“Y”表示明亮度（Luminance、Luma），“U”和“V”则是色度、浓度（Chrominance、Chroma）

YUV和RGB <https://www.douban.com/note/76361504/>

联合图像专家组（JPEG）标准[1,2,3]定义了一组针对连续色调静止图像的压缩算法。这个静止图像压缩算法同样也可以应用于视频压缩，把每一帧都当作一个独立的静态图像来进行压缩，然后再按次序进行传输。这样一种视频编码通常被称作运动JPEG(Motion-JPEG)

a JPEG的概况 b 描述RTP所支持的JPEG的子集 c 将JPEG帧通过RTP包来传输的机制

JPEG 压缩方法 DCT算法的相关知识与原理

<http://wenku.baidu.com/link?url=it6aoMOHILwr02GPvKh4lpQaWXvSe1SpSEjiPl_RnoaEPv8jEpMwCwon4VF1ruSMkWPr1nR-Cfs2Fc6ij4ZGQhN1Ajld1_hvDcNOo2i2YuG>

JFIF 是 JPEG File Interchange Format 的缩写，也即 JPEG 文件交换格式。

JFIF 文件格式定义了一些内容是 JPEG 压缩标准未定义的，如 resolution/aspect ratio，color space 等(??就是多加了APPx字段??)，它是专门为方便用户在不同的计算机和应用程序间传输 JPEG 图像

JFIF文件就是一个JPEG码流加上一个APPO段 里面包括了 像素密度 和 密度单位 ，可以在不同的机器上以一定的尺寸显示

在JPEG中0XFF具有标记的意思

所以在压缩数据流（???真正的图像信息,或者后文说的熵编码，扫描Scan???）中，如果出现了0XFF，就需要做特别处理了。方法是，如果在图像数据流中遇到0XFF，应该检测其紧接着的字符

（1）0X00，表示0XFF是图像流的组成部分；需要进行译码；

（2）0XD9，表示与0XFF组成标记EOI，即，代表图像流的结束，同时，图像文件结束；

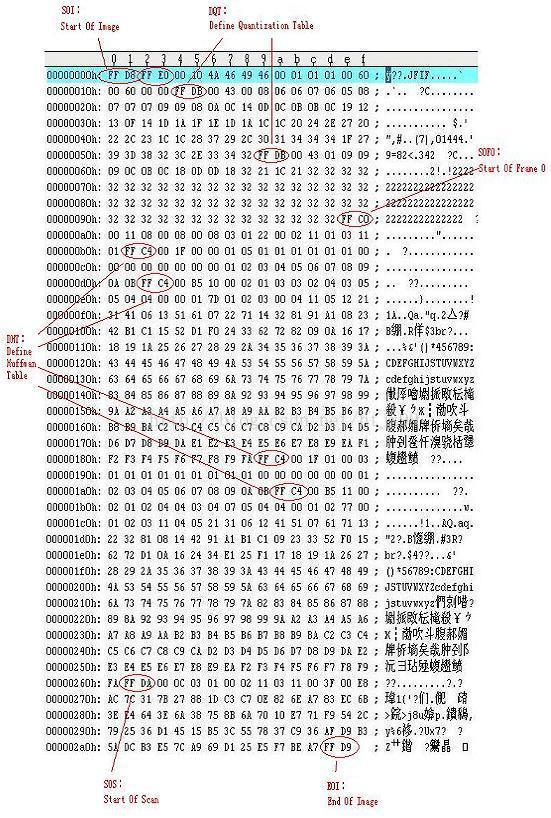
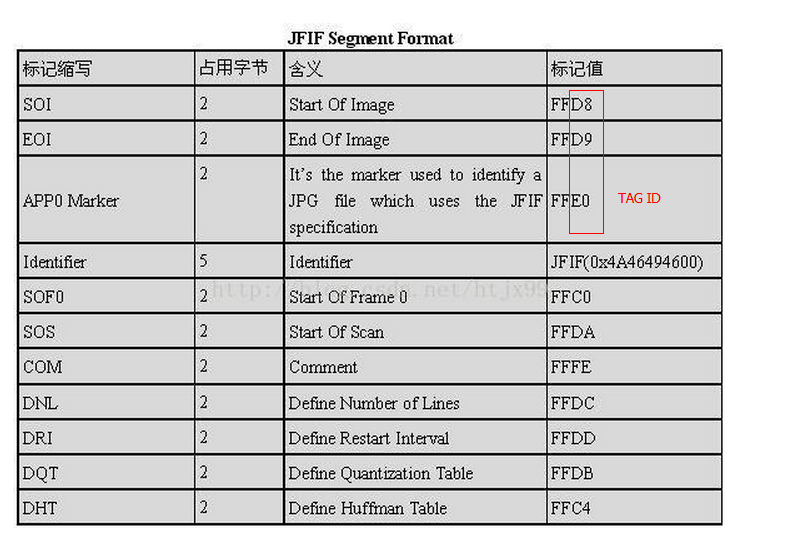
（3）0XD0--0XD7，组成RSTn标记，需要忽视整个RSTn标记，即不对当前0XFF和紧接着的0XDn两个字节进行译码，并按RST标记的规则调整译码变量；(复位标记 出现在扫描SCAN中 最小编码单元MCU的边界，由 DRI标记段来定义，Define Restart Interval)

（4）0XFF，忽略当前0XFF，对后一个0XFF进行判断；

（5）其它数值，忽然当前0XFF，并保留紧接着此数值用于译码；

？？ 标记段 有层级关系：一幅图像(SOI FFD8)用一个或多个“节”来表示，每一节（在JPEG标准中称为一帧 SOF FFC0）又进一步分成若干次扫描(FFDA)，一次扫描中包含了一系列最小编码单元（MCU FFD0~FFD7 ）???

而JPEG/JFIF文件格式则采用了big-endian格式，也就是打开文件看到的双字节的数据 就是 高位 低位

0. luma是亮度 chroma 是色度

00. JPEG标准定义了四种操作模式：顺序DCT模式，渐进DCT模式，无损模式，以及分级模式。

在不同的模式下，一幅图像用一个或多个“节”来表示，每一节（在JPEG标准中称为一帧）又进一步分成若干次扫描(Tag Name : SOS Start of Scan Tag ID: FFDA)。

在每一次扫描中，有一种到四种分量，这些分量代表着彩色信号的分量（例如“红绿蓝”或一个亮度分量和两个色差分量）。这些分量可以分开在不同的扫描中编码，也可以交织在一次单一的扫描中编码。

每一个扫描都是一个经过熵编码的比特流，位于两个头区段之间

一次扫描中包含了一系列最小编码单元（MCU），每个MCU定义了输出图像的一个小矩形块的数据。

每一帧或每一次扫描前面都有一个头区段：头信息、可选参数以及一个定位符构成了一个头区段

可选的压缩参数定义，例如量化表和哈夫曼编码表。

定位符是字节对齐的，并且不能在熵编码部分出现，这样对于扫描边界的确定就无需解析整个码流。

压缩数据有三种表示格式：交换格式、紧缩格式和表格描述格式。交换格式包含在熵编码过程中用到的所有码表的定义，紧缩模式中省略了一些码表定义，假定他们在外部定义或在前面的图像中定义。

JPEG标准并不关心组成图像的各个分量的含义或格式。诸如色彩空间和象素纵横比这些属性在JPEG码流的外部来定义。JPEG文件交换格式（JFIF）在应用标记段（APPO TagID=ffe0）提供这些额外信息

