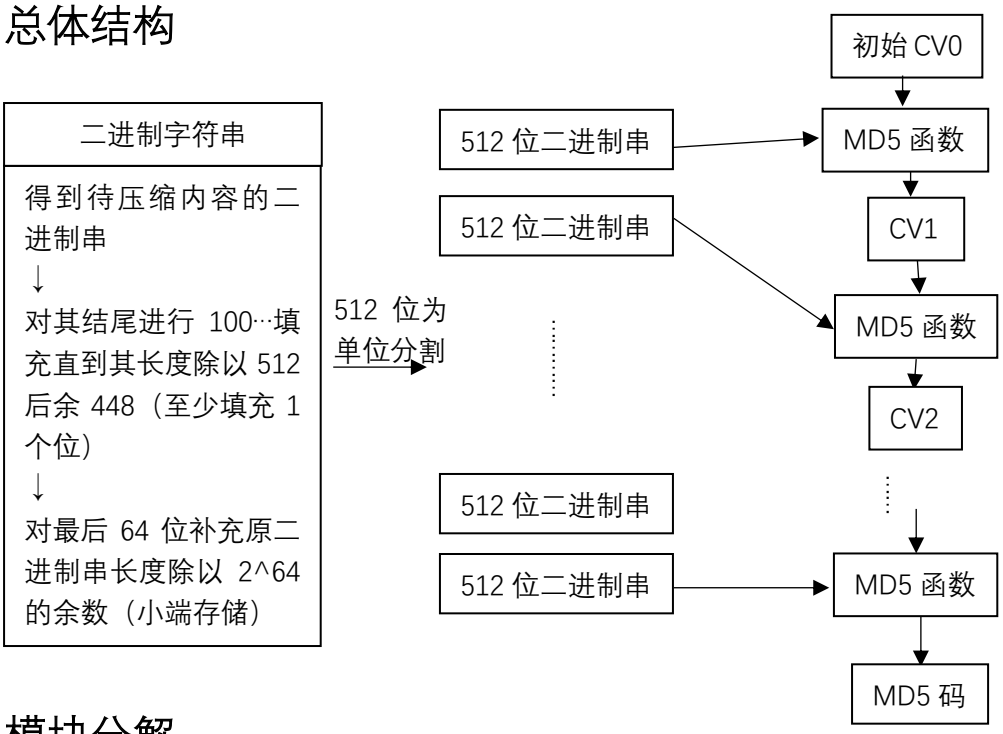
MD5 算法实现

算法原理概述

MD5 即 MD5 Message-Digest Algorithm（MD5 消息摘要算法）。

- MD4 (1990)、MD5(1992, RFC 1321) 由 Ron Rivest 发明，是广泛使用的 Hash 算法，用于确保信息传输的完整性和一致性。
- MD5 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。

总体结构



模块分解

1. 文件读取并转换为元素为 01 二进制的字符串

   该部分的代码实现在 ReadFile.java 中。首先一次性读取某文件的字节流并存储为 byte[]，再通过 Long.toString 函数对字节数组的每一个 byte 元素转换为 01 字符串并添加到输出变量的尾端，并返回输出。转换为二进制字符串是因为这样更容易操作。Byte 转换为 01 字符串时，可能会不足 8 位（高位为 0 省略），为了使每个字节都保持为 8 位，需要将 0 补全。

   测试结果：

   文本内容为：“GirlsFrontLine”（14 个英文字母）

   二进制码长度为 112（14*8）

2. 填充 padding 与分块

该部分的代码实现在 Split.java 中。

需要先得到原始信息长度 K，并通过一定转换得到填充标识长度 P，使得（K+P）%512 = 448。

```
int rest = 512 - lengthK % 512;
int lengthP = rest >= 64 ? (rest - 64) : (rest + 448);
```

并用 lengthK % 2^64 得到尾部。

再通过 substring 对得到的字符串进行分块，放进 String[]数组中

其中值得一提的是，对于最后的 64 位的串，我们是以该串的末端为队列头，取出后推进待加密消息的末端进行填充，操作的单位是 8 位（一个字节）。因此可以说消息的后 64 位与真正的 64 位是相反的（以 8 位为单位）。这里采用的代码如下：

```
int tail = lengthK % mod;
StringBuilder K64 = new StringBuilder();
for(int i = 0; i < 8; i ++) {
    long temp = tail & 0xFFL;
    StringBuilder s = new StringBuilder(Long.toString(temp, radix: 2));
    while (s.length() != 8) s.insert( offset: 0,  c: '0');
    K64.append(s);
    tail = tail >> 8;
}
```

3. MD 缓冲区

在 MDRegister 类中，有 4 个 long 成员，用于存储该次 MD5 函数运算的输入 CV 向量的初始值，这 4 个成员将一直保持不变用于最后迭代结束的加法。此外还有 1 个 4 元素长的 long 数组，用于存储 4 个向量，将随着每次迭代改变。

