MD5

第一步

将需要散列的 16进制 字符串补充到 长度mod512 = 0， 参考代码：MD1_Data 结构 的 构造函数。（同SM3）

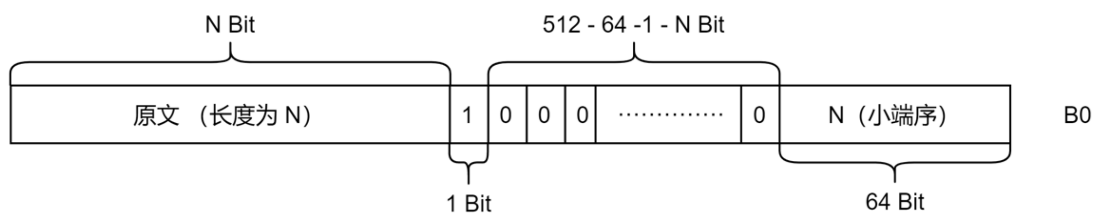
补充方式

设 需要散列的 文本为"M"，M的长度为N Bit , B_i = N/512 , L = N%512。

注意

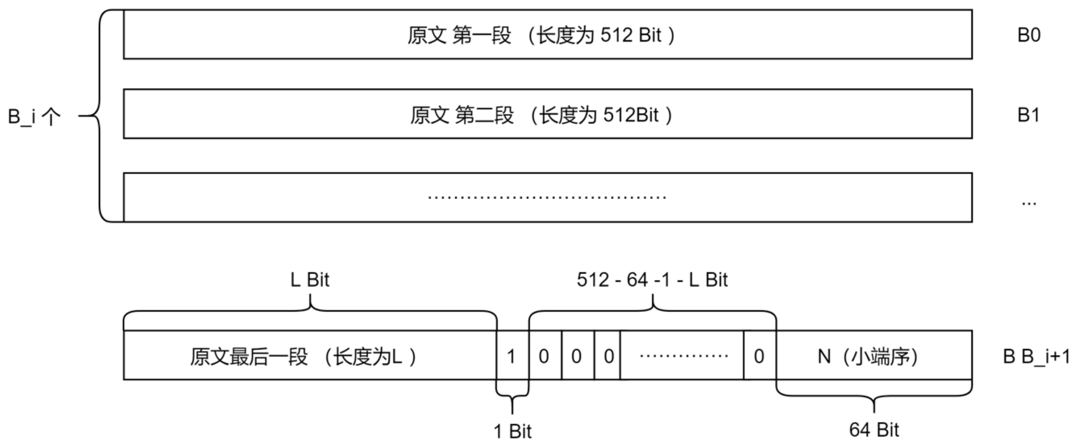
这个和SHA系列不一样，MD5最后补的 N 是小端序的！！！

N <= 512-64-1 Bit：

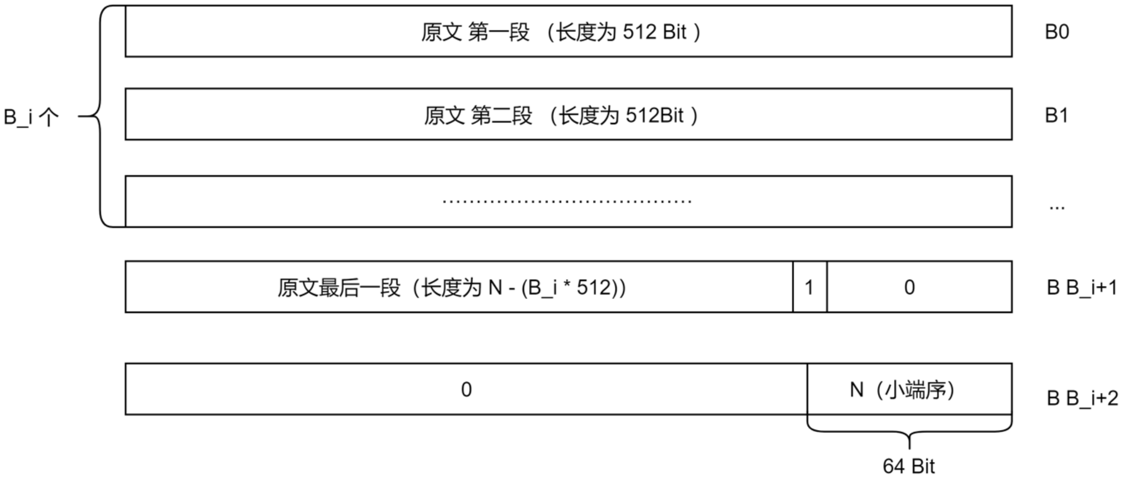


N > i * 512 Bit (i∈N* && i > 1):

512-L>64+1:



512-L<64+1:



第二步

将经过第一步的 M 叫为 Mc，将Mc分为 i 个 512 Bit 的B（上图中的 B0,B1,...,Bi-1）

对每一个B 计算 对应的W (16个)，参考代码： `BOOLEAN Mc_To_W(PMD5_Data Data);`

W0~W15为B的划分(B为512Bit长，划分一个W 32 Bit长)（不需要以四字节为单位转换端序）

我的代码中，对于一个消息 Mc，对应的W的样式如下：每个B有63个W

	W0	**W1**	**W2**	**W3**	**.....**	**W15**
B0	nW[0] [0]					
B1		nW[1] [1]				
B2			nW[2] [2]			
...						
Bi-1	nW[i-1] [0]	nW[i-1] [1]	nW[i-1] [2]	nW[i-1] [3]		nW[i-1] [63]

算出每个B的W0~W63,填入上述表格中。

第三步

对于i个B（每个B有16个W），都要计算：

```
for (ULONG32 i = 0; i < B_i; i++) {
        A = nH[i][0]; B = nH[i][1]; C = nH[i][2]; D = nH[i][3];
        for (ULONG32 j = 0; j < 64; j++) {
            if (j < 16) {
                F = (B & C) | (~B & D);
                G = j;
            }
            else if (j < 32) {
```

```
            F = (D & B) | (~D & C);
            G = (5 * j + 1) % 16;
        }
        else if (j < 48) {
            F = B ^ C ^ D;
            G = (3 * j + 5) % 16;
        }
        else {
            F = C ^ (B | ~D);
            G = (7 * j) % 16;
        }
        T1 = D;
        D = C;
        C = B;
        B = B + SL((A + F + K[j] + Data->nW[i][G]), r[j]);
        A = T1;
    }
    nH[i + 1][0] = A + nH[i][0];
    nH[i + 1][1] = B + nH[i][1];
    nH[i + 1][2] = C + nH[i][2];
    nH[i + 1][3] = D + nH[i][3];
}
```

nH的结构如下:

	[0]	[1]	[2]	[3]
nH[0]	0x67452301	0xefcdab89	0x98badcfe	0x10325476
nH[1]		nH[1] [1]		
...				
nH[i-1]				
nH[i]	nH[i] [0]	nH[i] [1]	nH[i] [2]	nH[i] [3]

共 i+1个nH（i为B的个数），每个nH有4个ULONG32。

nH[0]有初始值,如上表格所示。

最后得到的nH[i]即为MD5最后结果。+

补充细节

上述算法中需要用到的常量与工具

```
//加密时用到的64个常量
ULONG32 K[] = {
    0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,
    0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,
    0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,
    0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,
```

```
    0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,
    0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,
    0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,
    0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,
    0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,
    0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,
    0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05,
    0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,
    0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,
    0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,
    0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,
    0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391
};

ULONG32 r[] = {
    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
};

ULONG32 SL(ULONG32 x, int n) {//左循环n位
    ULONG32 y = x << n;
    x = x >> (32 - n);
    return x | y;
}
```