MD5 算法实现
实验报告

姓名：宋晓彤

学号：16340192

方向：嵌入式软件与系统

2018.11.25

一、 原理概述

MD5 即 Message-Digest Algorithm 5 (信息-摘要算法 5)，使用 little-endian(小端模式)，输入任意不定长度信息，以 512-bit 进行分组，生成四个 32-bit 数据，最后联合输出固定 128-bit 的信息摘要。

MD5 算法的基本过程为：填充、分块、缓冲区初始化、循环压缩、得出结果。

MD5 不是足够安全的。Hans Dobbertin 在 1996 年找到了两个不同的 512-bit 块，它们 在 MD5 计算下产生相同的 hash 值。至今还没有真正找到两个不同的消息，它们的 MD5 的 hash 值相等。

二、 总体结构与流程

MD5 算法具体流程如下：

1. 填充：
    a) 第一步：将输入的字符串转换成进制表达，用 K 位二进制码表达，当 K 对 512 取余不等于 448 时，要使用 100…0 填充上述二进制码，使 K % 512 = 448
    b) 第二步：为使最后的二进制码长度为 512 的整数，需要填充剩余的 64 位，此时，使用原始字符串转换的进制码的末 64 位进行填充，即对 2^{64} 取模后的结果
2. 分块：将长度为 512 倍数的二进制串，以每 512 位（64byte）为一块的方式分成一大块；将每一块以 32 位（4byte）一组的方式分成 16 组，使用这 16 个 4 字节的数值作为参数参与循环
3. 循环：使用以下函数完成 4 次循环，每次循环包含 16 次操作，每次循环包括 64 次操作，其中
    a) 每次操作时，赋值的对象为缓冲区的 a、b、c、d 四个数值以 adcb 的顺序循环
    b) 4 次循环按顺序分别使用 FF GG HH II 函数各进行 16 次操作

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

其中，

```
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))
```

三、模块分解

根据 MD5 算法的流程，我们将算法的过程分为如下几个功能模块：初始化，填充，分组，循环。

【模块一 初始化】

在 MD5 算法中，需要初始化的主要参数都是循环相关的，其他的模块都是一些基础操作，所以，我们对循环进行主要的分析。

首先，在对于缓冲区的初始化中，我们需要取 ABCD 四个基础值，由于 MD5 采用小端的数据存储方式，所以四个值分别应该初始化为 0x67452301L、0xefcdab89L、 0x98badcfeL、 0x10325476L;

其次，我们发现，在 FF GG HH II 这四个二级函数上，有两个位置的参数，S 和 ti，而这两个参数是固定的数值，所以我们对其进行初始化。（初始化见代码实现）

【模块二 填充】

首先，我们需要将输入的字符串转换为数值类变量，为了便于分组，我们使用 byte 将 string 类分解，此时，我们得到以字节计数的数组，此时，需要进行第一步填充，即把字节数填充为 512/8 * n + 448/8（n 为任意实数）的大小，填充的 byte[]应为以 128 开头、后续接相应个数的 0。

第二步，对字符串转换时得到的最初字节长度进行检验，如果不低于 8 个字节（64 位），则使用末尾 8 个字节进行填充，不够则在前面用相应个数的 0 进行补充。

最后得到 64 的倍数大小个字节数的 byte 数组。

【模块三 分块】

由于在分组时，需要分成一个 512 位的大块和 32 位的小块，32 位为一个 long 的大小，所以我们使用一个 long[][]的存储结构存放，第一级角标为大块的个数，第二级角标为 16 个小组中的组号，此时就将前面的模块得到的数据转换成 long 的二维数组，同时方便我们对块号和组号的查询。

【模块四 循环】

此循环过程中，分为 4 次循环，每个循环有 16 次操作，分别对一个块中的 16 个小组（每组 4 字节）的数据进行操作，同时需要辅助操作的数据还有初始化的 ABCD、s、t。

首先，对 ADCB 四个数值按顺序进行相应的赋值（更改数据）函数的操作，共进行 4*16=64 次，且操作的函数按顺序为 16 次 FF、16 次 GG、16 次 HH、16 次 II，s t 变量均含有 64 个元素，按顺序相应使用即可。