简单说来，JFIF文件就是一个JPEG码流加上一个APPO段，对于视频来说，另外还有一些参数在外部定义，比如帧率，逐行扫描还是隔行扫描等等

事实上，绝大部分JPEG硬件编解码器都只实现了其中的一个子集，也就是顺序DCT模式。典型的做法是，头区段信息由软件来解码，而用硬件来处理一个在YUV色彩空间中表示的经过熵编码的单一的扫描。

一次扫描中包含了一系列最小编码单元（MCU） 每个MCU定义了输出图像的一个小矩形块的数据。

JPEG数据中的复位标记表示解码器应当在当前点复位它的状态。如JPEG中定义的那样，复位标记是唯一的能够嵌入在熵编码码流（一个扫描中）里的标记，但他们只能够在MCU的边界处出现。

一个复位间隔是指两个复位标记之间的数据部分。每一帧的第一个复位间隔是一个例外，它们前面没有复位标记。当使用这些标记时，每一帧都由固定数目的复位间隔组成。

1. 每个FF开头是 一个TAG的开始 后面跟随TAG ID e.g C4 对应 TAG的名字是 DHT(Define Hubffman Table)

2. FF C4 对应 DHT ，TAG(FF C4)开始之后，就是这个TAG内容的长度 (TAG ID + TAG长度 + TAG 内容) ps: 长度是 TAG长度(2个字节) + TAG内容的长度

3. 在头部有 FFD8 ，表示图像的开始；结束部分有 FFD9 ，表示图像的结束

4. SOI Start of Image后面是 图片的识别信息 （TagName: APP0 jpeg码流外信息 ）如下：

APP0，Application 0, 应用程序保留标记0，标记代码为固定值0XFFE0，用2字节表示；该标记码之后包含了9个具体的字段：

（1）数据长度：2个字节，用来表示（1）--（9）的9个字段的总长度，即不包含标记代码但包含本字段；

（2）标示符：5个字节，固定值0X4A6494600，表示了字符串“JFIF0”；

（3）版本号：2个字节，一般为0X0102，表示JFIF的版本号为1.2；但也可能为其它数值，从而代表了其它版本号；

（4）X,Y方向的密度单位：1个字节，只有三个值可选，0：无单位；1：点数每英寸；2：点数每厘米；

（5）X方向像素密度：2个字节，取值范围未知；

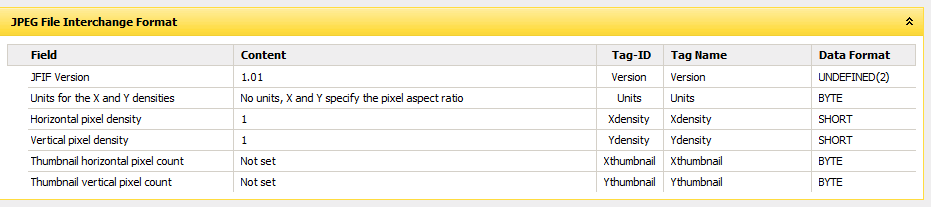
（6）Y方向像素密度：2个字节，取值范围未知；

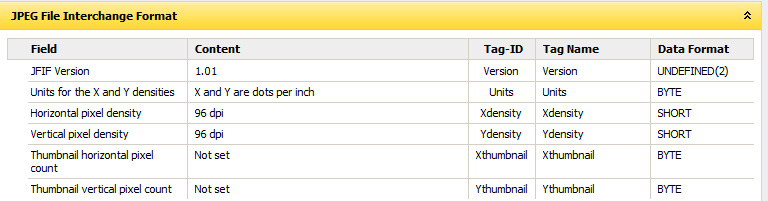
（7）缩略图水平像素数目：1个字节，取值范围未知；

（8）缩略图垂直像素数目：1个字节，取值范围未知；

（9）缩略图RGB位图：长度可能是3的倍数，保存了一个24位的RGB位图；如果没有缩略位图（这种情况更常见），则字段（7）（8）的取值均为0；



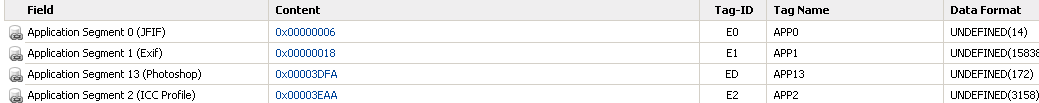


 定义了像素密度和单位 ，方便在不同的电脑上用多大的尺寸查看

5. APPn, Application n, 应用程序保留标记n(n=1---15),标记代码为2个字节，取值为0XFFE1--0XFFFF；包含了两个字段：

（1）数据长度，2个字节，表示（1）（2）两个字段的总长度；即，不包含标记代码，但包含本字段；

（2）详细信息：数据长度-2个字节，内容不定；



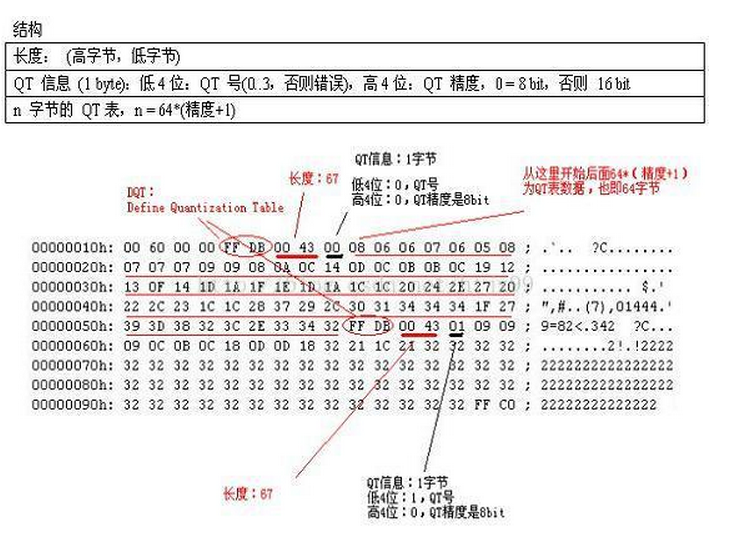
4. 两个量化表 DQT 对应的标记 FFDB

（1）数据长度：2个字节，表示（1）和多个（2）字段的总长度；即，不包含标记代码，但包含本字段； ( 除了标记ID/TagID 0xFFDB 整个标记段的长度 = 数据端长度(2个字节) + 量化表内容(精度1+表项目8x8) )

（2）量化表：数据长度-2个字节，其中包括以下内容：

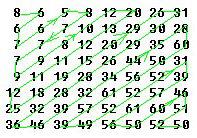
（a）精度及量化表ID，1个字节，高4位表示精度，只有两个可选值，0：8位；1:16位；低4位表示量化表ID，取值范围为0--3；

（b）表项，64\*（精度取值+1）个字节，例如，8位精度的量化表，其表项长度为64\*（0+1）=64字节；

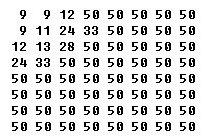


 QT 信息，占一个字节；这里是0，表示这个 QT 表编号为0，并且精度是8bit。然后后面就是64个8x8的 QT 表的各个 item

第一个 量化表的内容表示为十进制是： JPEG 亮度量化表



第二个量化表的内容为：



量化表是针对整个img 哈夫曼表是针对jpeg帧 frame

5. jpeg帧的开始(SOF0 Start of Frame 0) FFC0 后面带这个**帧的头信息** ( 字段头 + + 四个 Haffman 表对应的 FFC4  )

