**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

# 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。

# 总体结构与流程

MD5算法具体流程如下：

1. 填充：
   1. 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
   2. 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264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
   1. 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
   2. 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))

# 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。

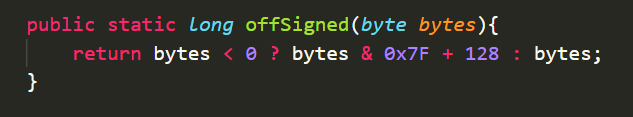
# 数据设计及算法

## 【数据设计】

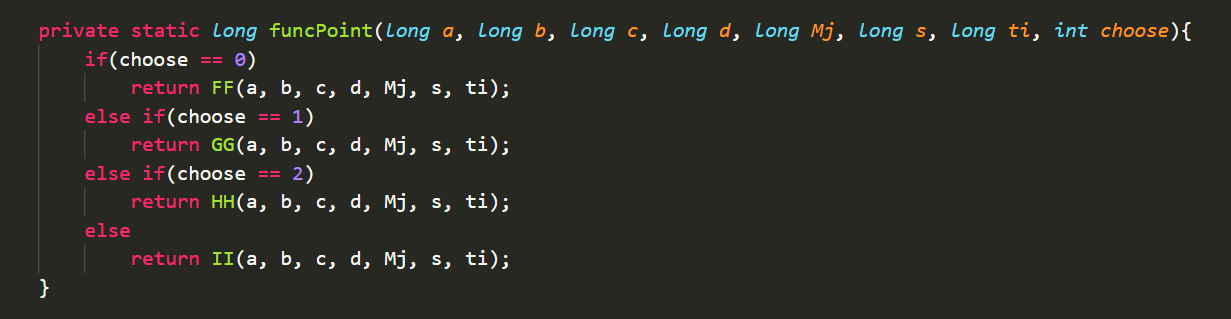
1. **变量声明**
   1. 缓冲区基础数据：A B C D
   2. 循环函数中的S：4\*16的二维数组，4为外层循环，16为内层循环，表示一个循环周期4\*16次操作时函数中参数s的值
   3. 循环函数中的T：长度为64的数组，表示个循环周期4\*16次操作时函数中参数ti的值
   4. K（初始位大小）P（需填充位大小）allBytes（转换后的字节数组）group[][]（使用块号和组号进行调用的byte数据）
2. **函数定义**
   1. Main函数->按算法步骤调用函数、最后将得到的结果转换为32位16进制输出
   2. changeToBytes函数->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
   3. 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
   4. 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
   5. 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。
   6. funcPoint offSign changek函数（在特殊处理中有介绍）
3. **特殊处理：**
   1. 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）



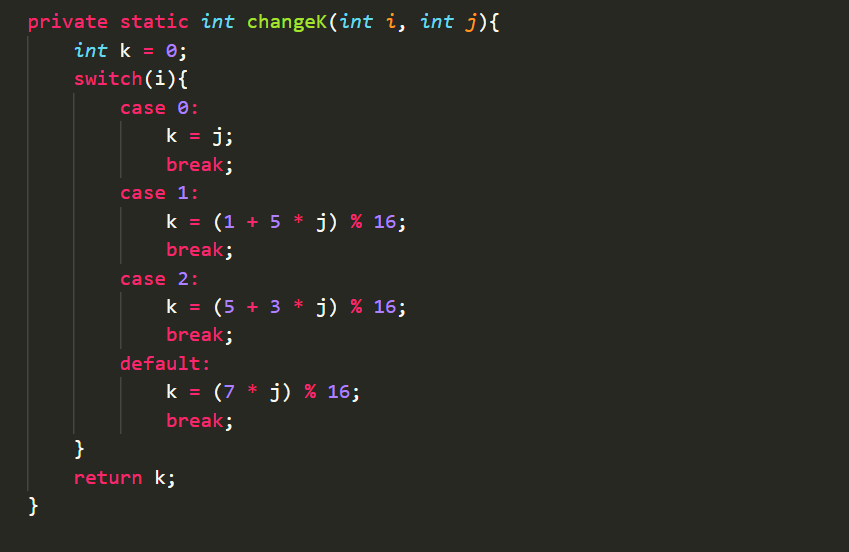
* 1. 当对已经填充好的bytes数组进行分块时，我们需要注意，byte的取值范围是-128~127，我们赋值的128会记录为负数，或者说，在byte中，1开头的数据为负数，但此时我们并不需要符号位，所以我们应当使用方法将符号位转换为计数位