4. MD5 函数

在 MD5 类中，有 4 个成员变量——传入的 CV 向量（在构造函数中利用四个参数作为 ABCD 初始化 MDRegister 成员）、long 数组（一个 512 位二进制串，在构造函数中将传入的二进制字符串转换位 8 位一元素的 long 数组，且使用小端存储）、int 数组（存储移位数，声明时可直接初始化）、long 数组（存储 T 表，声明时可直接初始化）。

```
public MDRegister CV;
private long X[];
private 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 long T[] = {
        0xd76aa478L, 0xe8c7b756L, 0x242070dbL, 0xc1bdceeeL,
        0xf57c0fafL, 0x4787c62aL, 0xa8304613L, 0xfd469501L,
        0x698098d8L, 0x8b44f7afL, 0xffff5bb1L, 0x895cd7beL,
        0x6b901122L, 0xfd987193L, 0xa679438eL, 0x49b40821L,
        0xf61e2562L, 0xc040b340L, 0x265e5a51L, 0xe9b6c7aaL,
        0xd62f105dL, 0x02441453L, 0xd8a1e681L, 0xe7d3fbc8L,
        0x21e1cde6L, 0xc33707d6L, 0xf4d50d87L, 0x455a14edL,
        0xa9e3e905L, 0xfcefa3f8L, 0x676f02d9L, 0x8d2a4c8aL,

        0xfffa3942L, 0x8771f681L, 0x6d9d6122L, 0xfde5380cL,
        0xa4beea44L, 0x4bdecfa9L, 0xf6bb4b60L, 0xbebfbc70L,
        0x289b7ec6L, 0xeaa127faL, 0xd4ef3085L, 0x04881d05L,
        0xd9d4d039L, 0xe6db99e5L, 0x1fa27cf8L, 0xc4ac5665L,
        0xf4292244L, 0x432aff97L, 0xab9423a7L, 0xfc93a039L,
        0x655b59c3L, 0x8f0ccc92L, 0xffeff47dL, 0x85845dd1L,
        0x6fa87e4fL, 0xfe2ce6e0L, 0xa3014314L, 0x4e0811a1L,
        0xf7537e82L, 0xbd3af235L, 0x2ad7d2bbL, 0xeb86d391L
};
```

```
public MD5(long A, long B, long C, long D, String str) {
    CV = new MDRegister(A, B, C, D);
    X = new long[16];
    for (int i = 0; i < 16; i ++) {
        String substr = str.substring(i * 32, i * 32 + 32);
        X[i] = 0x00000000L;
        for (int j = 1; j <= 4; j ++) {
            for (int k = 32 - 8 * j; k < 40 - 8 * j; k ++) {
                X[i] = X[i] << 1;
                if (substr.charAt(k) == '1') X[i] = X[i] | 0x00000001L;
            }
        }
    }
}
```

成员函数 F、G、H、I，都有三个参数 B,C,D(long)，分别进行轮函数

轮次	Function g	$g(b, c, d)$
1	$F(b, c, d)$	$(b\wedge c)\vee(\neg b\wedge d)$
2	$G(b, c, d)$	$(b\wedge d)\vee(c\wedge\neg d)$
3	$H(b, c, d)$	$b\oplus c\oplus d$
4	$I(b, c, d)$	$c\oplus(b\vee\neg d)$

成员函数 Fstep，Gstep，Hstep，Istep 分别是用 F、G、H、I 迭代一次的函数。函数参数有传入的向量 A、B、C、D（long）以及 i（int，表示该为第 i 次迭代（总共 16*4 次））
一次迭代包括两步：
(1) 对 A 迭代：a ← b + ((a + g(b, c, d) + X[k] + T[i]) <<<s)
(2) 缓冲区(A, B, C, D) 作循环轮换：(B, C, D, A) ← (A, B, C, D)
因此对每个 step 函数我们只需要修改 g 调用的函数以及 X[k]中 k 的式子即可。如在 Fstep 中 g 为 F，k 为 i。此后的 k 分别为 (1 + 5 * i) % 16、(5 + 3 * i) % 16、(7 * i) % 16、g 则分别为 G、H、I。其余的运算部分都是一样的。此处举例 Fstep 的代码：

```
private void Fstep(long A, long B, long C, long D, int i) {
    long temp1 = A + (F(B, C, D)&0xFFFFFFFFL) + X[i] + T[i];
    temp1 = temp1 & 0xFFFFFFFFL;
    long left = temp1 << s[i];
    long right = temp1 >>> (32 - s[i]);
    long temp2 = left | right;
    A = temp2 + B;
    A = A & 0xFFFFFFFFL;
    this.CV.CV[0] = D;
    this.CV.CV[1] = A;
    this.CV.CV[2] = B;
    this.CV.CV[3] = C;
}
```