SOFO，Start Of Frame, 帧图像开始，标记代码为固定值0XFFC0；

包含9个具体字段：

（1）数据长度：2个字节，（1）--（6）共6个字段的总长度；即，不包含标记代码，但包含本字段；

（2）精度：1个字节，代表每个数据样本的位数；通常是8位；

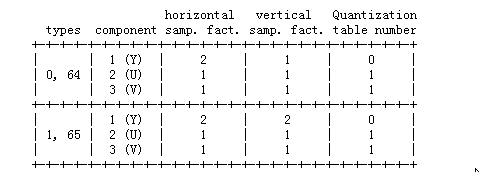
（3）图像高度：2个字节，表示以像素为单位的图像高度，如果不支持DNL就必须大于0；

（4）图像宽度：2个字节，表示以像素为单位的图像宽度，如果不支持DNL就必须大于0；

（5）颜色分量个数：1个字节，由于JPEG采用YCrCb颜色空间，这里恒定为3；( ？？扫描头字段也有 1：灰度图；3：YCrCb或YIQ；4：CMYK；??? 实际一样???)

（6）颜色分量信息：颜色分量个数\*3个字节，这里通常为9个字节；并依此表示如下一些信息：字段（6）应该重复出现3次，因为这里有3个颜色分量

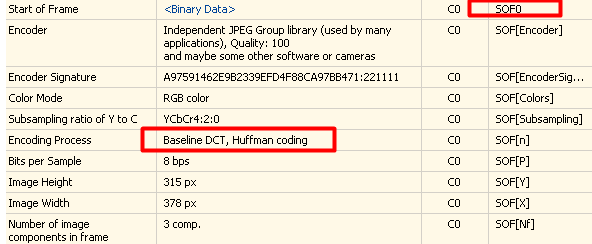
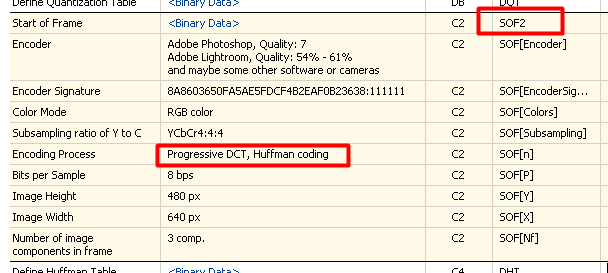
（a）颜色分量ID： 1个字节；

( ??? Y，U和V分量分别对应于分量1，2，3 ??? 只要根据 Y 分量的 横向和竖向的 降采样倍数 就可以判断YUV422 ~~YUV444~~ YUV420 ，JPEG/RTP类型0和1只支持 YUV422和YUV420 根据Y分量 如果是22 就是 YUV420 如果是 11 就是YUV444 如果是 21 就是YUV422)

（b）水平/垂直采样因子：1个字节，高4位代表水平采样因子，低4位代表垂直采样因子； **(?? 代表这个 frame 是 YUV 422 还是 YUV 420 ???)**

（c）量化表ID：1个字节，当前分量使用的量化表ID；（前面定义的 两个量化表之一 ）(**??? 代表这个frame的图像 哪个颜色风量 使用哪个量化表 ？？？** )

有些JPEG文件 **帧开始是 FFC2 (FFC0 FFC1 FFC2) 代表的是 ProgressiveDCT， JPEG/RTP只支持BaselineDCT(顺序DCT)**



10.

一般一个 JPG 文件里会有 2 类 Haffman 表：一个用于 DC 一个用于 AC ，也即实际有 4个表，亮度的 DC，AC 两个，色度的 DC，AC 两个。

DHT，Define Huffman Table定义Huffman表，标记码为0XFFC4；包含2个字段：

（1）数据长度，2个字节，表示（1）--（2）的总长度，即，不包含标记代码，但包含本字段；

（2）Huffman表，数据长度-2个字节，包含以下字段：

（a）表ID和表类型，1个字节，高4位表示表的类型，取值只有两个；0：DC直流；1：AC交流；低4位，Huffman表ID；需要提醒的是，DC表和AC表分开进行编码；

（b）不同位数的码字数量，16个字节；

（c）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，一般需要重复4次。

11 .

DRI，Define Restart Interval，定义 差分 编码累计复位 的间隔，标记码 tagID 0XFFDD；(??一般都没有的??)

包含2个具体字段：

（1）数据长度：2个字节，取值为固定值0X0004，表示（1）（2）两个字段的总长度；即，不包含标记代码，但包含本字段；

（2）MCU块的单元中重新开始间隔：2个字节，如果取值为n，就代表每n个MCU块就有一个RSTn标记；第一个标记是RST0，第二个是RST1,RST7之后再从RST0开始重复；如果没有本标记段(0xFFDD)，或者间隔值为0，就表示不存在重开始间隔和标记RST；（MCU(MinimunCodedUnix) 最小编码单元 每个Scan划分N个MCU 编码单元 一次扫描中包含了一系列最小编码单元（MCU），每个MCU定义了输出图像的一个小矩形块的数据，复位标记 嵌入在熵编码码流（一个扫描中）里的标记，但必须在MCU的边界处出现）

在JPEG中0XFF具有标记的意思，在压缩数据流（???真正的图像信息,或者后文说的熵编码，扫描Scan???）中，如果出现了0XFF，就需要做特别处理了。方法是，如果在图像数据流中遇到0XFF，应该检测其紧接着的字符

（1）0X00，表示0XFF是图像流的组成部分；需要进行译码；

（2）0XD9，表示与0XFF组成标记EOI，即，代表图像流的结束，同时，图像文件结束；

（3）0XD0--0XD7，组成RSTn标记，需要忽视整个RSTn标记，即不对当前0XFF和紧接着的0XDn两个字节进行译码，并按RST标记的规则调整译码变量；(复位标记 出现在扫描SCAN中 最小编码单元MCU的边界，由 DRI标记段来定义，Define Restart Interval)

（4）0XFF，忽略当前0XFF，对后一个0XFF进行判断；

（5）其它数值，忽然当前0XFF，并保留紧接着此数值用于译码；

11 . jpeg帧中的 一个scan开始  tag\_id FFDA tag\_name SOS/Start of Scan

SOS，Start Of Scan，扫描开始；标记码为0XFFDA，包含2个具体字段：

（1）数据长度：2个字节，表示（1）--（4）字段的总长度；

（2）颜色分量数目：1个字节，只有3个可选值，1：灰度图；3：YCrCb或YIQ；4：CMYK；

（3）颜色分量信息：包括以下字段，

（a）颜色分量ID：1个字节；

（b）直流/交流系数表ID，1个字节，高4位表示直流分量的Huffman表的ID；低4位表示交流分量的Huffman表的ID；

（4）压缩图像数据

（a）谱选择开始：1个字节，固定值0X00；

（b）谱选择结束：1个字节，固定值0X3F；

（c）谱选择：1个字节，固定值0X00；

本标记段中，（3）应该重复出现，有多少个颜色分量，就重复出现几次；(?? 代表**后面图像信息 包含颜色分量有多少个，然后每个颜色分量使用哪个哈夫曼表** ？？)

本段结束之后，就是真正的图像信息了；图像信息直到遇到EOI标记就结束了；

12.