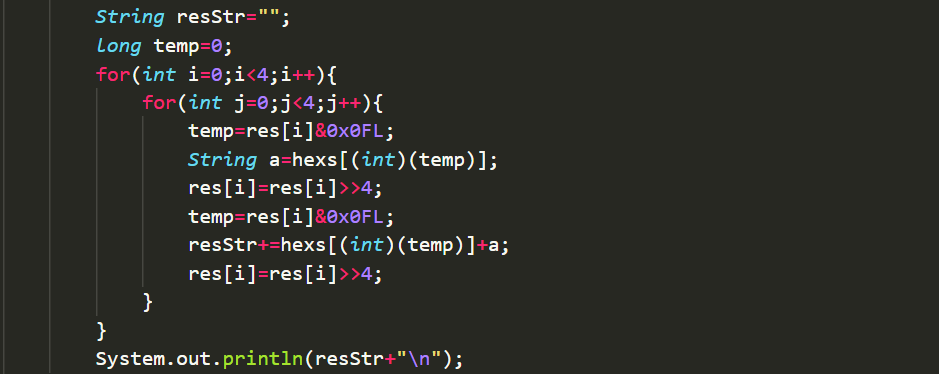
* 1. 在实现循环时，如果直接写64行代码会显得代码很厚重，此时，我们需要改变这部分代码的撰写方式：
     1. 首先，4次外循环中，每一次的函数是不一样的，分别为FF GG HH II，但是这四个函数传入的参数是完全相同的，所以我们根据当前的外循环（4次）序号判断应该，使用的函数，即funcPoint函数



* + 1. 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
    2. 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