【模块五 收尾】

次循环过程中，我们要对得到的二进制码进行输出，转换为相应的 16 进制表达。

四、 数据设计及算法

【数据设计】

1. 变量声明

a) 缓冲区基础数据：A B C D

b) 循环函数中的 S：4*16 的二维数组，4 为外层循环，16 为内层循环，表示一个循环周期 4*16 次操作时函数中参数 s 的值

c) 循环函数中的 T：长度为 64 的数组，表示个循环周期 4*16 次操作时函数中参数 ti 的值

d) K（初始位大小）P（需填充位大小）allBytes（转换后的字节数组）group[][]（使用块号和组号进行调用的 byte 数据）

2. 函数定义

a) Main 函数->按算法步骤调用函数、最后将得到的结果转换为 32 位 16 进制输出

b) changeToBytes 函数->将当前字符串进行 byte[]的转换，同时完成填充 bytes 的操作，输入一个 string 类型的待加密数据，返回一个用 byte 表示的填充后的 byte 数组

c) changeToLongGroups 函数->将当前的 byte 数组进行划分，64bytes 为一个块，4byte 为一个组，最后形成块数待定，组数为 16 的，数据类型为 long 的二维数组 1

d) recycle 函数->对一块中的 16 组进行循环操作，目的为更改缓冲区中 ABCD 的值，从而得到加密结果

e) 一级循环函数（F G H I）及二级循环函数（FF GG HH II）的简单实现。

f) funcPoint offSign changek 函数（在特殊处理中有介绍）

3. 特殊处理：

a) 在对填充数据的大小进行判断时，我们要注意分为三种情况：位数对 512 取模余数不足 448 时（补充至 448 即可）；余数正好为 448 时（注意：需要补充 512 位）；余数大于 448 时（此时补充的位数为 512+448-n）

```
int P = K % 512;
P = P < 448 ? 448-P : 512-P+448;
```

b) 当对已经填充好的 bytes 数组进行分块时，我们需要注意，byte 的取值范围是-128~127，我们赋值的 128 会记录为负数，或者说，在 byte 中，1 开头的数据为负数，但此时我们并不需要符号位，所以我们应当使用方法将符号位转换为计数位

```
public static long offSigned(byte bytes){
    return bytes < 0 ? bytes & 0x7F + 128 : bytes;
}
```

c) 在实现循环时，如果直接写 64 行代码会显得代码很厚重，此时，我们需要改变这部分代码的撰写方式：

i. 首先，4 次外循环中，每一次的函数是不一样的，分别为 FF GG HH II，但是这四个函数传入的参数是完全相同的，所以我们根据当前的外循环（4 次）序号判断应该，使用的函数，即 funcPoint 函数

```
private static long funcPoint(long a, long b, long c, long d, long Mj, long s, long ti, int choose){
    if(choose == 0)
        return FF(a, b, c, d, Mj, s, ti);
    else if(choose == 1)
        return GG(a, b, c, d, Mj, s, ti);
    else if(choose == 2)
        return HH(a, b, c, d, Mj, s, ti);
    else
        return II(a, b, c, d, Mj, s, ti);
}
```

ii. 其次。这 64 次函数调用的赋值操作是分别对用 A D C B 的循环序列的，所以我们根据此时循环次数对 4 取模的数值判断当前应该被赋值的缓冲区元素是什么

iii. 第三，4 层外循环中，被操作的小组即 4byte 的待加密数据不是都是按照 0-15 的元素顺序来的，所以，我们需要一个设置索引的函数，在每次操作时取相应序号的 byte

```
private static int changeK(int i, int j){
    int k = 0;
    switch(i){
        case 0:
            k = j;
            break;
        case 1:
            k = (1 + 5 * j) % 16;
            break;
        case 2:
            k = (5 + 3 * j) % 16;
            break;
        default:
            k = (7 * j) % 16;
            break;
    }
    return k;
}
```

iv. 最终实现见下方代码实现

d) 需要进行最后的转换部分

```
String resStr="";
long temp=0;
for(int i=0;i<4;i++){
    for(int j=0;j<4;j++){
        temp=res[i]&0x0FL;
        String a=hexs[(int)(temp)];
        res[i]=res[i]>>4;
        temp=res[i]&0x0FL;
        resStr+=hexs[(int)(temp)]+a;
        res[i]=res[i]>>4;
    }
}
System.out.println(resStr+"\n");
```