EOI，End Of Image，图像结束；标记代码为0XFFD9；

**流传输m-jpeg**

**M-JPEG over HTTP** <http://www.cnblogs.com/leo_wl/p/3272058.html>

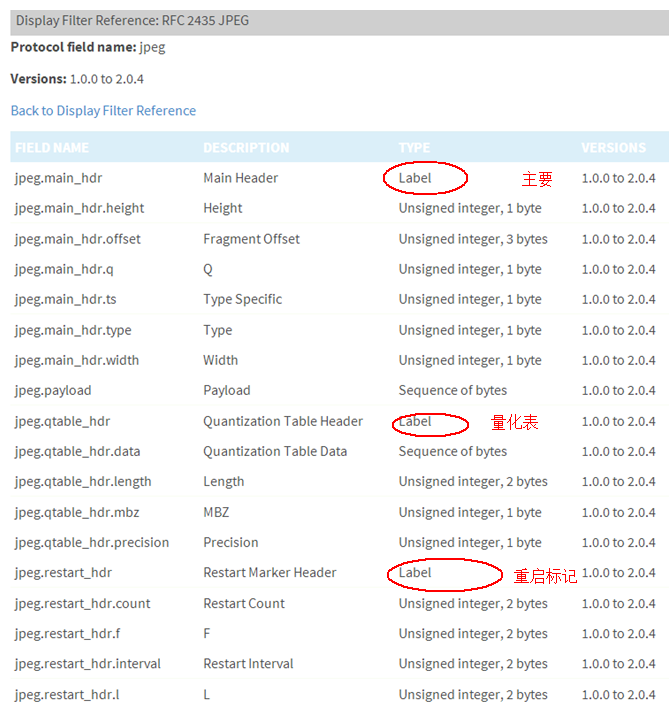
**M-JPEG over RTP**

JPEG压缩视频的RTP荷载格式 RFC2435——RTP Payload Format for JPEG-compressed Video

中文：[RTP打包传输MJPEG码流](http://blog.csdn.net/liuchen1206/article/details/47005657) <http://blog.csdn.net/liuchen1206/article/details/47005657>

英文：

wireshark中 对 mjpeg over rtp 的 RTP/Jpeg格式头 定义 <https://www.wireshark.org/docs/dfref/j/jpeg.html>



RFC2435——RTP Payload Format for JPEG-compressed Video 中文版

为了最大化硬件编解码器的互操作性，我们（rfc2435协议）假定使用顺序DCT模式[1,附录F]，并且限制预定义的RTP/JPEG类型码为单一扫描的隔行图像。这甚至比基本JPEG更为严格，很多硬件实现都不能正确解码基本JPEG(例如，很多硬件不能解码逐行扫描)。 实际上，在一个视频码流中，大部分表格描述的数据在一个视频码流中很少发生变化，这样在省略掉所有可以省掉的表格之后，RTP/JPEG数据就可以用紧缩格式来表示了（紧缩模式中省略了一些码表定义，假定他们在外部定义或在前面的图像中定义）。

每一帧一开始是一个熵编码的扫描。

RTP/JPEG头 (负载类型为MJPEG的码流)

1. 位于RTP头和JPEG荷载之间。

2. 表示原帧头和扫描头中的信息中可以变化的参数，例如量化表和图像大小（为了实现自适应码率传输，允许用户手工调节量化等级或分辨率）。因此RTP/JPEG头中分配了专门的数据域来表示这些信息。

a. 量化表中只有一个小子集是经常使用的(？？就是默认有几个标准的，服务端和客户端都定义好的几个量化表？？)，我们用一个短整数来表示整个量化表集。 一些特定范围的值表示使用自定义的量化表，这种情况下量化表位于JPEG荷载之前

b. 图像的宽和高是显式编码的

还有一些帧头中不变的参数， 类似于哈夫曼码表(对应jpeg文件中 ，每一帧jpeg frame(TagName: SOF Start of Frame 都会定义4个曼哈弗表 )后面 )和色彩空间(tagName:APP0 JIFF )这样的参数，在整个视频流的生命期中都保持不变

1. RTP头部

RTP的时间戳是以90000Hz采样的，同一帧的每一个包都必须有同样的时间戳。

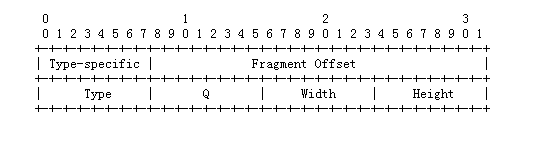
一帧的最后一个包的RTP标志位必须为1。

1. JPEG头 也就是RTP/JPEG头

“主JPEG头” 前8个字节

同一个JPEG帧的的各个包的所有数据域，除了“分段偏移”之外，都必须保持一致。

这个头之后可能会跟着一个复位头和/或量化表头，这取决于“类型”域和“量化”域的值。



类型特定：8比特 这个数据域的含义取决于“类型”域的值。如果没有指定，这个域必须为0并且被接收端忽略。

比如 ‘类型’域的值在0/1 64/65的情况下，可选值如下：(区分扫描方式：渐进扫描还是隔行扫描)

0：图象是渐进扫描(progressively scanned.)的。在计算机显示器上，它可以按照指定的宽高来显示。  
1：当前图像是隔行扫描视频信号的奇数场。主JPEG头中给出的高度是整个图象高度的一半。当前场应当与后面紧跟的偶数场一起重新恢复出整帧图象。偶数场的行恰好处  
于奇数场对应行的上方。  
2：当前图象是隔行扫描视频信号的偶数场。  
3：当前图象是隔行扫描视频信号的一场，但它将按整帧图象的大小来单独显示。在计算机显示器上，每一行都显示两遍，图象高度加倍。

(??? 在live555中JPEGVideoSource类并没有定义跟‘类型特定’相关的虚函数，所以应该不支持扫描模式选择，只实现了0 逐进扫描progressively scanned. 见 JPEGVideoRTPSInk.cpp doSpecialFrameHandling???)

分段偏移: 24比特 分段偏移是当前包在整个JPEG帧中的偏移位置，以字节为单位，以网络字节次序编码（最重要位在前）。

分段偏移加上当前包中的荷载数据长度不能超出2^24字节。(????也就是一个帧的荷载数据不能超过2^24 ~16M ???)

类型: 8比特 “类型”域 (RTP/JPEG类型) 给出了可能出现在JPEG紧缩格式表格描述 和 JPEG未定义的JFIF风格参数的信息。

类型0－63含义是固定的， 由本版本RFC2435以及未来的RFC2435来定义 (??0~63 决定量化表的数目，0代表只有两个量化表，然后根据Q域来决定量化表是计算出来 还是 包含在 帧的第一包中??)

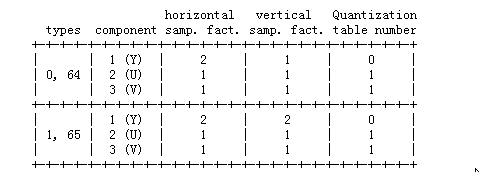
类型64－127与类型0－63相同，但是，添加 在主JPEG头后紧跟一个复位标记头，并且在JPEG数据(JPEG荷载)中存在复位标记。

类型128－255可以通过一个会话建立协议来动态定义（这不在本文档的讨论范围之内）。

(??? 目前live555支持0~63和64~127范围 ,而MJPEG/RTP协议也只是定义了类型0/1 64/65?? 对应 JPEGVideoSource:: type() ???)

