JPEG 文件编码方式说明（Version 1.0）

由于个人水平有限，文档难免有许多不足或错误，欢迎各位对文档提出宝贵的意见和建议，如有任何疑问访问我的博客: http://blog.sina.com.cn/u/5617154115，内有邮箱联系方式。

一、JPEG 文件格式编码理论介绍

1、颜色模式转换

我们知道，BMP 图片的编码是基于 RGB 颜色空间（即每一个像素都由(R,G,B)三个分量构成）进行的，而 JPEG 采用的是 YCrCb 颜色空间，即每一个像素都由(Y,Cr,Cb)三个分量构成，其中 Y 代表亮度，Cr,Cb 则代表色度和饱和度(也有人将 Cb,Cr 两者统称为色度)。三者通常以 Y,U,V 来表示，即用 U 代表 Cb，用 V 代表 Cr。要想对 BMP 图片进行压缩，首先需要进行颜色空间的转换。RGB 和 YCrCb 之间的转换关系如下所示：

$$\begin{pmatrix} Y \\ Cr \\ Cb \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.500 & -0.4187 & -0.0813 \\ -0.1687 & -0.3313 & 0.5 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

在实际的 jpeg 编码中，为了方便数据的存储和计算，对于上述得到的 Y,Cr,Cb 值，均加上 128，使其变为 8 位的无符号整数。即

$$\begin{pmatrix} Y \\ Cr \\ Cb \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.500 & -0.4187 & -0.0813 \\ -0.1687 & -0.3313 & 0.5 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$$

至此，我们将图片的颜色空间由 RGB 空间转化成 YCrCb 空间。

注：对于灰度图来说，只有 Y 分量，没有 Cr，Cb 分量（或者说均为 0）。因为对于灰度图来说，每一个像素的 R 值，G 值，B 值都是相同的。不妨设 R 值，G 值，B 值均为 a，代入上述第一个矩阵方程，可以得到 Y=a，Cr=0，Cb=0。

2、采样

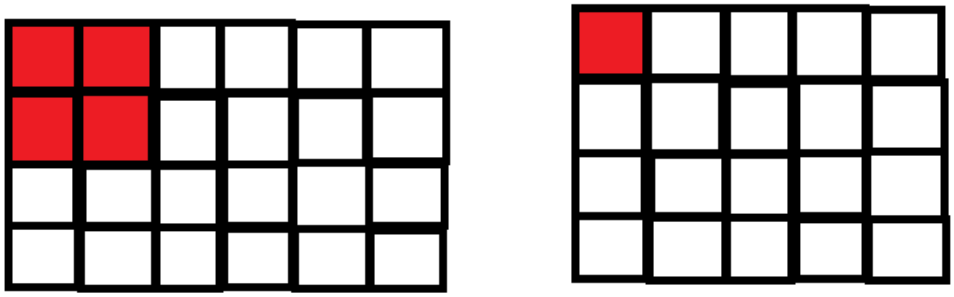
研究发现，人眼对亮度分量更为敏感，对色度分量的数据相对不敏感，因此，我们可以认为 Y 分量要比 Cr,Cb 分量重要的多，所以对于色度分量，可以只取一部分，以增加压缩比。。在 BMP 图片中，RGB 三个分量各采用一个字节进行采样，也就是我们常听到的 RGB888 的模式；而 JPEG 图片中，通常采用两种采样方式：YUV411 和 YUV422，其含义是 YUV 三个分量的数据取样比例。举例来说，如果 Y 取四个数据单元，即水平取样因子 Hy 乘以垂直取样因子 Vy 的值为 4，而 U 和 V 各取一个数据单元，即 Hu×Vu=1,Hv×Vv=1。那么这种部分取样就称为 YUV411。（在 2x2 的单元中，本应分别有 4 个 Y，4 个 U，4 个 V 值，用 12 个字节进行存储。经过 4:1:1 采样处理后，每个单元中的值分别有 4 个 Y、1 个 U、1 个 V，只要用 6 个字节就可以存储了）。这样的采样方式，虽然损失了一定的精度但也在人眼不太察觉到的范围内减小了数据的存储量。当然，JPEG 格式里面也允许将每个点的 U,V 值都记录下来；

填充

在 Jpeg 里,将每 8x8 个原始数据称为一个数据单元(DU)，因此，如果原始图片的长宽不是 8 的倍数，则需要进行填充，必须使其满足是 8 的倍数。又由于采样方式的影响。但若采样因子为 4：1：1，即（2*2）：（1*1）：（1*1），Y 分量的水平和垂直方向都是每 2 个像素采样 2 次；Cr 分量和 Cb 分量的水平和垂直方向都是每 2 个像素采样 1 次。则需要额外满足另一条件。在编码中，将若干个数据单元记为一个 MCU，jpeg 以 MCU 为单位进行编码。对于采样因子为 4：1：1 的情况，每 4 个完整的数据单元记为一个 MCU，其中 Y 分量有 64×4 个采样点；Cr 分量和 Cb 分量各自只有 64×1 个采样点。因此需要对原始图片进行填充，使其长宽满足是 8×2 的倍数。而对于采样因子为 1：1：1 的情况，只需使其长宽满足是 8×1 的倍数。

例：对于一幅 32*35 的图像，若采样因子为 4：1：1，则需将其填充为 32*48 的图像，若采样因子为 1：1：1，则需将其填充为 32*40 的图像，

相反，在解码过程中，对于超出原图高度和宽度的数据应该直接舍去。



如上图，每一个小方格都是 8×8 的数据，即为一个数据单元。

对于采样因子为 4：1：1 的情况，对应于左图，其中红色部分为一个 MCU，包含 4 个数据单元。在数据流中，MCU 的顺序是 MCU1，MCU2，MCU3，MCU4，MCU5，MCU6，（对于左上图来说，顺序为从上到下，从左到右），红色部分为 MCU1。

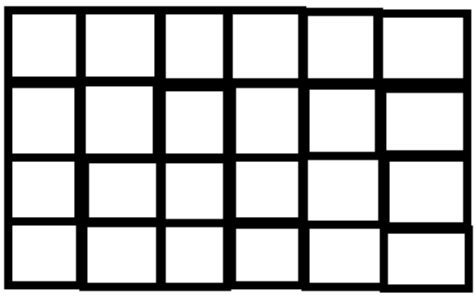
每个 MCU 有 4 个数据单元。如果以 MCU1 说明 MCU 数据的次序，如上述左图所示，红色部分为 MCU1，由于有 4 个数据单元，对于 Y 来说，每个数据单元每 1 个像素采样 1 次。从而可以得到 4 块 Y 分量，记为 Y1_DU,Y2_DU,Y3_DU,Y4_DU，对于 Cr 来说，由于每 2 个像素采样 1 次。从而由 MCU1 采样只能得到一块 Cr 分量，记为 Cr_DU，对于 Cb 来说，与 Cr 类似，不再赘述。

因此，由于 MCU1 得到的采样数据按顺序排列为：

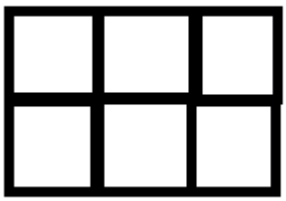
Y1_DU，Y2_DU，Y3_DU，Y4_DU，CrDU，CbDU

从而对于得到各个分量的数量有如下

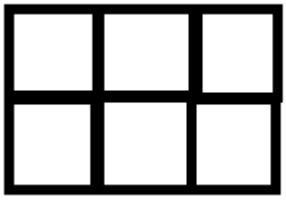
Y 分量:



Cr 分量:



Cb 分量:

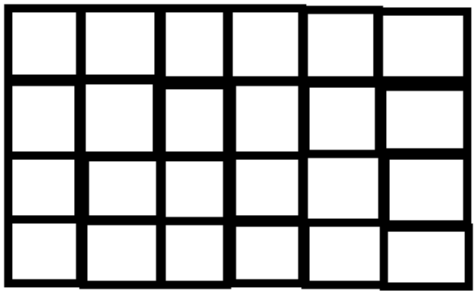


对于采样因子为 1：1：1 的情况，对应于右图，其中红色部分为一个 MCU，包含 1 个数据单元。在数据流中，MCU 的顺序是 MCU1，MCU2，MCU3，MCU4，MCU5，MCU6，…..，MCU20。红色部分为 MCU1。每个 MCU 有 1 个数据单元。如果以 MCU1 说明 MCU 数据的次序，则依次为