五、 代码实现

【初始化】

```
static final String hexs[]={"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"};
private static final long A=0x67452301L;
private static final long B=0xefcdab89L;
private static final long C=0x98badcfeL;
private static final long D=0x10325476L;
private static long[] res = {A, B, C, D};
private static long[] deal = {A, B, C, D};
private static final int s[][] = {
    {7,12,17,22,7,12,17,22,7,12,17,22,7,12,17,22},
    {5,9,14,20,5,9,14,20,5,9,14,20,5,9,14,20},
    {4,11,16,23,4,11,16,23,4,11,16,23,4,11,16,23},
    {6,10,15,21,6,10,15,21,6,10,15,21,6,10,15,21}
};
private static final long T[] = {
    0xd76aa478L, 0xe8c7b756L, 0x242070dbL, 0xc1bdceeeL, 0xf57c0fafL, 0x4787c62aL, 0xa8304613L, 0xfd46
    0xf61e2562L, 0xc040b340L, 0x265e5a51L, 0xe9b6c7aaL, 0xd62f105dL, 0x02441453L, 0xd8a1e681L, 0xe7d3
    0xfffa3942L, 0x8771f681L, 0x6d9d6122L, 0xfde5380cL, 0xa4beea44L, 0x4bdecfa9L, 0xf6bb4b60L, 0xbebf
    0xf4292244L, 0x432aff97L, 0xab9423a7L, 0xfc93a039L, 0x655b59c3L, 0x8f0ccc92L, 0xffeff47dL, 0x8584
```

【main】

```
public static void main(String []args){
    MD5 md=new MD5();
    String test = "123456";
    byte[] allBytes = changeToBytes(test);
    long[][] groups = changeToLongGroups(allBytes);
    ///System.out.println(groups.length);
    for(int t = 0; t < groups.length; ++t){
        recycle(groups[t]);

        String resStr="";
        long temp=0;
        for(int i=0;i<4;i++){
            for(int j=0;j<4;j++){
                temp=res[i]&0x0FL;
                String a=hexs[(int)(temp)];
                res[i]=res[i]>>4;
                temp=res[i]&0x0FL;
                resStr+=hexs[(int)(temp)]+a;
                res[i]=res[i]>>4;
            }
        }
        System.out.println(resStr+"\n");
    }
}
```

【changeToBytes】

```
    private static byte[] changeToBytes(String str) {
        int K = str.length() * 8;
        int P = K % 512;
        P = P < 448 ? 448-P : 512-P+448;
        byte[] allBytes = new byte[K/8+P/8+8];
        for(int i = 0; i < K/8; ++i)
            allBytes[i] = str.getBytes()[i];
        for(int i = 0; i < P/8; ++i)
            allBytes[K/8 + i] = (i == 0) ? (byte)128 : (byte)0;

        long len=(long)(K/8<<3);
            for(int i=0;i<8;i++){
                allBytes[K/8 + P/8 +i]=(byte)(len&0xFFL);
                len=len>>8;
            }
        // if(K >= 64)
```

【changeToLongGroups】

```
private static Long[][] changeToLongGroups(byte[] allBytes) {

    int groupNum = allBytes.length / 64;

    Long[][] res = new Long[groupNum][16];
    for(int i = 0; i < groupNum; ++i) {
        for(int j=0; j<16; ++j){
            res[i][j] = (offSigned(allBytes[64*i + j*4])) |
                (offSigned(allBytes[64*i + j*4 + 1])) << 8|
                (offSigned(allBytes[64*i + j*4 + 2])) << 16|
                (offSigned(allBytes[64*i + j*4 + 3])) << 24;

        }
    }

    return res;
}
```