对于第一类取值范围，类型0和类型1目前已经定义了，对应第二类范围中(有复位标记的情况)的类型64和 类型65。

类型0，1指的是基本DCT顺序模式("Baseline DCT sequential" mode JPEG定义了四种操作模式：顺序DCT模式，渐进DCT模式，无损模式，以及分级模式)、8比特采样、正方形象素、YUV三种颜色分量以及标准哈夫曼码表(定义在附录 [1, Annex K.3])，一次隔行扫描(???single interleaved scan，后面说可以逐行和隔行???)并带一个扫描分量选择子，来指示是分量1，2还是3。Y，U和V分量分别对应于分量1，2，3。分量1使用0号标准哈夫曼码表和0号量化表，分量2和3使用1号标准哈夫曼码表和1号量化表。(可以看JPEG的定义 SCAN的头信息字段，定义了颜色分量数目数量(1：灰度图；3：YCrCb或YIQ；4：CMYK) 和 颜色分量信息(每个分量使用哪个哈弗曼表)  tag\_id=FFDA tag\_name=SOS/Start of Scan)



采样因子： 类型0的视频的色度分量水平方向上二倍降采样（一般称为4：2：2）

类型1的视频的色度分量在水平和垂直两个方向上都二倍降采样（一般称为4：2：0） YUV420

类型码2－5定为保留，并禁止使用。

类型0和类型1既可以用于传输渐进扫描(progressively scanned)的图象数据，也可以用于传输隔行扫描(interlaced)的图象数据。这两种不同的数据格式在’类型特定’域 加以区分

Q: 8比特 Q域定义了当前帧的量化表。Q值为0－127时量化表可以通过“类型”域 决定的一个参数来计算出来（具体计算方法见后）。

Q值为128－255时会有一个量化表头出现在当前帧第一个包的主JPEG头之后。这个量化表头用来明确定义量化表。

(??? 也就是在Q取128~255，量化表也只会出现在这个帧的第一个包中，这个帧后面的包就没有???

对应 u\_int8\_t JPEGVideoSource:: qFactor() u\_int8\_t const\* JPEGVideoSource::quantizationTables(u\_int8\_t& precision\_精度,u\_int16\_t& length\_长度); ??? )

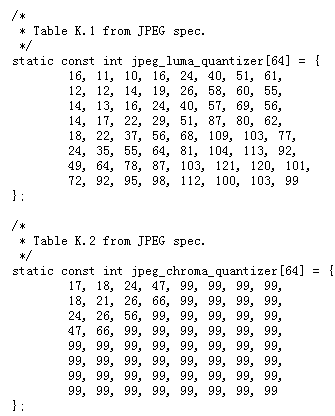
对于JPEG类型0和类型1（以及相应的类型64和65, ‘类型’域的值=0/1 64/65），Q值1－99的定义如下。其它128以下的值保留。

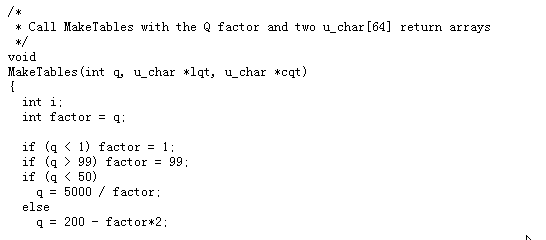
类型0和类型1都需要有两个量化表。这些量化表的计算方法如下：对于1 <= Q <= 99(??相当于Quality??)，用JPEG组织的如下公式来计算一个标量量化因子S：

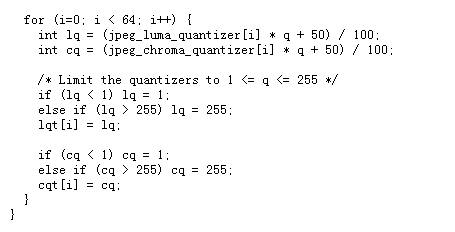


然后把这个S值代入[1]中的表K.1和K.2（每个值都扩展到8比特）,就分别得到了量化表0和量化表1。计算量化表的C源码在附录A中给出。当Q值在128－255之间时，就需要使用动态定义的量化表。这些量化表既可以在带内定义，也可以在带外通过一个会话建立协议来定义。但在每一帧的第一个包中必须有一个量化表头。当量化表在带外定义时，可以通过将包头中的长度域设为0来省略掉量化表。当在带内传输量化表时，并不需要在每一帧都重复传送一遍。(??Q>128情况 每个Q值要一一对应一个量化表,见’量化表头’的解释??)类似于带外的情况，不包含量化表的帧可以在包头中将长度域设置为0。尽管这样做减小了传输量化表带来的OVERHEAD，但是也带来了一些负面效应。一个新的接收者在收到完整量化表之前接收到的所有帧都不能够正确解码。

下面的代码用来通过一个Q因子值生成一个量化表：(live555中实现看 JPEGVIdeoRTPSource.cpp makeDefaultQtables )

(对应色度和亮度量化表)





**(**也就是平时我们对YUV进行JPEG编码(by Bitmap/YUVImage)，使用Quality就是Q因子，然后通过上述计算，等到一个量化表，然后对数据进行编码压缩，所以没有用到特定的量化表。解码时候直接用jpeg图片中的量化表，所以没有Q因子)

宽度: 8比特 宽度域编码图像的宽度，以8象素为单位（例如，宽度为40表示图像宽度为320象素）。最大宽度为2040象素。

高度: 8比特 高度域编码图像的高度，以8象素为单位（例如，高度为30表示图像高度为240象素）。

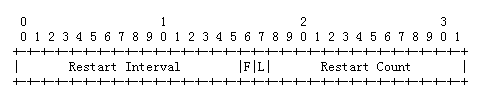
当编码交织视频时，这里表示的是一个视频场的高度，因为每个场是单独编码的。（??奇场和偶场??）

最大高度是2040象素。

对应 JPEGVideoSource:: width() 和 JPEGVideoSource:: height()

**复位标记头**

在“类型”域 为64－127时，复位标记必须紧跟在主JPEG头之后。它提供了正确解码一个包含复位标记的数据流(??也就是一个扫描scan中会包含0xFF开头跟随0XD0--0XD7，组成RSTn标记??)所需要的额外信息。



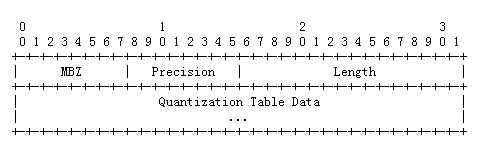
”复位间隔” 域 给出了两个复位标记之间MCU的数目。它和JFIF头中DRI标记段的16比特值是一致的。这个值不能为零。

1. 如果一帧中的复位间隔不能保证在包边界处对齐(??也就是一帧图像分包时候 不能保证每个包最后就是一个复位间隔??)，F比特和L比特必须设为1以及复位计数  
   必须设为0x3fff。这样接收端就必须在解码之前首先重新组装整个帧。
2. 为了支持部分帧解码，必须把一帧分成若干块，每一块包含整数个复位间隔。复位计数域给出第一个复位间隔在当前块中的位置，从而接收端可以知道这些数据对应于当前帧的哪  
   个部分。复位间隔长度的选取应能够使一个块完全放进一个包中（??也就是一个包中有整数个复位间隔??，N个包组成一帧）。在这种情况下，F比特和L比特都必须设为1。然而，如果一个块要放在多个包里，只有第一个包的F比特设为1，也只有最后一个包的L比特设为1。(F first L last )