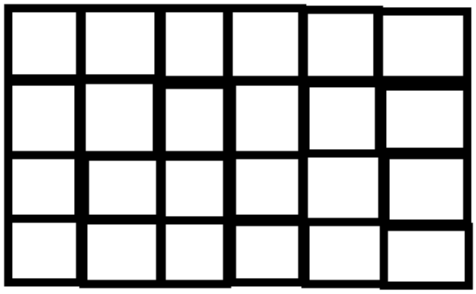
YDU, CrDU, CbDU。

从而对于得到各个分量的数量有如下

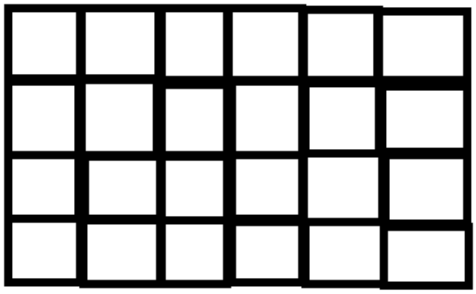
Y 分量:



Cr 分量:



Cb 分量:



解码过程中，先从 JPG 文件中读出采样系数,这样就知道了 MCU 的大小,算出整个图象有几个 MCU.解码程序再循环逐个对 MCU 解码,一直到检查到 EOI 标记.对于每个 MCU,按正规的次序解出每个 DU,然后组合,转换成(R,G,B)。

3、分块

由于后面的 DCT 变换是是对 8x8 的子块进行处理的，因此，在进行 DCT 变换之前必须把源图象数据进行分块。（在第 2 部分实际上已经采用了分块的分析方式，即将 8x8 的数据块记为数据单元 DU）。源图象中每点的 3 个分量(Y,Cr,Cb)是交替出现的，所以先要把这 3 个分量分开，存放到 3 张表中去。然后由左及右，由上到下依次读取 8x8 的子块，存放在长度为 64 的表中，即可以进行 DCT 变换。注意，编码时，程序从源数据中读取一个 8x8 的数据块后，进行 DCT 变换，量化，编码，然后再读取、处理下一个 8*8 的数据块。

由于在第 2 步（采样）中已经将长宽补成了 8 的倍数，使其可以很方便的进行一块块的处理。

4、离散余弦变换 DCT

分块之后，**还必须将每个数值减去 128**，然后一一带入 DCT 变换公式，即可达到 DCT 变换的目的。(图像的数据值必须减去 128，是因为 DCT 公式所接受的数字范围是-128 到 127 之间。)

DCT（Discrete Cosine Transform，离散余弦变换），是码率压缩中常用的一种变换编码方法。任何连续的实对称函数的傅里叶变换中只含有余弦项，因此，余弦变换同傅里叶变换一样具有明确的物理意义。DCT 是先将整体图像分成 N*N 的像素块，然后针对 N*N 的像素块逐一进行 DCT 操作。需要提醒的是，JPEG 的编码过程需要进行正向离散余弦变换，而解码过程则需要反向离散余弦变换。

8*8 数据块正向离散余弦变换计算公式：

$$F(u,v)=\frac{2}{\sqrt{8*8}}c(u)c(v)\sum_{x=0}^{8-1}\sum_{y=0}^{8-1}f(x,y)\cos\frac{(2x+1)u\pi}{2*8}\sin\frac{(2y+1)v\pi}{2*8}$$

$$c(u),c(v)=\begin{cases}\frac{1}{\sqrt{2}}, u,v=0\\ 1\text{，其他}\end{cases}$$

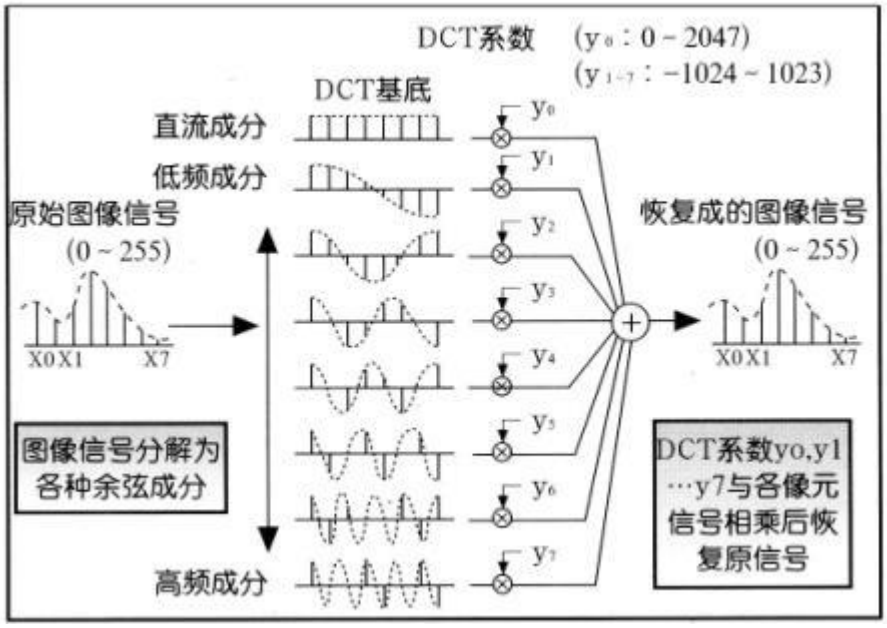
8*8 数据块反向离散余弦变换计算公式：

$$f(x,y)=\frac{2}{\sqrt{8*8}}\sum_{u=0}^{8-1}\sum_{v=0}^{8-1}F(u,v)\mathrm{c}(u)\mathrm{c}(v)\cos\frac{(2x+1)u\pi}{2*8}\sin\frac{(2y+1)v\pi}{2*8}$$

8*8 的二维像素块经过 DCT 操作之后，就得到了 8*8 的变换系数矩阵。这些系数，都有具体的物理含义，例如，u=0，v=0 时的 F（0,0）是原来的 64 个数据的均值，相当于直流分量，也有人称之为 DC 系数或者直流系数。随着 u，v 的增加，相另外的 63 个系数则代表了水平空间频率和垂直空间频率分量（高频分量）的大小，多半是一些接近于 0 的正负浮点数，我们称之为交流系数 AC。DCT 变换后的 8*8 的系数矩阵中，低频分量集中在矩阵的左上角。高频成分则集中在右下角。

这里，我们暂时先只考虑水平方向上一行数据（8 个像素）的情况时的 DCT 变换，从而来说明其物理意义。如下图所示：

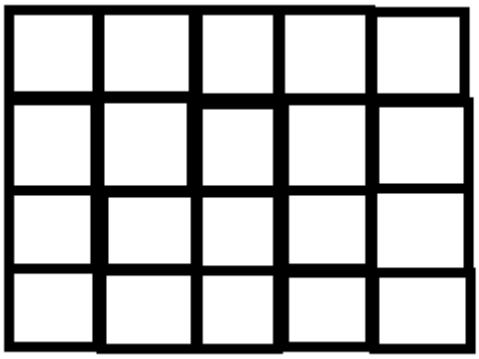
原始的图像信号（最左边的波形）经过 DCT 变换之后变成了 8 个波，其中第一个波为直流成分，其余 7 个为交流成分。



可见图像信号被分解为直流成分和一些从低频到高频的各种余弦成分。而 DCT 系数只表示了该种成分所占原图像信号的份额大小。显然，恢复图像信息可以表示为下面的式子：

F(n) = C(n)*E(n),这里，E(n)是一个基底，C(n)是 DCT 系数，F(n)则是图像信号；如果考虑垂直方向的变化，那就需要一个二维的基底。大学里面的信号处理，傅里叶变换等课程上也讲过，任何信号都可以被分解为基波和不同幅度的谐波的组合，而 DCT 变换的物理意义也正是如此。

由于大多数图像的高频分量比较小，相应的图像高频分量的 DCT 系数经常接近于 0，再加上高频分量中只包含了图像的细微的细节变化信息，而人眼对这种高频成分的失真不太敏感，所以，可以考虑将这一些高频成分予以抛弃，从而降低需要传输的数据量。这样一来，传送 DCT 变换系数的所需要的编码长度要远远小于传送图像像素的编码长度。到达接收端之后通过反离散余弦变换就可以得到原来的数据，虽然这么做存在一定的失真，但人眼是可接受的，而且对这种微小的变换是不敏感的。