【recycle】

```
private static void recycle(Long[] group){
    // System.out.println("1");
    for(int i = 0; i < 4; ++i){
        for(int j = 0; j < 16; ++j){
            int k = changeK(i, j);
            if(j%4 == 0)
                deal[0] = funcPoint(deal[0], deal[1], deal[2], deal[3], group[k], s[i][j], T[i*16+j], i);
            else if(j%4 == 1)
                deal[3] = funcPoint(deal[3], deal[0], deal[1], deal[2], group[k], s[i][j], T[i*16+j], i);
            else if(j%4 == 2)
                deal[2] = funcPoint(deal[2], deal[3], deal[0], deal[1], group[k], s[i][j], T[i*16+j], i);
            else
                deal[1] = funcPoint(deal[1], deal[2], deal[3], deal[0], group[k], s[i][j], T[i*16+j], i);

        }

    }

    res[0] = (deal[0] + res[0]) & 0xFFFFFFFFL;
    res[1] = (deal[1] + res[1]) & 0xFFFFFFFFL;
    res[2] = (deal[2] + res[2]) & 0xFFFFFFFFL;
    res[3] = (deal[3] + res[3]) & 0xFFFFFFFFL;
}
```

```
private static long F(long X, long Y, long Z) {
    return (X & Y) | ((~X) & Z);
}
private static long G(long X, long Y, long Z) {
    return (X & Z) | (Y & (~Z));
}
private static long H(long X, long Y, long Z) {
    return X^Y^Z;
}
private static long I(long X, long Y, long Z) {
    return Y ^ (X | (~Z));
}
```

```
}
private static long FF(long a, long b, long c, long d, long Mj, long s, long ti) {
    a += (F(b, c, d) & 0xFFFFFFFFL) + Mj + ti;
    a = ((a&0xFFFFFFFFL)<< s) | ((a&0xFFFFFFFFL) >> (32 - s));
    a += b;
    return (a & 0xFFFFFFFFL);
}
private static long GG(long a, long b, long c, long d, long Mj, long s, long ti) {
    a += (G(b, c, d) & 0xFFFFFFFFL) + Mj + ti;
    a = ((a&0xFFFFFFFFL)<< s) | ((a&0xFFFFFFFFL) >> (32 - s));
    a += b;
    return (a & 0xFFFFFFFFL);
}
private static long HH(long a, long b, long c, long d, long Mj, long s, long ti) {
    a += (H(b, c, d) & 0xFFFFFFFFL) + Mj + ti;
    a = ((a&0xFFFFFFFFL)<< s) | ((a&0xFFFFFFFFL) >> (32 - s));
    a += b;
    return (a & 0xFFFFFFFFL);
}
private static long II(long a, long b, long c, long d, long Mj, long s, long ti) {
    a += (I(b, c, d) & 0xFFFFFFFFL) + Mj + ti;
    a = ((a&0xFFFFFFFFL)<< s) | ((a&0xFFFFFFFFL) >> (32 - s));
    a += b;
    return (a & 0xFFFFFFFFL);
}
```

六、 实验结果

在网络上使用工具查询我们可以得到如下样例

要加密的字符串： 123456 加密

字符串	123456
16位 小写	49ba59abbe56e057
16位 大写	49BA59ABBE56E057
32位 小写	e10adc3949ba59abbe56e057f20f883e
32位 大写	E10ADC3949BA59ABBE56E057F20F883E

要加密的字符串： aaaaa 加密

字符串	aaaaa
16位 小写	380a41396ed63dca
16位 大写	380A41396ED63DCA
32位 小写	594f803b380a41396ed63dca39503542
32位 大写	594F803B380A41396ED63DCA39503542

运行代码后检验输出为

```
E:\FILE_myself\Learning\juniorfirst\web安全\homework\MD5>javac MD5.java

E:\FILE_myself\Learning\juniorfirst\web安全\homework\MD5>java MD5
E10ADC3949BA59ABBE56E057F20F883E
```

```
E:\FILE_myself\Learning\juniorfirst\web安全\homework\MD5>javac MD5.java

E:\FILE_myself\Learning\juniorfirst\web安全\homework\MD5>java MD5
594F803B380A41396ED63DCA39503542
```

检验正确！

p.s. 在本代码中，填充的第二步补充 64 位数据时，使用的值为字符串转换后的 bit 长度，此时检验可以得到 MD5 加密工具的结果，但是当使用二进制表达对 2^{64} 取模的方法填充的时候（见代码的注释部分），此时是不能得到正确结果的。

```
// if(K >= 64)
//  for(int i = 0; i < 8; ++i)
//      allBytes[K/8+P/8+i] = allBytes[K/8-8+i];
// else {
//  for(int i = 0; i < 8; ++i) {
//      if((i+K/8) < 8) allBytes[K/8+P/8+i] = 0 & 0xff;
//      else allBytes[K/8+P/8+i] = allBytes[i+K/8-8];
//  }
// }
```