对应 virtual u\_int16\_t JPEGVideoSource:: restartInterval();

**量化表头**

‘Q’ 域 为128－255时，量化表头必须出现在主JPEG头之后（如果存在复位标记头，则位于复位标记头之后）。对于Q值为128－254的情况，Q值与量化表之间的映射必须是静态的，也就是说，保证接收端只需要读一次与某个Q值对应的量化表，就可以正确解码出所有用该Q值编码的帧(???也就是Q值和量化表是一一对应的，传输过程中可能每帧的Q值都不一样，但是如果有某些帧的Q值一样，那么对应的量化表也应该一样的???)。解码器不能依赖于任何以前的量化表，而需要在每帧都重新载入这些量化表。Q＝255并且长度为0的包是不允许的。

 MBZ 总是0 , RFC2345协议没有定义其值

长度域 给出了后面量化表数据的长度，以字节为单位。长度域为零表示当前帧没有量化表数据。（???同一个帧数据的多个包，只有第一个包有量化表???）

如果长度域的值比剩余的字节数大，整个包必须丢弃。

包含量化表数据时，表的个数取决于’类型’域 的值。例如。’类型’域=0 类型0使用两个表（一个用于亮度分量，另一个用于色差分量）。

每个表是一个64个值的数组，按zig-zag次序，与JFIF的DQT标记段（Defined Quantization Table TagID: FFDB）一致。(?? 8\*8数组 单位可能是8bit或者16bit 取决于’精度’域 ??)

精度域 每一个比特指示了表中系数的大小。

如果这个比特为0，系数为8比特，表长度为64个字节。

如果该比特为1，系数就是16比特的，表长度为128字节。对于16比特的表系数，字节次序是网络次序。

精度域的最右边的比特对应于第一个表，后面的表依次对应于左边的下一个比特。超出表个数的那些比特必须被忽略。(也就是 ‘类型’域=0，有两个量化表，如果Q>128，那么一个帧的第一个包带有两个量化表，’精度’域的只有最右两个比特位是有效的，分别代表 亮度量化表和色度量化表的精度是8bit还是16bit )

**JPEG荷载**

紧跟RTP/JPEG头的数据是包含一次扫描的熵编码的图像数据。这次扫描不包含扫描头，扫描头的信息可以从RTP/JPEG头中推出。(根据’类型’域的值,会对应一些已定义的参数)

扫描的结束可能是隐含的（整幅图象都已经完全解码 ??后面没有数据了??），也可能是显式的，即跟着一个EOI标记。

一次扫描可能会用一些未定义字节填充到任意长度（一些现存的硬件编解码器会在一帧图象的底部生成一些额外的行，解码器需要对它们进行哈夫曼解码来去除这些额外的行。

‘类型’码（64~127）决定着复位标记是否存在。如果支持复位标记，数据包的复位头中必须包含一个非零的复位间隔值，并且复位标记必须是字节对齐的，以一个0xFF起始。另外的0xFF字节可以出现在复位间隔之中。在打包过程中，用这样的方法来进行对齐，例如字对齐，从而实现比较高效的拷贝。除此之外，复位标记不能出现在码流中的任何其它地方。不支持复位标记的类型的码流在任何地方都不能包含复位标记。在数据包中，如果熵编码产生了一个0xFF字节，则必须在它后面填充一个0x00字节(??也就是告诉解码器 这里的FF不是标记段，跟JPEG定义的一样??)。

**分片和组装(Fragmentation and Reassembly)**

由于JPEG的每一帧都相当大，必须经过切分才便于传输。在将一帧切分成若干个包的过程中，应当避免在低层进行分片。如果要求支持部分帧解码，被切分出的每一个包就应当包含整数个复位间隔（如下）。组成同一帧的数据包的时间戳必须保持一致，并且最后一个包的RTP标记位必须为1。每个包的’分段偏移’域的值是这个包中数据在原来整个帧中的偏移位置，以字节为单位。这些包必须按照次序进行传输，并且它们所包含的图象数据不能重叠。整个一帧图象以一个分段偏移为0的包为起始，并以一个RTP标记位为1的包为结束。可以通过RTP的顺序号或者分段偏移结合每个包的长度来检测丢包。数据的重组可以不使用分段偏移的数据（只使用RTP标记位和RTP顺序号），但是在出现包的乱序的情况下，就不可能通过简单高效的拷贝操作来实现图象数据的重组(??也就是不能单纯地链接接收到的包，需要RTP顺序号调整??)。而且，如果前一帧的最后一个包丢失的话，即使当前帧完好无损，接收段也不总能够检测到当前帧(因为最后一个RTP包含有M bit标记一个帧的结束)。

**复位标记**

复位标记插入在JPEG码流中，告诉接收端哈夫曼解码器和直流预测器应当在当前位置复位，并且允许从当前点开始进行部分解码。然而，为了充分实现部分解码，解码器必须知道一个复位间隔中包含的是哪些MCU。为此，原来的JPEG标准中在复位标记中提供了一个短的次序号域(也就是FFD0~FFD7 RSTn标记)。但是对于典型的网络MTU长度来说，这个数域不够长，不能很好的处理丢包问题。因此，在RTP/JPEG的复位头中包含了额外的信息来处理这个问题。

复位间隔的大小应当使得整数个复位间隔能够恰好放在一个数据包里。这样就可以保证这些包可以相互独立地进行解码。如果一个复位间隔的结束处超出了一个包的长度(也就是一个复位间隔已经无能恰好放在一个包中)，可以使用复位标记头中的F比特和L比特来对它进行切分。但是这样生成的包的集合必须全部接收到，接收端才可以正确解码出那个复位间隔中的数据。一旦解码器接收到一个F和L都为1的包，或者是一连串的包，第一个F为1，最后一个L为1，它就可以开始解码了。起始MCU在整幅图象中的位置可以通过将复位计数的值与复位间隔的值相乘来确定。这样的一个包（或一连串包）可以包含任意数量的连续的复位间隔。为了兼容生成码流中就包含复位标记但无法按这些复位标记来分片的编码器(??也就是不能按前面的要求来分RTP包，复位间隔不能恰好一个包另外也不能对每个包标记F和L位??)，将复位计数域设为0x3FFF并且F和L均为1。这样一种模式意味着解码器必须对整幅图象首先进行重组，然后才能解码。