对于上述图片的每个格子，均分别进行 8×8 的 DCT 变换，

5、量化

图像数据转换为 DCT 频率系数之后，还要进行量化阶段，才能进入编码过程。量化阶段需要两个 8*8 量化矩阵数据，一个是专门处理亮度的频率系数，另一个则是针对色度的频率系数，将频率系数除以量化矩阵的值之后取整，即完成了量化过程。当频率系数经过量化之后，将频率系数由浮点数转变为整数，这才便于执行最后的编码。不难发现，经过量化阶段之后，所有的数据只保留了整数近似值，也就再度损失了一些数据内容。在 JPEG 算法中，由于对亮度和色度的精度要求不同，分别对亮度和色度采用不同的量化表。前者细量化，后者粗量化。

下图给出 JPEG 的亮度量化表和色度量化表，该量化表是从广泛的实验中得出的。**当然，你也可以自定义量化表。**

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

JPEG 的标准亮度量化表

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

JPEG 的标准色度量化表

这两张表依据心理视觉阀制作，对 8bit 的亮度和色度的图象的处理效果不

错。

具体量化过程如下：

$$F^{Q}(u,v) = Round(\frac{F(u,v)}{\mathrm{Q}(u,v)})$$

其中 F(u,v)是 DCT 变化后的数据，Q(u,v)是量化表中的数据，Round 为四舍五入函数。

注：在解码过程中，反量化的过程如下

$$F(u,v) = F^{Q}(u,v)\mathrm{Q}(u,v)$$

量化表是控制 JPEG 压缩比的关键，这个步骤除掉了一些高频量，损失了很多细节信息。但事实上人眼对高频信号的敏感度远没有低频信号那么敏感。所以处理后的视觉损失很小，从上面的量化表也可以看出，低频部分采用了相对较短的量化步长，而高频部分则采用了相对较长的量化步长，这样做，也是为了在一定程度上得到相对清晰的图像和更高的压缩率。另一个重要原因是所有的图片的点与点之间会有一个色彩过渡的过程，而大量的图象信息被包含在低频率空间中，经过 DCT 处理后，在高频率部分，将出现大量连续的零。

以 MCU1 为例，MCU1 中，有 Y1_DU，Y2_DU，Y3_DU，Y4_DU，CrDU，CbDU，这 6 个是由采样的得到的原始数据，由上所述，对于上面这 6 个数据单元，分别进行 DCT 变换和量化，为方便起见，将 DCT 变换和量化后得到的数据依然记为

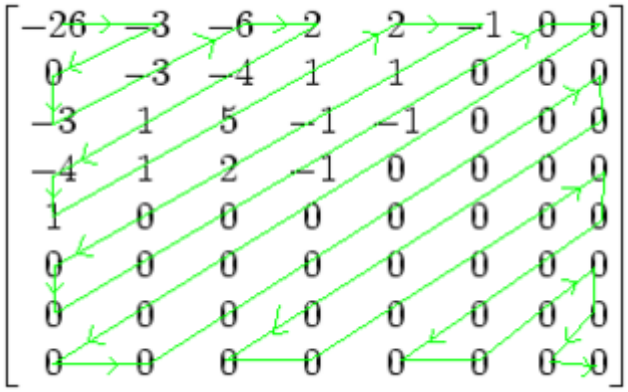
Y1_DU，Y2_DU，Y3_DU，Y4_DU，CrDU，CbDU

6、Zigzag 扫描排序

DCT 和量化均将一个 8x8 的数组变换成另一个 8x8 的数组，但是内存里所有数据都是线形存放的，如果我们一行行的存放这 64 个数字，每行的结尾的点和下行开始的点就没有什么关系，所以 JPEG 规定按如下图中的数字顺序依次保存和读取 64 个系数值。

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

这样数列里的相邻点在图片上也是相邻的了。不难发现，这种数据的扫描、保存、读取方式，是从 8*8 矩阵的左上角开始，按照英文字母 Z 的形状进行扫描的，一般将其称之为 Zigzag 扫描排序。如下图所示：



7、DC 系数的差分脉冲调制编码

在经过 DCT 和量化处理后，将一个 8x8 的数组变换成另一个 8x8 的数组。其中数组的第一个元素为 DC 系数，其余 63 个数据为 AC 分量。DC 系数有两个特点：

（1）系数的数值比较大；

（2）相邻的 8*8 图像块的 DC 系数值变化不大；

根据这两个特点，DC 系数一般采用差分脉冲调制编码 DPCM（Difference Pulse Code Modulation）。对于上述经过 DCT 和量化得到的 MCU1

Y1_DU，Y2_DU，Y3_DU，Y4_DU，CrDU，CbDU

Y1_DU，Y2_DU，Y3_DU，Y4_DU 均为 8×8 的数据块，每个数据块的第一个系数为 DC 系数，将这四个数据块的 DC 系数拿出，放在一起，即

Y1_DU(1,1)，Y2_DU(1,1)，Y3_DU(1,1)，Y4_DU(1,1)

即每个数据块的 DC 系数取出，方便起见，记上述 4 个元素构成的数组为 A[n]，其中 n=4，。则处理方式如下

D[1] = A[1]，D[2] = A[2] – A[1]，D[3] = A[3] – A[2]，D[i]=A[i]-A[i-1]，D[n]=A[n]-A[n-1]

即取每个 DC 值与前一个 DC 值的差值来进行编码（第一个数值除外）。对差值进行编码所需要的位数会比对原值进行编码所需要的位数少了很多。假设某一个 8*8 图像块的 DC 系数值为 15，而上一个 8*8 图像块的 DC 系数为 12，则两者之间的差值为 3。

注：由于我们将所有的颜色分量单元按颜色分量（Y、Cr、Cb）分类。因此 3 个颜色分量的直流变量是分开进行差分编码的（如果 C 对于一个 MCU，Cr，Cb 不止一块）。

8、DC 系数的中间格式计算

JPEG 中为了更进一步节约空间，并不直接保存数据的具体数值，而是将数据按照位数分为 16 组，保存在表里面。这也就是所谓的变长整数编码 VLI。即，第 0 组中保存的编码位数为 0，其编码所代表的数字为 0；第 1 组中保存的编码位数为 1，编码所代表的数字为-1 或者 1……，如下面的表格所示，这里，暂且称其为 VLI 编码表：

VLI 编码表

数值	组	实际编码值
0	0	
-1,1	1	0,1
-3,-2,2,3	2	00,01,10,11
-7,-6,-5,-4,4,5,6,7	3	000,001,010,011,100,101,110,111
-15,...,-8,8,...,15	4	0000,...,0111,1000,...,1111
-31,...,-16,16,...31	5	00000,...,01111,10000,...,11111
-63,...,-32,32,...63	6	000000,...,011111,100000,...,111111
-127,...,-64,64,...,127	7	0000000,...,0111111,1000000,...,1111111
-255,..,-128,128,..,255	8	...
-511,..,-256,256,..,511	9	...
-1023,..,-512,512,..,1023	10	...
-2047,..,-1024,1024,..,2047	11	...
-4095,..,-2048,2048,..,4095	12	
-8191,..,-4096,4096,..,8191	13	
-16383,..,-8192,8192,..,16383	14	
-32767,..,-16384,16384,..,32767	15	

例：对于前面提到的那个 DC 差值为 3 的数据，通过查找 VLI 可以发现，整数 3 位于 VLI 表格的第 2 组，因此，可以写成（2，3）的形式，该形式，称之为 DC 系数的中间格式。即若某一个数 m 位于上述标准的第 i 组，则该数的中间形式为（i，m），，这里的 i 代表数字 m 的编码长度为 i 位。

9、DC 系数熵编码

在得到 DC 系数的中间格式之后，为进一步压缩图象数据，提高压缩比，需要对其再进行熵编码，这里选用 Huffman 编码。JPEG 标准具体规定了两种熵编码方式：Huffman 编码和算术编码。JPEG 基本系统规定采用 Huffman 编码（因为不存在专利问题），但 JPEG 标准并没有限制 JPEG 算法必须用 Huffman 编码方式或者算术编码方式。

Huffman 编码：对出现概率大的字符分配字符长度较短的二进制编码，对出现概率小的字符分配字符长度较长的二进制编码，从而使得字符的平均编码长度最短。Huffman 编码的原理请参考数据结构中的 Huffman 树或者最优二叉树。

Huffman 编码 DC 系数时,对于亮度和色度采用不同的 Huffman 编码表。因此，对于 DC 系数，有两张 Huffman 编码表。

具体的 Huffman 编码采用查表的方式来高效地完成。然而，在 JPEG 标准中没有定义缺省的 Huffman 表，用户可以根据实际应用自由选择，也可以使用 JPEG 标准推荐的 Huffman 表。或者预先定义一个通用的 Huffman 表，也可以针对一副特定的图像，在压缩编码前通过搜集其统计特征来计算 Huffman 表的值。