在我的函数中，循环左移是分别左移 s[i]，得到结果左部分，无符号循环右移 32-s[i]，得到结构右半部分，两者相与得到左移结果。最后将 ABCD 分别复制给对象的 CV 成员变量（要循环轮换）。
而对于一个 512 位的待压缩消息，我们只需要分别做 16 次 Fstep，Gstep，Hstep，Istep，最后再与传入的初始 CV 向量相加则可得到用于下一次 MD5 的 CV 向量参数（或 MD5 结果）

```java
public void md5() {
    for (int i = 0; i < 16; i ++) {
        this.Fstep(CV.CV[0], CV.CV[1], CV.CV[2], CV.CV[3], i);
    }

    for (int i = 16; i < 32; i ++) {
        this.Gstep(CV.CV[0], CV.CV[1], CV.CV[2], CV.CV[3], i);
    }

    for (int i = 32; i < 48; i ++) {
        this.Hstep(CV.CV[0], CV.CV[1], CV.CV[2], CV.CV[3], i);
    }

    for (int i = 48; i < 64; i ++) {
        this.Istep(CV.CV[0], CV.CV[1], CV.CV[2], CV.CV[3], i);
    }


    for (int i = 0; i < 4; i ++) {
        CV.CV[i] = CV.CV[i] + CV.CV0()[i];
        CV.CV[i] = CV.CV[i] & 0xFFFFFFFFL;
    }
}
```

5. 顶层模块

该部分的代码实现在 Main.java 中.

顶层模块需要获得待压缩的文件路径，调用 ReadFile 类中的 readToString 函数得到其二进制字符串，再调用 Split 类中的 padding 填充字符串，调用 Split 类中的 split 函数得到字符串数组（每个长 512）。同时需要声明初始向量 IV（ABCD）（小端存储）。

- *A* = 0x67452301
- *B* = 0xEFCDAB89
- *C* = 0x98BADCFE
- *D* = 0x10325476

Word *A*	01	23	45	67
Word *B*	89	AB	CD	EF
Word *C*	FE	DC	BA	98
Word *D*	76	54	32	10

然后对字符串数组中的每个元素都作相同的操作更新向量 CV 中的值:

```java
for (String s : Y) {
    MD5 md5 = new MD5(CV0.CV[0], CV0.CV[1], CV0.CV[2], CV0.CV[3], s);
    md5.md5();
    CV0 = md5.CV;
}
```

最终得到的 CV0 就是我们要的 MD5 码

MD5 码是小端存储，因此打印之前需要将其重新换位输出。

```java
for (int i = 0; i < 4; i ++) {
    long[] b = new long[4];
    result[i] = 0xFFFFFFFFL;
    b[0] = CV0.CV[i] & 0xFFL;
    b[1] = CV0.CV[i] & 0xFF00L;
    b[2] = CV0.CV[i] & 0xFF0000L;
    b[3] = CV0.CV[i] & 0xFF000000L;
    result[i] = (b[0] << 24) | (b[1] << 8) | (b[2] >>> 8) | (b[3] >>> 24);
    System.out.print(Long.toString( i: result[i] & 0xFFFFFFFFL,  radix: 16));
}
```

数据结构

变量	数据类型
单个寄存器	long（&0xFFFFFFFFL 以当成 32 位长）
寄存器组	MDRegister 类 或 long 数组
二进制字符串	String
T 表	长度 64 的 long 数组
S 表	长度 64 的 int 数组

Java 源代码

MD5/src 中。

编译运行结果

```
"C:\Program Files\Java\jdk-11.0.1\bin\java.exe" "-
请输入文件地址：
C:\Users\azu\OneDrive\Course\web安全技术\MD5\M.txt
380b6a155ca863e2c041fa6074059d53
Process finished with exit code 0
```

文本内容为：

The MD5 message-digest algorithm is a widely used hash function producing a 128-bit hash value. Although MD5 was initially designed to be used as a cryptographic hash function, it has been found to suffer from extensive vulnerabilities. It can still be used as a checksum to verify data integrity, but only against unintentional corruption. It remains suitable for other non-cryptographic purposes, for example for determining the partition for a particular key in a partitioned database.

借用一个 MD5 加密网站的结果：

https://md5jiami.51240.com/

字符串	The MD5 message-digest algorithm is a widely used hash function producing a 128-bit hash value. Although MD5 was initially designed to be used as a cryptographic hash function, it has been found to suffer from extensive vulnerabilities. It can still be used as a checksum to verify data integrity, but only against unintentional corruption. It remains suitable for other non-cryptographic purposes, for example for determining the partition for a particular key in a partitioned database.
16位 小写	5ca863e2c041fa60
16位 大写	5CA863E2C041FA60
32位 小写	380b6a155ca863e2c041fa6074059d53
32位 大写	380B6A155CA863E2C041FA6074059D53

与运行结果一致。字符串有长度将近 500 个字符，因此估计其可分成八段 512 位串，因此可验证功能完整（单个 MD5 以及 CV 传递）与可行。

由于非 ASCII 码内的字符可能会有编码问题导致输出 MD5 有差异因此推荐使用英文字符测试。

虽然 MD5/out/production 内有编译后的 class 以及 artifacts 内有 jar 包，但由于版本较新（JDK11，版本 55），因此推荐用 MD5/src 内的 java 文件重新编译运行（Main.java 作为 main 运行类）。

MD5/M.txt 为测试用待压缩文件，可无视，也可借用测试，内容为为 ASCII 码内字符，无编码问题。