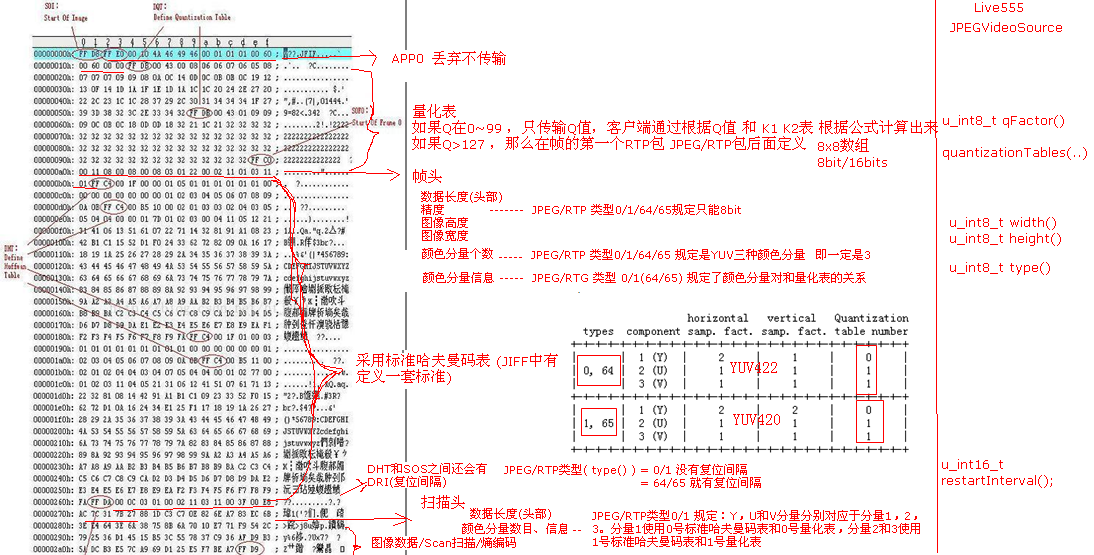
RFC2435协议 附录B中给出了将**RTP/JPEG头中的信息变换到JPEG帧头**(tagID= FFC0 包括图像宽和高，采用的色彩分量个数和每个分量对应的量化表 )和扫描头(tagID= FFDA 包含采用的色彩分量和每个分量对应的曼哈弗表)的C源码(MakeQuantHeader MakeHuffmanHeader MakeDRIHeader 入口函数是MakeHeaders 在live555定义在JPEGVideoRTPSource.cpp)

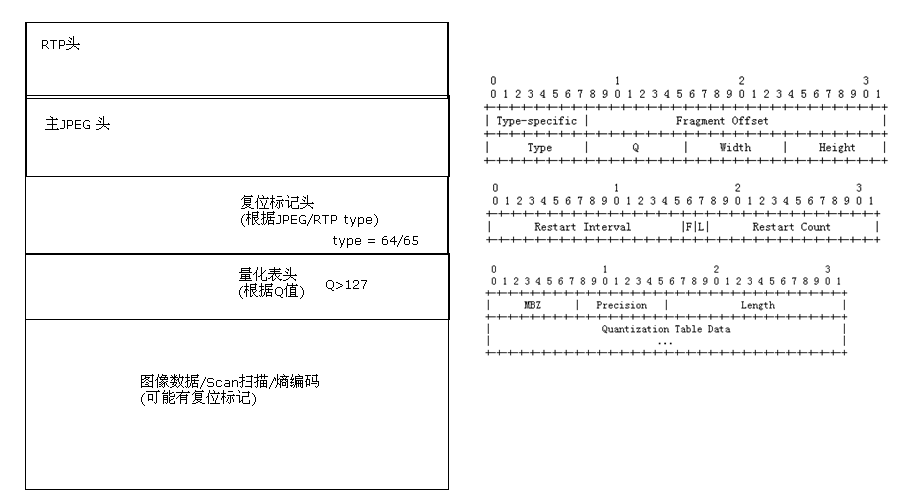
总结：

RFC2435 RTP/JPEG

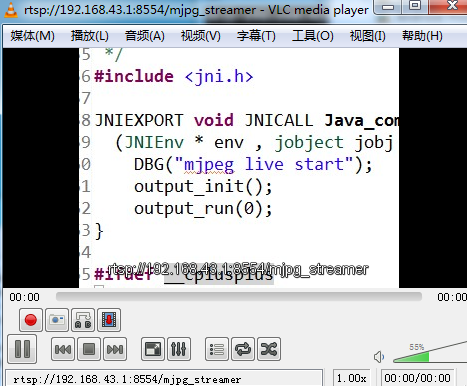
(只)定义了两种 RTP/JPEG类型0/1 和 64/65 这两种类型都 只用标准的哈弗曼表 只能用FRAME0(SOF0 也就是编解码是BaselineDCT)， 0 和 1区别在与YUV422和YUV420 0/1和64/65的区别扫描中是否有复位间隔

定义了量化表的来源 Q在0到99( 99< <127 都强制为99)的时候，使用根据Q值用计算公式和K.1 K.2表得到，不用传输量化表；>127的话，就要传输自定义的量化表

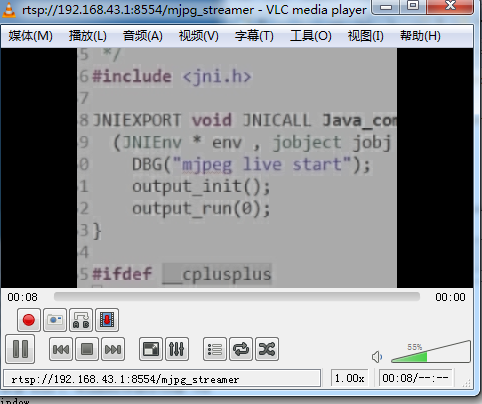
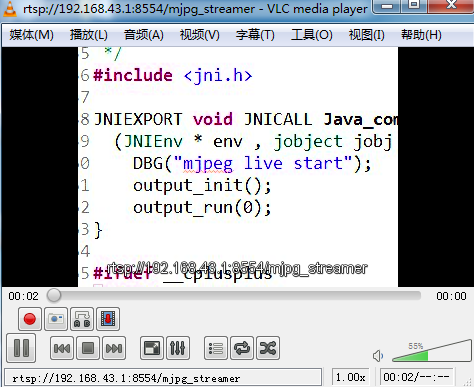
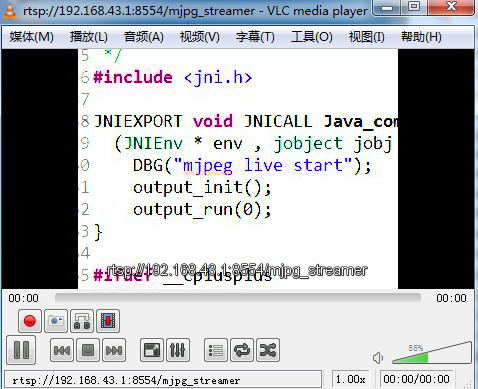




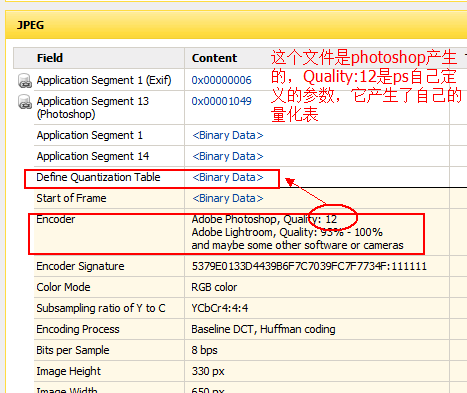
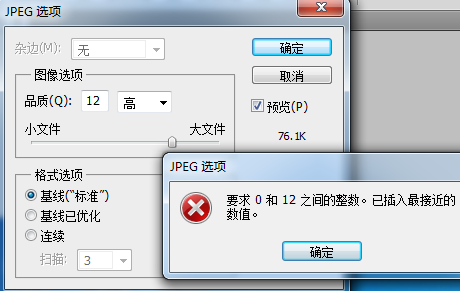
原图 Q=90