对于亮度 DC 系数的中间格式(2，3)而言，数字 2 **查推荐的 DC 亮度 Huffman 表**得到 011，数字 3 通过**查找 VLI 编码表**得到其被编码为 11；

JPEG 推荐的亮度 DC 系数的 huffman(哈夫曼)码表

category	码长	码字	权值
0	2	00	0x00
1	3	010	0x01
2	3	011	0x02
3	3	100	0x03
4	3	101	0x04
5	3	110	0x05
6	4	1110	0x06
7	5	11110	0x07
8	6	111110	0x08
9	7	1111110	0x09
10	8	11111110	0xA0

权值（共 8 位）表示该直流分量数值的二进制位数。

总结：对于 DC 编码得到的元素数值 D[i]，首先在 VLI 中找到所在的组号 j，记为(j,D[i])。同时找到了 D[i]的编码，

推荐的亮度 DC 系数的 Huffman 码表的安排，可能的意思是

比如(2,3)来说，因为 2 表示位数，位数少的的数值频率可能较高，所以设置比较小的码字，从而权值即为位数。而码字的生成规则（例如加 1，往左移 1 位添 0 等）是为了前缀编码。（这个表不一定就必须要这样）。

如果根据统计的观点得到 Huffman 表，则通过如下方式得到：

建立 Huffman 表，统计位数信息，即(j,D[i])中的 j 的频率。根据频率大小进行编码。记 j 为权值，建立 Huffman 表。按照 Huffman 码字的位数从上到下进行排列，左边记其码长，右边列出其权值 j。之后对于数值 D[i]，根据数值的二进制位数即权值来匹配 j 使用的权值 j 编码。而码长的作用是为了后续的十六进制编码，并且进行权值的对应。

因此，一个完整的 Huffman 表，例表如下

序号	码字长度	码字	权值
1	2	00	0x00
2	2	01	0x01
3	2	10	0x02
4	3	110	0x07
5	4	1110	0x1e
6	5	11110	0x2e

对于色度 Cr,Cb 的 DC 系数，采用相似的方法。

10、AC 系数的行程长度编码(RLC)

量化之后的 AC 系数的特点是：63 个系数中含有很多值为 0 的系数。因此，可以采用行程编码 RLC（Run Length Coding）来更进一步降低数据的传输量。利用该编码方式，可以将一个字符串中重复出现的连续字符用两个字节来代替，

其中，第一个字节代表重复的次数，第二个字节代表被重复的字符串。例如，(4,6)就代表字符串“6666”。但是，在 JPEG 编码中，RLC 的含义就同其原有的意义略有不同。在 JPEG 编码中，假设 RLC 编码之后得到了一个（M,N）的数据对，其中 M 是两个非零 AC 系数之间连续的 0 的个数（即，行程长度），N 是下一个非零的 AC 系数的值。**采用这样的方式进行表示，是因为 AC 系数当中有大量的 0，而采用 Zigzag 扫描也会使得 AC 系数中有很多连续的 0 的存在**，如此一来，便非常适合于用 RLC 进行编码。

例，现有一个字符串，如下所示：

57,45,0,0,0,0,23,0,-30,-8,0,0,1,000.....

经过 RLC 之后，将呈现出以下的形式：

(0,57) ； (0,45) ； (4,23) ； (1,-30) ； (0,-8) ； (2,1) ； (0,0)

注意，如果 AC 系数之间连续 0 的个数超过 16，则用一个扩展字节(15,0)来表示 16 连续的 0。

对于之前经过 DCT 和量化得到的 MCU1

Y1_DU，Y2_DU，Y3_DU，Y4_DU，CrDU，CbDU

Y1_DU，Y2_DU，Y3_DU，Y4_DU 均为 8×8 的数据块，每个数据块的第 2-64 个系数为 AC 系数，AC 编码中以 8×8 的数据单元进行编码。从而先编码 Y1_DU 中 AC 系数，再编码 Y2_DU 中 AC 系数，直到 Y4_DU 中 AC 系数。

11、AC 系数的中间格式

根据前面提到的 **VLI 表格**，对于前面的字符串：

(0,57) ； (0,45) ； (4,23) ； (1,-30) ； (0,-8) ； (2,1) ； (0,0)

只处理每对数右边的那个数据，对其进行 VLI 编码：查找上面的 VLI 编码表格，可以发现，57 在第 6 组当中，因此，可以将其写成(0,6),57 的形式，该形式，称之为 AC 系数的中间格式。

同样的(0,45)的中间格式为：(0,6),45；(1,-30)的中间格式为：(1,5),-30；

12、AC 系数熵编码

在得到 AC 系数的中间格式之后，为进一步压缩图象数据，有必要对其进行熵编码。JPEG 标准具体规定了两种熵编码方式：Huffman 编码和算术编码。JPEG 基本系统规定采用 Huffman 编码（因为不存在专利问题），但 JPEG 标准并没有限制 JPEG 算法必须用 Huffman 编码方式或者算术编码方式。

Huffman 编码 AC 系数时，对于亮度和色度采用不同的 Huffman 编码表。因此，对于 AC 系数，也有两张 Huffman 编码表。

具体的 Huffman 编码采用查表的方式来高效地完成。然而，在 JPEG 标准中没有定义缺省的 Huffman 表，用户可以根据实际应用自由选择，也可以使用 JPEG 标准推荐的 Huffman 表。或者预先定义一个通用的 Huffman 表，也可以针对一副特定的图像，在压缩编码前通过搜集其统计特征来计算 Huffman 表的值。

举例说明：下面我们举例来说明 8*8 图像子块经过 DCT 及量化之后的处理过程：

假设一个图像块经过量化以后得到以下的系数矩阵：

15	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

显然第一个是 DC 系数，之前已经考虑过了，下面通过 Zigzag 扫描，遇到第一个非 0 的 AC 系数为-2，遇到 0 的个数为 1，AC 系数经过 RLC 编码后可表示为（1，-2），通过查找 VLI 表发现，-2 在第 2 组，因此，该 AC 系数的中间格式为（1,2）-2；

其余的点类似，可以求得这个 8*8 子块熵编码中 AC 系数的中间格式为

（1，2）（-2），（0，1）（-1），（0，1）（-1），（0，1）（-1），（2，1）（-1），（EOB）（0，0）

对于 AC 系数的中间格式(1,2)(-2)而言，(1,2)查推荐的 AC 亮度 Huffman 表得到 11011，-2 通过查找 VLI 编码表得到其被编码为 01；

对于 AC 系数的中间格式(0,1)(-1)而言，(0,1)查推荐的 AC 亮度 Huffman 表得到 00，数字-1 通过查找 VLI 编码表得到其被编码为 0；

对于 AC 系数的中间格式(2,1)(-1)而言，（2,1）查推荐的 AC 亮度 Huffman 表得到 11100，数字-1 通过查找 VLI 编码表得到其被编码为 0；

对于 AC 系数的中间格式(0,0)而言，查推荐的 AC 亮度 Huffman 表得到 1010；

JPEG 推荐的亮度 AC 系数的 huffman(哈夫曼)码表

Run/ Category	Base Code	Length	Run/ Category	Base Code	Length
0/0	**1010 (= EOB)**	**4**			
0/1	00	3	8/1	11111010	9
0/2	01	4	8/2	1111111111000000	17
0/3	100	6	8/3	1111111110110111	19
0/4	1011	8	8/4	1111111110111000	20
0/5	11010	10	8/5	1111111110111001	21
0/6	111000	12	8/6	1111111110111010	22
0/7	1111000	14	8/7	1111111110111011	23
0/8	1111110110	18	8/8	1111111110111100	24
0/9	1111111110000010	25	8/9	1111111110111101	25
0/A	1111111110000011	26	8/A	1111111110111110	26
1/1	1100	5	9/1	111111000	10
1/2	111001	8	9/2	1111111110111111	18
1/3	1111001	10	9/3	1111111111000000	19
1/4	111110110	13	9/4	1111111111000001	20
1/5	11111110110	16	9/5	1111111111000010	21
1/6	1111111110000100	22	9/6	1111111111000011	22
1/7	1111111110000101	23	9/7	1111111111000100	23
1/8	1111111110000110	24	9/8	1111111111000101	24
1/9	1111111110000111	25	9/9	1111111111000110	25
1/A	1111111110001000	26	9/A	1111111111000111	26
2/1	11011	6	A/1	111111001	10
2/2	11111000	10	A/2	1111111111001000	18
2/3	1111110111	13	A/3	1111111111001001	19
2/4	1111111110001001	20	A/4	1111111111001010	20
2/5	1111111110001010	21	A/5	1111111111001011	21
2/6	1111111110001011	22	A/6	1111111111001100	22
2/7	1111111110001100	23	A/7	1111111111001101	23
2/8	1111111110001101	24	A/8	1111111111001110	24
2/9	1111111110001110	25	A/9	1111111111001111	25
2/A	1111111110001111	26	A/A	1111111111010000	26
3/1	111010	7	B/1	111111010	10
3/2	111110111	11	B/2	1111111111010001	18
3/3	11111110111	14	B/3	1111111111010010	19
3/4	1111111110010000	20	B/4	1111111111010011	20
3/5	1111111110010001	21	B/5	1111111111010100	21
3/6	1111111110010010	22	B/6	1111111111010101	22
3/7	1111111110010011	23	B/7	1111111111010110	23
3/8	1111111110010100	24	B/8	1111111111010111	24
3/9	1111111110010101	25	B/9	1111111111011000	25
3/A	1111111110010110	26	B/A	1111111111011001	26
4/1	111011	7	C/1	1111111010	11
4/2	1111111000	12	C/2	1111111111011010	18
4/3	1111111110010111	19	C/3	1111111111011011	19
4/4	1111111110011000	20	C/4	1111111111011100	20
4/5	1111111110011001	21	C/5	1111111111011101	21
4/6	1111111110011010	22	C/6	1111111111011110	22
4/7	1111111110011011	23	C/7	1111111111011111	23
4/8	1111111110011100	24	C/8	1111111111100000	24
4/9	1111111110011101	25	C/9	1111111111100001	25
4/A	1111111110011110	26	C/A	1111111111100010	26