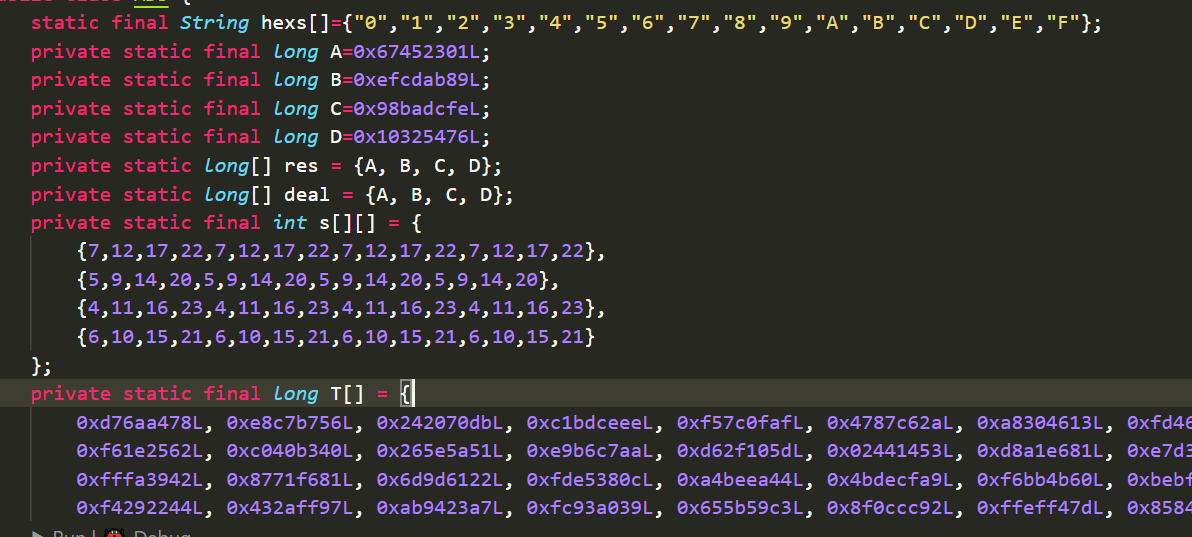


* + 1. 最终实现见下方代码实现
  1. 需要进行最后的转换部分



# 代码实现

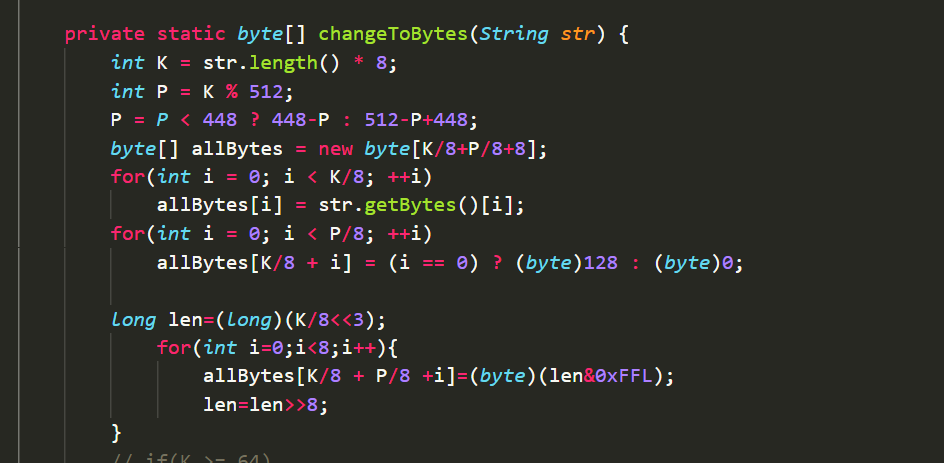
## 【初始化】



## 【main】



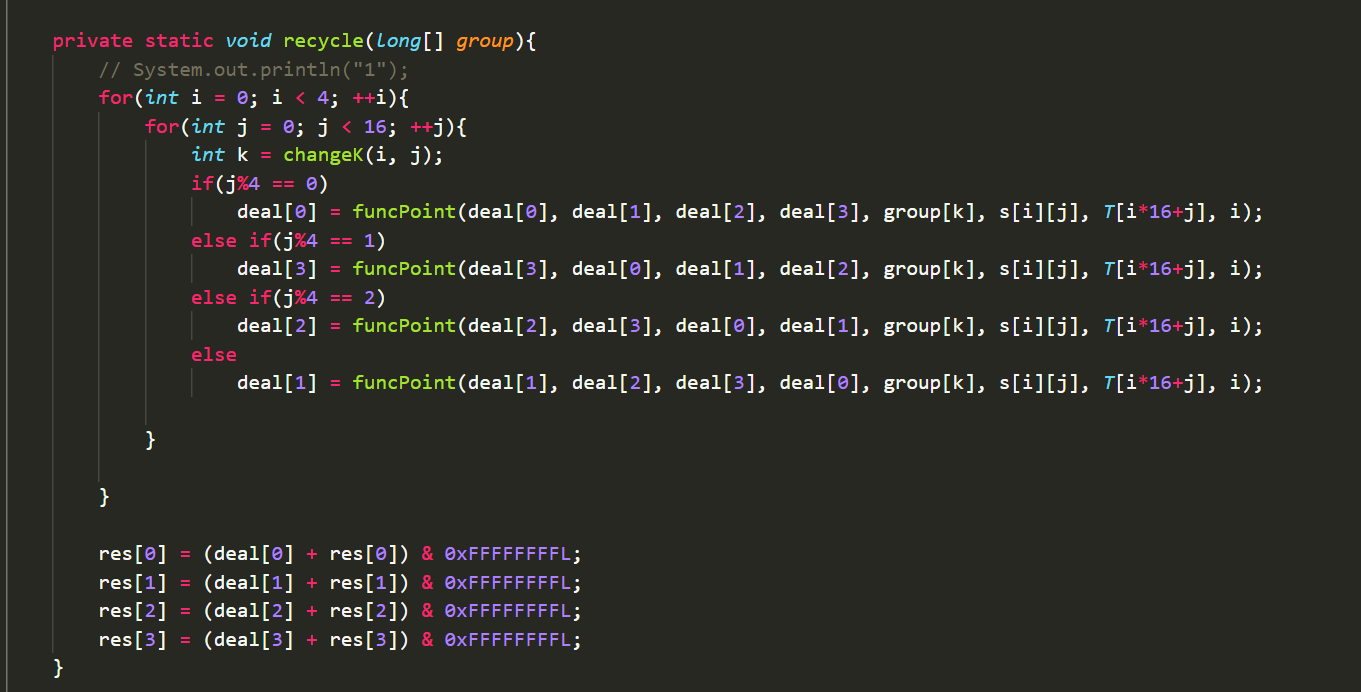
## 【changeToBytes】

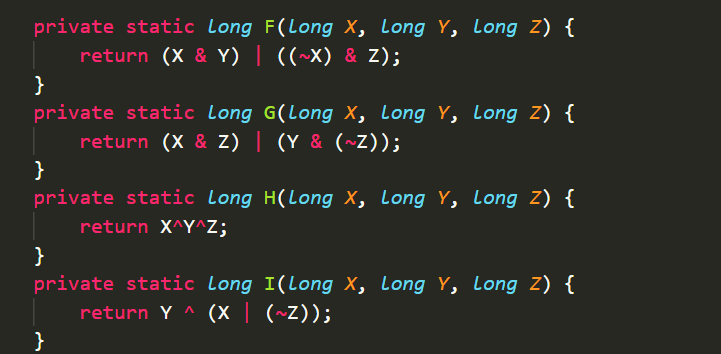


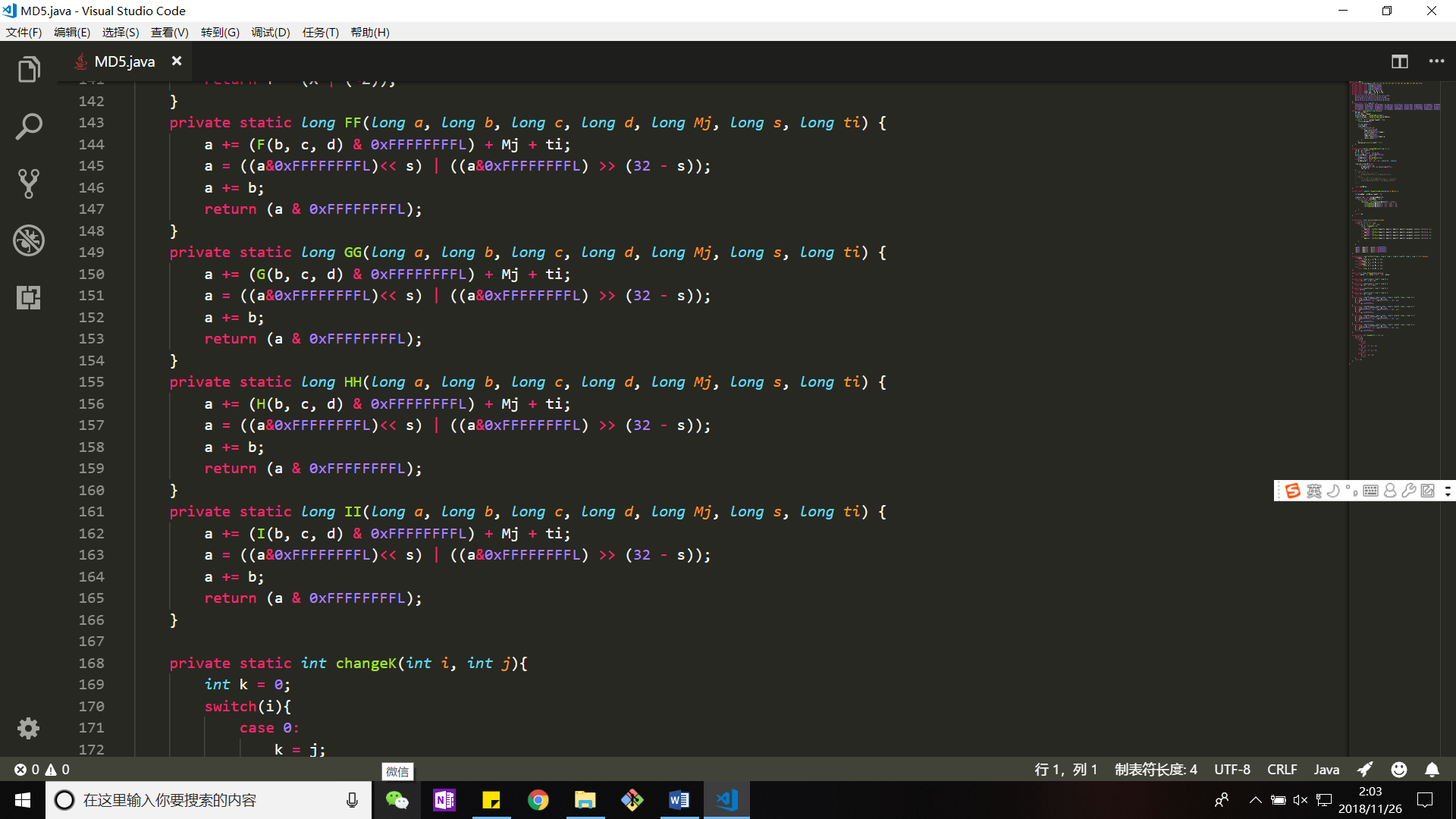
## 【changeToLongGroups】



## 【recycle】







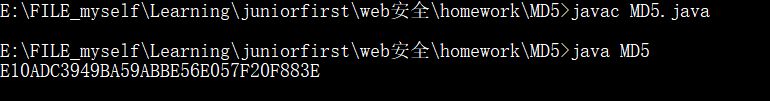
# 实验结果

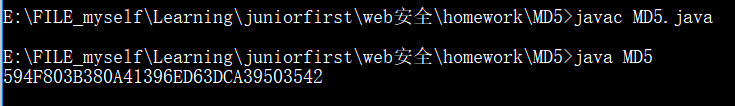
在网络上使用工具查询我们可以得到如下样例





运行代码后检验输出为





**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264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。

