_698  }  
IMG_699  return Array(a, b, c, d);  
IMG_700  
IMG_701}  
IMG_702  
IMG_703/\*  
IMG_704 \* These functions implement the four basic operations the algorithm uses.  
IMG_705 \*/  
IMG_706function md5\_cmn(q, a, b, x, s, t)  
IMG_707{  
IMG_708  return safe\_add(bit\_rol(safe\_add(safe\_add(a, q), safe\_add(x, t)), s),b);  
IMG_709}  
IMG_710function md5\_ff(a, b, c, d, x, s, t)  
IMG_711{  
IMG_712  return md5\_cmn((b & c) | ((~b) & d), a, b, x, s, t);  
IMG_713}  
IMG_714function md5\_gg(a, b, c, d, x, s, t)  
IMG_715{  
IMG_716  return md5\_cmn((b & d) | (c & (~d)), a, b, x, s, t);  
IMG_717}  
IMG_718function md5\_hh(a, b, c, d, x, s, t)  
IMG_719{  
IMG_720  return md5\_cmn(b ^ c ^ d, a, b, x, s, t);  
IMG_721}  
IMG_722function md5\_ii(a, b, c, d, x, s, t)  
IMG_723{  
IMG_724  return md5\_cmn(c ^ (b | (~d)), a, b, x, s, t);  
IMG_725}  
IMG_726  
IMG_727/\*  
IMG_728 \* Calculate the HMAC-MD5, of a key and some data  
IMG_729 \*/  
IMG_730function core\_hmac\_md5(key, data)  
IMG_731{  
IMG_732  var bkey = str2binl(key);  
IMG_733  if(bkey.length > 16) bkey = core\_md5(bkey, key.length \* chrsz);  
IMG_734  
IMG_735  var ipad = Array(16), opad = Array(16);  
IMG_736  for(var i = 0; i < 16; i++)  
IMG_737  {  
IMG_738    ipad[i] = bkey[i] ^ 0x36363636;  
IMG_739    opad[i] = bkey[i] ^ 0x5C5C5C5C;  
IMG_740  }  
IMG_741  
IMG_742  var hash = core\_md5(ipad.concat(str2binl(data)), 512 + data.length \* chrsz);  
IMG_743  return core\_md5(opad.concat(hash), 512 + 128);  
IMG_744}  
IMG_745  
IMG_746/\*  
IMG_747 \* Add integers, wrapping at 2^32. This uses 16-bit operations internally  
IMG_748 \* to work around bugs in some JS interpreters.  
IMG_749 \*/  
IMG_750function safe\_add(x, y)  
IMG_751{  
IMG_752  var lsw = (x & 0xFFFF) + (y & 0xFFFF);  
IMG_753  var msw = (x >> 16) + (y >> 16) + (lsw >> 16);  
IMG_754  return (msw << 16) | (lsw & 0xFFFF);  
IMG_755}  
IMG_756  
IMG_757/\*  
IMG_758 \* Bitwise rotate a 32-bit number to the left.  
IMG_759 \*/  
IMG_760function bit\_rol(num, cnt)  
IMG_761{  
IMG_762  return (num << cnt) | (num >>> (32 - cnt));  
IMG_763}  
IMG_764  
IMG_765/\*  
IMG_766 \* Convert a string to an array of little-endian words  
IMG_767 \* If chrsz is ASCII, characters >255 have their hi-byte silently ignored.  
IMG_768 \*/  
IMG_769function str2binl(str)  
IMG_770{  
IMG_771  var bin = Array();  
IMG_772  var mask = (1 << chrsz) - 1;  
IMG_773  for(var i = 0; i < str.length \* chrsz; i += chrsz)  
IMG_774    bin[i>>5] |= (str.charCodeAt(i / chrsz) & mask) << (i%32);  
IMG_775  return bin;  
IMG_776}  
IMG_777  
IMG_778/\*  
IMG_779 \* Convert an array of little-endian words to a string  
IMG_780 \*/  
IMG_781function binl2str(bin)  
IMG_782{  
IMG_783  var str = "";  
IMG_784  var mask = (1 << chrsz) - 1;  
IMG_785  for(var i = 0; i < bin.length \* 32; i += chrsz)  
IMG_786    str += String.fromCharCode((bin[i>>5] >>> (i % 32)) & mask);  
IMG_787  return str;  
IMG_788}  
IMG_789  
IMG_790/\*  
IMG_791 \* Convert an array of little-endian words to a hex string.  
IMG_792 \*/  
IMG_793function binl2hex(binarray)  
IMG_794{  
IMG_795  var hex\_tab = hexcase ? "0123456789ABCDEF" : "0123456789abcdef";  
IMG_796  var str = "";  
IMG_797  for(var i = 0; i < binarray.length \* 4; i++)  
IMG_798  {  
IMG_799    str += hex\_tab.charAt((binarray[i>>2] >> ((i%4)\*8+4)) & 0xF) +  
IMG_800           hex\_tab.charAt((binarray[i>>2] >> ((i%4)\*8  )) & 0xF);  
IMG_801  }  
IMG_802  return str;  
IMG_803}  
IMG_804  
IMG_805String.prototype.trim = function(){return this.replace(/(^s\*)|(s\*$)/g, "");}  
IMG_806  
IMG_807function md5(text) {  
IMG_808    return hex\_md5(text.trim());  
IMG_809}

http://blog.csdn.net/forgotaboutgirl/article/details/7258109