因此，最后这个 8*8 子块亮度 AC 系数信息压缩后的数据流为 1101101，000，000，000，111000，1010。
这个 8*8 子块亮度总的信息压缩后的数据流为 01111，1101101，000，000，000，111000，1010。总共 31 比特，其压缩比是 64*8/31=16.5，大约每个像素用半个比特。

若不采用推荐的，可以采用统计的观点得到 Huffman 表。与 DC 系数相似，可以得到类似的，如下表所示，其中权值的高 4 位表示当前数值前面有多少个连续的零，低 4 位表示该交流分量数值的二进制位数，也就是接下来需要读入的位数。

序号	码长长度	码字	权值
1	2	00	0x00
2	2	01	0x01
3	3	100	0x11
4	3	101	0x02
5	5	11000	0x21
6	5	11001	0x03
7	5	11010	0x31
8	5	11011	0x41
9	5	11100	0x12
10	6	111010	0x51
11	7	1110110	0x61
12	7	1110111	0x71
13	7	1111000	0x81
14	7	1111001	0x91
15	7	1111010	0x22
16	7	1111011	0x13
17	8	11111000	0x32

例如对于 AC 系数的中间格式(2,1)(1)而言，（2,1）查上述 AC 亮度 Huffman 表得到 11100，数字-1 通过查找 VLI 编码表得到其被编码为 0；

至此，我们以针对 MCU1，描述了 AC 系数和 DC 系数的编码方式，这样对于 Y1_DU，Y2_DU，Y3_DU，Y4_DU，CrDU，CbDU 这 6 个数据单元来说，均进行了编码。

以 Y1_DU 为例，在真正的数据部分中，先放 Y1_DU 的 DC 系数的差分编码，紧接着的是 AC 系数的编码，
之后，紧接着的是 Y2_DU 的编码，同理，先放 Y2_DU 的 DC 系数的差分编码，紧接着的是 AC 系数的编码，直到 Y4_DU。之后紧接着的是 CrDU 的编码，先放 CrDU 的 DC 系数的差分编码（实际上没有差分编码了，因为 Cr 只有一块），紧接着的是 CrDU 的 AC 系数的编码，之后紧接着的是 CbDU 的编码，先放 CbDU 的 DC 系数的差分编码（实际上没有差分编码了，因为 Cb 只有一块），紧接着的是 CbDU 的 AC 系数的编码。

在 MCU1 的所有块都排列完之后，紧接着排列 MCU2 的编码数据，与 MCU1 完全类似，不再赘述，直到排完最后一个 MCU 块。（实际上，排完最后一个 MCU 块

的编码数据后，文件就编码结束了（当然最后还有两个字节的结束标记符））

以上我们介绍了 JPEG 压缩的原理，几乎所有传统的压缩方法在这里都用到了。这几种方法的结合正是产生 JPEG 高压缩比的原因。顺便说一下，该标准是 JPEG 小组从很多种不同中方案中比较测试得到的，并非空穴来风。

13、理论部分结束

至此，jpg 编码理论基本结束，JPEG 解码程序流程图如下

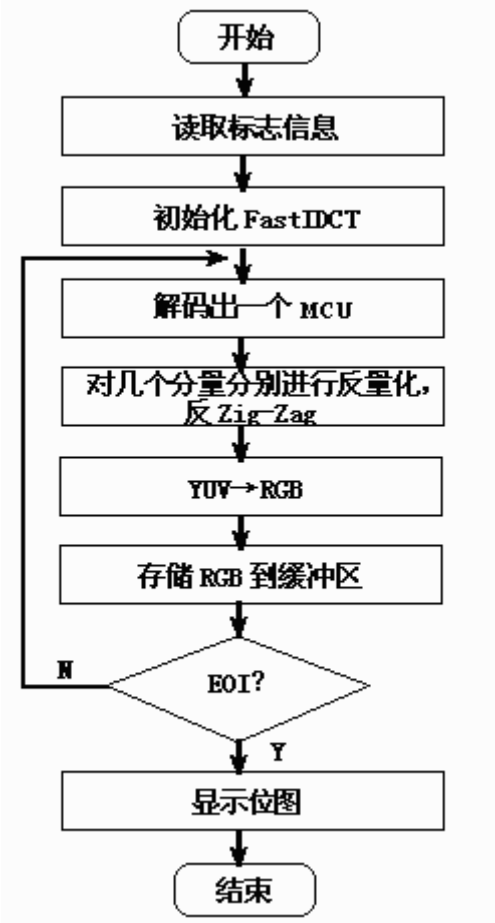


图 JPEG 解码程序流程图

以下就是如何将这些压缩得到的数据、Huffman 表，图片基本信息等等采用集中在一起，按照一定次序按照十六进制进行编码。

二、JPEG 文件格式介绍

JPEG 文件使用的数据存储方式有多种。最常用的格式称为 JPEG 文件交换格式（JPEG File Interchange Format，JFIF）。而 JPEG 文件大体上可以分成两个部分：标记码(Tag)和压缩数据。

标记码由两个字节构成，其前一个字节是固定值 0xFF，后一个字节则根据不同意义有不同数值。在每个标记码之前还可以添加数目不限的无意义的 0xFF 填充，也就说连续的多个 0xFF 可以被理解为一个 0xFF，并表示一个标记码的开始。而在一个完整的两字节的标记码后，就是该标记码对应的压缩数据流，记录了关

于文件的诸种信息。

常用的标记有 SOI、APP0、DQT、SOF0、DHT、DRI、SOS、EOI。

注意，SOI 等都是标记的名称。在文件中，标记码是以标记代码形式出现。例如 SOI 的标记代码为 0xFFD8，即在 JPEG 文件中的如果出现数据 0xFFD8，则表示此处为一个 SOI 标记。

本文附录列出一张完整的 JPEG 定义的标记表，供读者查阅。这里仅列出几个常用标记的标记代码、占用字节长度和表示的意义。

- **SOI**，Start of Image，图像开始
  - 标记代码　　2 字节　　固定值 FFD8
- **APP0**，Application，应用程序保留标记 0
  - 标记代码　　2 字节　　固定值 FFE0
  - 包含 9 个具体字段：

字段	含义	字节数	属性
1	数据长度	2	1-9 个字段的总长度
2	标识符	5	固定值 4A46494600，即字符串“JFIF0”
3	版本号	2	一般为 0102，表示 JFIF 的版本号 1.2
4	X 和 Y 的密度单位	1	只有三个值可选 0：无单位；1：点数/英寸；2：点数/厘米
5	X 方向像素密度	2	取值范围未知
6	Y 方向像素密度	2	取值范围未知
7	缩略图水平像素数目	1	取值范围未知
8	缩略图垂直像素数目	1	取值范围未知
9	缩略图 RGB 位图	3 的倍数	缩略图 RGB 位图数据

注：上述第 9 个标记段可以包含图像的一个微缩版本，存为 24 位的 RGB 像素。如果没有微缩图像（这种情况更常见），则字段⑦“缩略图水平像素数目”和字段⑧“缩略图垂直像素数目”的值均为 0。