量化因子 如果是 90 改成 100 会变成

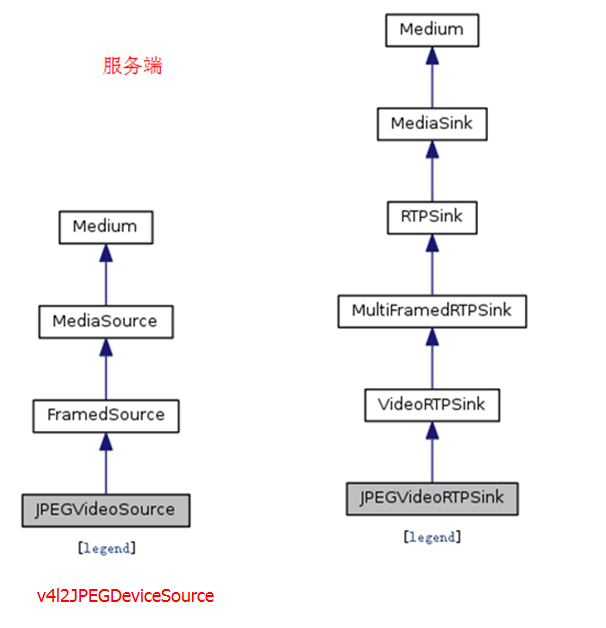
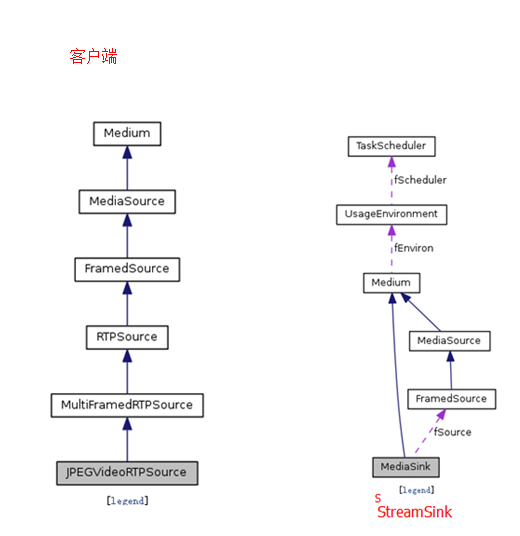
 改成80(会模糊了)  70 

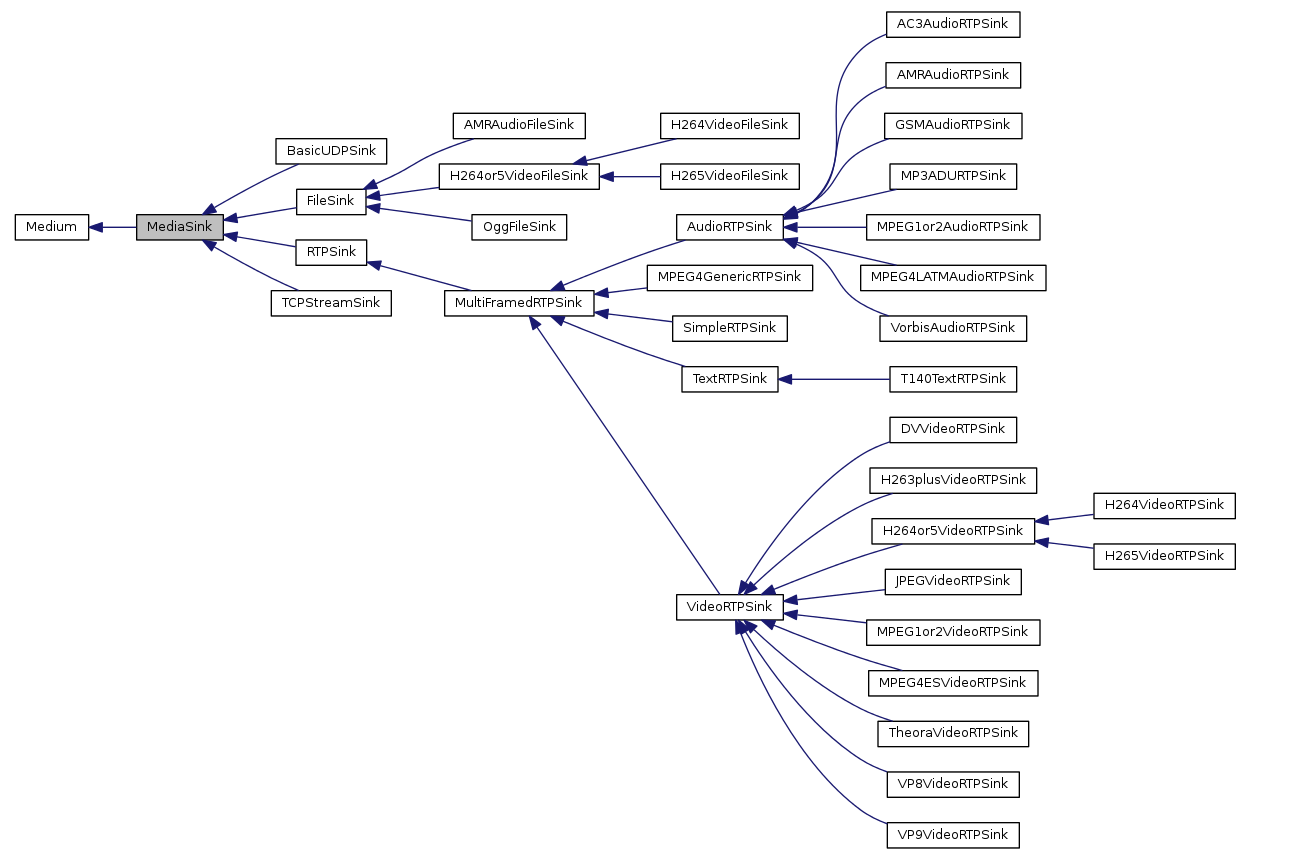
**PhotoShop中 Q因子和量化表**

live: liveMedia/include/JPEGVideoSource.hh

JPEGVideoRTPSource



class v4l2JPEGDeviceSource: public JPEGVideoSource

JPEGVideoRTPSource" encapsulates an incoming stream of motion-JPEG RTP packets. This class is used by clients that receive network packets.

被客户端使用 用于所有 Mjpeg的rtp包

"JPEGVideoSource" encapsulates a stream of motion-JPEG frames that are \*not\* RTP packets. For example, if you have a video camera, or video encoder, that produces a sequence of JPEG frames, then you

would use a subclass of this class to encapsulate it. (Note that"JPEGVideoSource" is an abstract base class that you would need to subclass, and implement a handful of virtual functions, for your own

JPEG source.)

非RTP打包的 mjpeg 视频源 比如来自摄像头或者编码器等 需要继承 ，在服务端实现

Source和Sink：可以把source理解为发送端的流，sink理解为接受端。

MediaSink是各种类型的Sink的基类，MediaSource是各种类型Source的基类

各种类型的流媒体格式和编码的支持即是通过对这两个类的派生实现的。

Source和Sink通过RTP子会话(MediaSubSession)联系在一起

服务端 客户端

v4l2JpegDeviceSource (JPEGVideoSource) JPEGVideoRTPSink JPEGVideoRTPSource

[应用libjpeg提取jpeg质量因子](http://blog.csdn.net/lzhq28/article/details/7775222) <http://blog.csdn.net/lzhq28/article/details/7775222> 原理就是把图片中的量化表 与每个Q因子(0~99)和公式产生的量化表遍历对比

**jpeg的软解码库 和使用 libjpeg**

解压： <http://www.cnblogs.com/hzhida/archive/2012/05/30/2524989.html>

压缩 ： <http://blog.csdn.net/luckywang1103/article/details/17291981>