- **APPn**，Application，应用程序保留标记 n，其中 n=1～15(任选)
  - 标记代码　　2 字节　　固定值 FFE1~0xFFF
  - 包含 2 个具体字段：

字段	含义	字节数	属性
1	数据长度	2	1-2 个字段的总长度
2	详细信息	2	内容不定

例如，Adobe Photoshop 生成的 JPEG 图像中就用了 APP1 和 APP13 两个标记段分别存储了一幅图像的副本。

- **DQT**，Define Quantization Table，定义量化表
  - 标记代码　　2 字节　　固定值 FFDB
  - 包含 9 个具体字段：

字段	含义	字节数	属性		
1	数据长度	2	字段 1 和多个字段 2 的总长度		
2	量化表	数据长度-2 字节	精度及量化表 ID	1 字节	高 4 位：精度，只有两个可选值 0：8 位；1：16 位 低 4 位：量化表 ID，取值范围为 0～3
			表项	(64×(精度+1))字节	例如 8 位精度的量化表，其表项长度为 64×（0+1）=64 字节

本标记段中，字段 2 可以重复出现，表示多个量化表，但最多只能出现 4 次。

- **SOF0**，Start of Frame，帧图像开始
  - 标记代码　　2 字节　　固定值 FFC0
  - 包含 9 个具体字段：

字段	含义	字节数	属性		
1	数据长度	2	1-6 个字段的总长度		
2	精度	1	每个数据样本的位数，通常是 8 位		
3	图像高度	2	图像高度（单位：像素）		
4	图像宽度	2	图像宽度（单位：像素）		
5	颜色分量数	3	只有 3 个数值可选，1：灰度图；3：YCrCb 或 YIQ；4：CMYK，而 JFIF 中使用 YCrCb，故这里颜色分量数恒为 3		
6	颜色分量信息	颜色分量数×3 字节，通常为 9 字节	颜色分量 ID	1 字节	
			水平/垂直采样因子	1 字节	高 4 位：水平采样因子 低 4 位：垂直采样因子
			量化表	1 字节	当前分量使用的量化表的 ID

本标记段中，字段 6 应该重复出现，有多少个颜色分量（字段⑤），就出现多少次（一般为 3 次）。

DHT，Difine Huffman Table，定义哈夫曼表

- 标记代码　　2 字节　　固定值 FFC4
- 包含 2 个具体字段：

字段	含义	字节数	属性		
1	数据长度	2	字段 1 和多个字段 2 的总长度		
2	哈夫曼表	数据长度-2 字节	表 ID 及表类型	1 字节	高 4 位：类型，只有两个值可选，0：DC 直流； 1：AC 交流 低 4 位：哈夫曼表 ID，注意，DC 表和 AC 表分开编码
			不同位数的码字数量	16 字节	
			编码内容	16 个不同位数的码字数量之和（字节）	

本标记段中，字段②可以重复出现（一般 4 次），也可以只出现 1 次。例如，Adobe Photoshop 生成的 JPEG 图片文件中只有 1 个 DHT 标记段，里边包含了 4 个哈夫曼表；而 Macromedia Fireworks 生成的 JPEG 图片文件则有 4 个 DHT 标记段，每个 DHT 标记段只有一个哈夫曼表。

- **SOS**，Start of Scan，扫描开始 12 字节

- 标记代码　　2 字节　　固定值 FFDA

包含 4 个具体字段

字段	含义	字节数	属性		
1	数据长度	2	字段 1-4 的总长度		
2	颜色分量数	1	应该和 SOF 中的字段 5 的值相同，即： 1：灰度图是；3： YCrCb 或 YIQ；4：CMYK。 而 JFIF 中使用 YCrCb，故这里颜色分量数恒为 3		
3	颜色分量信息	1	颜色分量 ID	1 字节	
			直流/交流系数表号	1 字节	高 4 位：直流分量使用的哈夫曼树编号 低 4 位：交流分量使用的哈夫曼树编号

4	压缩图像数据	3	003F00

本标记段中，字段 3 应该重复出现，有多少个颜色分量（字段 2），就出现多少次（一般为 3 次）。**本段结束后，紧接着就是真正的图像信息了**。图像信息直至遇到一个标记代码就自动结束，一般就是以 EOI 标记表示结束。

- **EOI**，End of Image，图像结束 2 字节
  - 标记代码　　2 字节　　固定值 FFD9

这里补充说明一下，由于在 JPEG 文件中 0xFF 具有标志性的意思，所以在压缩数据流(真正的图像信息)中出现 0xFF，就需要作特别处理。具体方法是，在数据 0xFF 后添加一个没有意义的 0x00。换句话说，如果在图像数据流中遇到 0xFF，应该检测其紧接着的字符，如果是

1）0x00，则表示 0xFF 是图像流的组成部分，需要进行译码；

2）0xD9，则与 0xFF 组成标记 EOI，则图像流结束，同时图像文件结束；

3）0xD0~0xD7,则组成 RSTn 标记，则要忽视整个 RSTn 标记，即不对当前 0xFF 和紧接的 0xDn 两个字节进行译码，并按 RST 标记的规则调整译码变量；

3）0xFF，则忽视当前 0xFF，对后一个 0xFF 再作判断；

4）其他数值，则忽视当前 0xFF，并保留紧接的此数值用于译码。

下面来详细讲述 JPEG 文件的解码过程。

1．读入文件的相关信息

按照上述的 JPEG 文件数据存储方式，把要解码的文件的相关信息一一读出，为接下来的解码工作做好准备。参考方法是，设计一系列的结构体对应各个标记，并存储标记内表示的信息。**其中图像长宽、多个量化表和哈夫曼表、水平/垂直采样因子等多项信息比较重要。**以下给出读取过程中的两个问题。

1）整个文件的大体结构

JFIF 格式的 JPEG 文件(*. jpg)的一般顺序为：

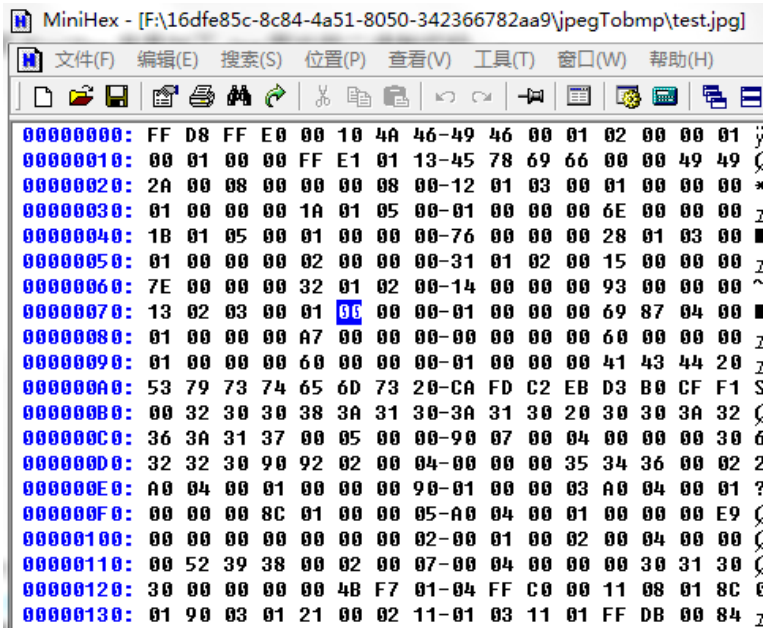
标识符	属性
SOI	FFD8
APPO	FFE0
[APPn]（可选）	FFEn
DQT	FFDB
SOFO	FFC0
DHT	FFC4
SOS	FFDA
压缩数据	真正的压缩后的图像信息
EOI	FFDA

2）字的高低位问题

JPEG 文件格式中，一个字（16 位）的存储使用的是 Motorola 格式，而不是 Intel 格式。也就是说，一个字的高字节（高 8 位）在数据流的前面，低字节（低 8 位）在数据流的后面，与平时习惯的 Intel 格式不一样。

为了更形象地找到图片的编码方式，可以下载软件 MiniHex，该软件可以查看任意文件的十六进制编码。

下面用 MiniHex 查看如下 jpg 图片的十六进制代码，



该图像十六进制编码

其十六进制编码开头的 20 个字节

FF D8 FF E0 00 10 4A 46 49 46 00 01 02 00 00 01 00 01 00 00

可以看到，其包括两个标记

前两个字节 FF D8 为 SOI 标记，表示文件开始了；

紧接着的两个字节 FF E0，表示 APP0 字段从这开始；

紧接着的两个字节 00 10 表示 APP0 中字段 1-9 的总长度，为 16 个字节；
紧接着的五个字节 4A 46 49 46 00 表示字符“JFIF0”；
紧接着的两个字节 01 02，表示 JFIF 的版本号 1.2；
紧接着的一个字节 00 表示 X 和 Y 的密度单位（0：无单位；1：点数/英寸；2：点数/厘米），即表示这张图无单位；
紧接着的两个字节 00 01 表示 X 方向像素密度，密度为 1；
紧接着的两个字节 00 01 表示 Y 方向像素密度，密度为 1；
紧接着的一个字节 00 表示缩略图水平像素数目 为 0；
紧接着的一个字节 00 表示缩略图垂直像素数目 为 0；
最后的这两个字节也说明了没有微缩图像。

如下图，在 APP0 字段后，紧接着的两个字节为 FF E1，即表示 APP1 字段的开始。

```
FF D8 FF E0 00 10 4A 46-49 46 00 01 02 00 00 01
00 01 00 00 FF E1 01 13-45 78 69 66 00 00 49 49
2A 00 08 00 00 00 08 00-12 01 03 00 01 00 00 00
01 00 00 00 1A 01 05 00-01 00 00 00 6E 00 00 00
1B 01 05 00 01 00 00 00-76 00 00 00 28 01 03 00
01 00 00 00 02 00 00 00-31 01 02 00 15 00 00 00
7E 00 00 00 32 01 02 00-14 00 00 00 93 00 00 00
13 02 03 00 01 00 00 00-01 00 00 00 69 87 04 00
01 00 00 00 A7 00 00 00-00 00 00 00 60 00 00 00
01 00 00 00 60 00 00 00-01 00 00 00 41 43 44 20
53 79 73 74 65 6D 73 20-CA FD C2 EB D3 B0 CF F1
00 32 30 30 38 3A 31 30-3A 31 30 20 30 30 3A 32
36 3A 31 37 00 05 00 00-90 07 00 04 00 00 00 30
32 32 30 90 92 02 00 04-00 00 00 35 34 36 00 02
A0 04 00 01 00 00 00 90-01 00 00 03 A0 04 00 01
00 00 00 8C 01 00 00 05-A0 04 00 01 00 00 00 E9
00 00 00 00 00 00 00 02-00 01 00 02 00 04 00 00
00 52 39 38 00 02 00 07-00 04 00 00 00 30 31 30
30 00 00 00 00 4B F7 01-04 FF C0 00 11 08 01 8C
```

APP1 字段

紧接着的两个字节为 01 13 表示字段的长度为 256+16+3 个字节，即 275 个字节。上述黑框部分即为 APP1 字段的所有内容（另外再加上标志符 FF E1）。

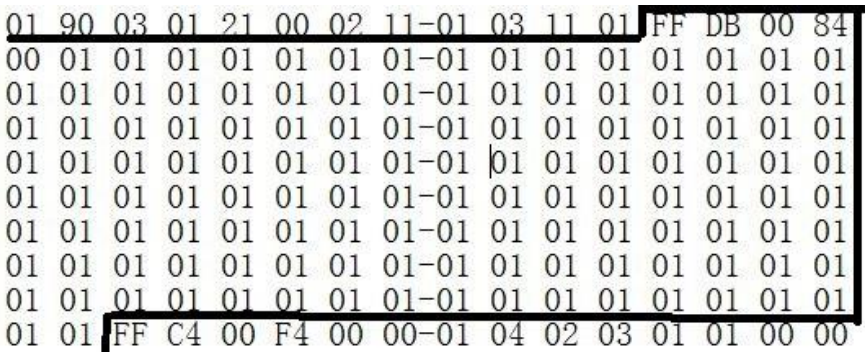
在 APP1 字段之后，紧接着的两个字节为 FF C0，表示 SOF0 字段开始，其整个字段如下

FF C0 00 11 08 01 8C 01 90 03 01 21 00 02 11-01 03 11 01

紧接着的两个字节为 00 11 表示字段的长度为 17 个字节。
紧接着的一个字节为 08 表示每个数据样本的位数。
紧接着的两个字节为 01 8C，表示图像的宽度为 256+8×16+12 个像素，即 396 个像素；
紧接着的两个字节为 01 90，表示图像的宽度为 256+9×16 个像素，即 400 个像素；
紧接着的一个字节为 03，表示颜色分量数为 YCrCb。之后的为颜色分量信息
紧接着的三个字节为 01 21 00，其中第一个字节表示颜色分量 ID（01 代表 Y），第二个字节为 21，高四位为 2，表示水平采样因子为 2，低四位为 1，表

示垂直采样因子为 1。第三个字节为 00，表示该颜色分量使用的量化表 ID 号为 00；
紧接着的三个字节为 02 11 01，其中第一个字节表示颜色分量 ID（02 代表 Cr），第二个字节为 11，高四位为 1，表示水平采样因子为 1，低四位为 1，表示垂直采样因子为 1。第三个字节为 01，表示该颜色分量使用的量化表 ID 号为 01；
紧接着的三个字节为 03 11 01，其中第一个字节表示颜色分量 ID（03 代表 Cb），第二个字节为 11，高四位为 1，表示水平采样因子为 1，低四位为 1，表示垂直采样因子为 1。第三个字节为 01，表示该颜色分量使用的量化表 ID 号为 01；

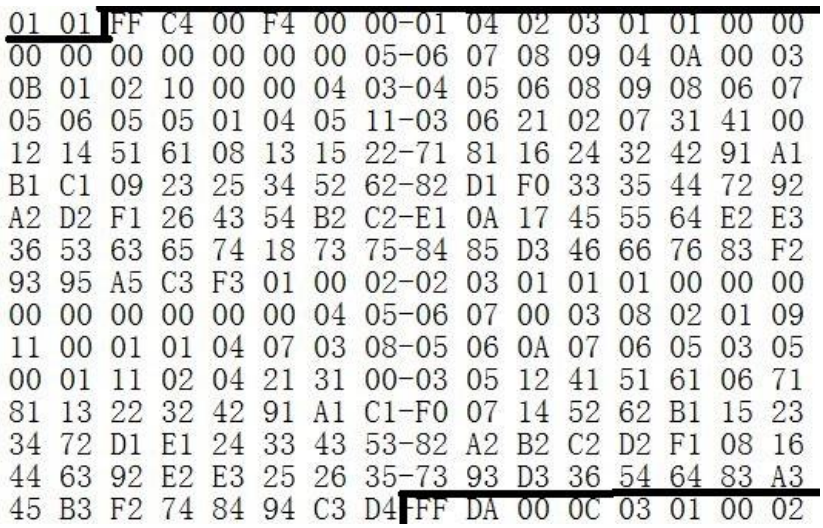
如下图，在 SOF0 字段后，紧接着的两个字节为 FF DB，即表示 DQT 字段（定义量化表）的开始。



DQT 字段

紧接着的为量化表数据。
紧接着的两个字节为 00 84，表示字段的长度为 8×16+4 个字节，即 132 个字节。
紧接着的一个字节为 00，其中高字位为 0，表示精度为 8 位，低 4 位为 0，表示，表示量化表的 ID 为 0 。紧跟着的便是真正的量化表数据，ID 为 0 的量化表长度为(64×(精度+1))字节，即 64×（0+1）=64 字节（这里的精度记为 0，表示 8 位）
读取 64 位字节后，紧接着的字节为 01，其中高字位为 0，表示精度为 8 位，低 4 位为,1，表示，表示量化表的 ID 为 1，ID 为 1 的量化表长度为(64×(精度+1))字节，即 64×（0+1）=64 字节。
注：可以验证 132 = 2 + 1 + 64 + 1 + 64。
从该字段可知，Y 分量的量化表的所有数值和 Cr，Cb 分量的量化表的所有数值均为 1。

如下图，在 DQT 字段后，紧接着的两个字节为 FF C4，即表示 DHT 字段（定义哈夫曼表）的开始。



DHT 字段

紧接着的为哈夫曼表数据。

紧接着的两个字节为 00 F4，表示字段的长度为 15×16+4 个字节，即 244 个字节。

紧接着的为真正的哈夫曼表数据，数据长度为 244-2 个字节，即 242 个字节。

紧接着的一个字节为 00，表示 DC 直流 0 号表。

紧接着的 16 字节，为不同位数的码字数量，从上图中，即

00 01 04 02 03 01 01 00 00 00 00 00 00 00 00 00

其中第一个字节为 00，表示哈夫曼码字长为 1 位的码字数量为 0 个，第二个字节为 01，表示哈夫曼码字长为 2 位的码字数量为 1 个，以此类推。上面这 16 个字节的数据所表示的个数相加，即 01+04+02+03+01+01=12 个字节，即继续往下读 12 个字节的数据，这十二个字节记为哈夫曼表的权值。从上图中，即为

05 06 07 08 09 04 0A 00 03 0B 01 02

从而可以得到下表，

DC 直流 0 号表

序号	码字长度	码字	权值
1	2	00	0x05
2	3	010	0x06
3	3	011	0x07
4	3	100	0x08
5	3	101	0x09
6	4	1100	0x04
7	4	1101	0x0A
8	5	11100	0x00
9	5	11101	0x03
10	5	11110	0x0B
11	6	111110	0x01
12	7	111110	0x02

解码过程中，对于上述哈夫曼表格中码字的获得，在本小节最后有说明。

紧接着的一个字节为 10，表示 AC 交流 0 号表。

紧接着的 16 字节，为不同位数的码字数量，从上图中，即 00 00 04 03 04 05 06 08 09 08 06 07 05 06 05 05，其中第一个字节为 00，表示哈夫曼码字长为 1 位的码字数量为 0 个，第二个字节为 00，表示哈夫曼码字长为 2 位的码字数量为 0 个，以此类推。上面这 16 个字节的数据所表示的个数相加，即 04+03+04+05+06+08+09+06+07+05+06+05+05=73 个字节，即继续往下读 73 个字节的数据，这 73 个字节即为哈夫曼表的权值。从上图中，即为

01 04 05 11 03 06 21 02 07 31 41 00 12 14 51 61 08 13 15 22-71 81 16 24 32 42 91 A1 B1 C1 09 23 25 34 52 62 82 D1 F0 33 35 44 72 92 A2 D2 F1 26 43 54 B2 C2 E1 0A 17 45 55 64 E2 E3 36 53 63 65 74 18 73 75 84 85 D3 46 66 76 83 F2 93 95 A5 C3 F3

从而可以得到下表，

AC 直流 0 号表

序号	码字长度	码字	权值
1	3	000	0x01
2	3	001	0x04
3	3	010	0x05
4	3	011	0x11
5	4	1000	0x03
6	4	1001	0x06
7	4	1010	0x21
8	5	10110	0x02
9	5	10111	0x07
10	5	11000	0x31
……	……	…..	…..
73	7	…….	0xF3

紧接着的一个字节为 01，表示 DC 交流 1 号表。

紧接着的 16 字节，为不同位数的码字数量，从上图中，即 00 02 02 03 01 01 01 00 00 00 00 00 00 00 00 00，其中第一个字节为 00，表示哈夫曼码字长为 1 位的码字数量为 0 个，第二个字节为 02，表示哈夫曼码字长为 2 位的码字数量为 2 个，以此类推。上面这 16 个字节的数据所表示的个数相加，即 02+02+03+01+01+01=10 个字节，即继续往下读 10 个字节的数据，这 10 个字节即为哈夫曼表的权值。从上图中，即为

04 05 06 07 00 03 08 02 01 09

从而可以得到下表，

DC 直流 1 号表

序号	码字长度	码字	权值
1	2	00	0x04
2	2	01	0x05
3	3	100	0x06
4	3	101	0x07
5	4	1100	0x00
6	4	1101	0x03
7	4	1110	0x08

8	5	11110	0x02
9	6	111110	0x01
10	7	1111110	0x09

紧接着的一个字节为 11，表示 AC 交流 1 号表。
紧接着的 16 字节，为不同位数的码字数量，从上图中，即
00 01 01 04 07 03 08 05 06 0A 07 06 05 03 05 00，其中第一个字节为 00，表示哈夫曼码字长为 1 位的码字数量为 0 个，第二个字节为 01，表示哈夫曼码字长为 2 位的码字数量为 1 个，以此类推。上面这 16 个字节的数据所表示的个数相加，即 01+01+04+07+03+08+05+06+0A+07+06+07+06+05+03+05=71 个字节，即继续往下读 71 个字节的数据，这 71 个字节即为哈夫曼表的权值。从上图中，即为
01 11 02 04 21 31 00 03 05 12 41 51 61 06 71 81 13 22 32 42 91 A1 C1 F0 07 14 52 62 B1 15 23 34 72 D1 E1 24 33 43 53 82 A2 B2 C2 D2 F1 08 16 44 63 92 E2 E3 25 26 35 73 93 D3 36 54 64 83 A3 45 B3 F2 74 84 94 C3 D4
从而可以得到下表，

AC 直流 1 号表

序号	码字长度	码字	权值
1	2	00	0x01
2	3	010	0x11
3	4	0110	0x02
4	4	0111	0x04
5	4	1000	0x21
6	4	1001	0x31
7	5	10101	0x00
8	5	10110	0x03
9	5	10111	0x05
10	5	11000	0x12
……	……	…..	…..
73	7	…….	0xD4

如上图，在 DHT 字段后，紧接着的两个字节为 FF DA，即表示 SOS 字段（定义哈夫曼表）的开始。紧接着的为哈夫曼表数据。
即为 FF DA 00 0C 03 01 00 02 11 03 11 00 3F 00
紧接着的两个字节为 00 0C，表示字段的长度为 12 个字节，
紧接着的一个字节为 03 ，表示颜色分量为 YCrCb。之后的为颜色分量信息
紧接着的二个字节为 01 00，其中第一个字节表示颜色分量 ID（01 代表 Y），第二个字节为 00，高四位为 0，表示直流分量使用的哈夫曼树编号，低四位为 0，表示交流分量使用的哈夫曼树编号。
紧接着的三个字节为 02 11，其中第一个字节表示颜色分量 ID（02 代表 Cr），第二个字节为 11，高四位为 1，表示直流分量使用的哈夫曼树编号，低四位为 1，表示交流分量使用的哈夫曼树编号。

紧接着的三个字节为 03 11，其中第一个字节表示颜色分量 ID（03 代表 Cb），第二个字节为 11，高四位为 1，表示直流分量使用的哈夫曼树编号，低四位为 1，表示交流分量使用的哈夫曼树编号。
紧接着的为三个字节的固定数值 00 3F 00。

在这三个字节之后，便是真正的图像压缩数据了，在第一部分的第 12 小节最后，已经给出了真正图像压缩数据的排列格式了，不再赘述。

哈夫曼树中码字的得到：
在读出哈夫曼表的数据后，就要建立哈夫曼树。具体方法为：

1）第一个码字必定为 0。
如果第一个码字位数为 1，则码字为 0；
如果第一个码字位数为 2，则码字为 00；
如此类推。

2）从第二个码字开始，
如果它和它前面的码字位数相同，则当前码字为它前面的码字加 1；
如果它的位数比它前面的码字位数大，则当前码字是前面的码字加 1 后再在后边添若干个 0，直至满足位数长度为止。

b）举例说明
继续以上边的例子说明问题。

- 由于没有 1 位的码字，所以第一个码字的位数为 2，即码字为 00；
- 由于 2 位的码字有两个，所以第二个码字位数仍为 2，即码字为 00+1=01；
- 第三个码字为 3 位，比第二个码字长 1 位，所以第三个码字为：01+1=10，然后再添 1 个“0”，得 100；
- ……

参考文献：

1、 Visual C++ 实现 MPEG-JPEG 编解码技术
http://download.csdn.net/detail/zhangyutangde/6494747
2、数字图像处理编程入门（吕风军）.chm
http://vdisk.weibo.com/s/I2lvLUL_SDn
3、THE JPEG Standard，Jiun-De Huang
http://disp.ee.ntu.edu.tw/meeting/%E4%BF%8A%E5%BE%B7/JPEG/JPEG.doc
4、JPEG 图片文件编解码详解
http://wenku.baidu.com/link?url=4Jgv4LhqtPCUKSqv1wJIwazWE103VhE5CZy0Zaykd35oF73ciV46xU_8MybghtYzNLQeDqWNqY5JWUlgYhMhto4R_beb3rOAX_5h9Q-6iDq
5、JPEG 编码解码流程
http://wenku.baidu.com/link?url=A4QX1crr32EbPtuCtLLSn5Pc-ETfgX20G9KqLBkWD7FSS0CF-tQEFy-Ct4OGY5MMNezVlFsY3007Z-zTj6mUbp203xionfjqinIG2VgA16m
6、The JPEG Still Picture Compression Standard， Gregory K.Wallace
http://www.docin.com/p-71945906.html
7、源代码：将 jpeg 图片转化为 bmp 图片的 C 代码
百度网盘地址 http://pan.baidu.com/s/1eRJ4Ogi