**图片隐写相关总结**

**一、identify图片元数据提取**

图片元数据，包括格式、宽、高、分辨率、颜色通道、透明通道、质量指数（压缩率）、位深度、像素数等，可以通过工具ImageMagic来读取，ImageMagic是个很强大的图片处理工具，包括创建图片、编辑图片、转换图片、压缩图片等等，而且支持几乎所有的图片格式。此处介绍读取元数据的方法。

下载地址：<https://imagemagick.org/script/download.php>

### **读取元数据方法**

命令：identity -verbose 5.jpg  
在终端输入上述命令，即可查看5.jpg相关的元数据，终端输出如下：

**[root@blog im5]# identify -verbose 5.jpg**

**Image: .\img.png**

**Format: PNG (Portable Network Graphics)**

**Mime type: image/png**

**Class: DirectClass**

**Geometry: 1920x1080+0+0**

**Units: Undefined**

**Type: TrueColorAlpha**

**Endianess: Undefined**

**Colorspace: sRGB**

**Depth: 8-bit**

**Channel depth:**

**Red: 8-bit**

**Green: 8-bit**

**Blue: 8-bit**

**Alpha: 1-bit**

**Channel statistics:**

**Pixels: 2073600**

**Red:**

**min: 0 (0)**

**max: 99 (0.388235)**

**mean: 23.3212 (0.0914559)**

**standard deviation: 10.3488 (0.0405833)**

**kurtosis: 8.774**

**skewness: 2.61234**

**entropy: 0.661315**

**Green:**

**min: 43 (0.168627)**

**max: 124 (0.486275)**

**mean: 73.8182 (0.289483)**

**standard deviation: 14.5364 (0.0570055)**

**kurtosis: -1.21146**

**skewness: -0.00118234**

**entropy: 0.872026**

**Blue:**

**min: 64 (0.25098)**

**max: 154 (0.603922)**

**mean: 110.348 (0.432739)**

**standard deviation: 15.3797 (0.0603125)**

**kurtosis: -0.935226**

**skewness: -0.196375**

**entropy: 0.871399**

**Alpha:**

**min: 255 (1)**

**max: 255 (1)**

**mean: 255 (1)**

**standard deviation: 0 (0)**

**kurtosis: 0**

**skewness: 0**

**entropy: 0.660578**

**Image statistics:**

**Overall:**

**min: 0 (0)**

**max: 255 (1)**

**mean: 115.622 (0.45342)**

**standard deviation: 11.7786 (0.0461904)**

**kurtosis: 5972.14**

**skewness: 273.901**

**entropy: 0.76633**

**Rendering intent: Perceptual**

**Gamma: 0.45455**

**Chromaticity:**

**red primary: (0.64,0.33)**

**green primary: (0.3,0.6)**

**blue primary: (0.15,0.06)**

**white point: (0.3127,0.329)**

**Alpha color: grey74**

**Background color: white**

**Border color: srgb(223,223,223)**

**Transparent color: none**

**Interlace: PNG**

**Intensity: Undefined**

**Compose: Over**

**Page geometry: 1920x1080+0+0**

**Dispose: Undefined**

**Iterations: 0**

**Compression: Zip**

**Orientation: Undefined**

**Properties:**

**date:create: 2019-08-27T18:44:05+08:00**

**date:modify: 2019-08-27T14:31:12+08:00**

**png:cHRM: chunk was found (see Chromaticity, above)**

**png:gAMA: gamma=0.45455 (See Gamma, above)**

**png:IHDR.bit-depth-orig: 8**

**png:IHDR.bit\_depth: 8**

**png:IHDR.color-type-orig: 6**

**png:IHDR.color\_type: 6 (RGBA)**

**png:IHDR.interlace\_method: 1 (Adam7 method)**

**png:IHDR.width,height: 1920, 1080**

**png:sRGB: intent=0 (Perceptual Intent)**

**signature: cf4198447e67df2147c2db732d9c998423b0b5c586f1b2fc382ce01f560de357**

**Artifacts:**

**verbose: true**

**Tainted: False**

**Filesize: 433KB**

**Number pixels: 2.074M**

**Pixels per second: 15.59MB**

**User time: 0.094u**

**Elapsed time: 0:01.133**

**Version: ImageMagick 7.0.1-6 Q16 x64 2016-05-21 http://www.imagemagick.org**

从输出可以看到，读取出了很全面的信息，包括图片的格式、mime类型、颜色空间、位深度、颜色通道信息、透明通道信息、压缩方式、质量指数（压缩率）、像素数等等

使用identify得到图片的平均颜色值

Channel statistics: 下面的；Red:的mean值；Green:的mean值；Blue:的mean值，就是我们需要的三个值

**[root@blog im5]$ identify -verbose 5.jpg | grep mean | head -3 | awk '{match($0," ");print $2}' | awk '{printf("%.0f\n",$1)}'17398124**

三个值分别是r/g/b 三种颜色

（也可以横向打印）

**[root@blog im5]$ identify -verbose 5.jpg | grep mean | head -3 | awk '{match($0," ");print $2}' | awk '{printf("%.0f,",$1)}' | head -c-1**

**173,98,124**

效果展示：

原图



得到平均颜色值:

**[root@blog im4]$ identify -verbose blue.jpg | grep mean | head -3 | awk '{match($0," ");print $2}' | awk '{printf("%.0f,",$1)}' | head -c-155,128,225**

效果：



其他一些命令：

得到图片的宽高

**[root@blog im5]$ identify -format %wx%h 5.jpg**

**799x582**

#-resize: 调整文件大小,这里是把文件转为1个像素大小

#!：表示不管原图片比例，强制缩放后的图片大小是1×1

#format:指定输出的信息,这里的r/g/b是三原色的值

#info -： Specify 'file' as '-' for standard input or output.

#不指定文件而是使用-,用来指定标准输入或标准输出

查看帮助

**[root@blog im5]# identify -help**

查看手册

**[root@blog im5]# man identify**

1. **RGB转换**

在图像处理中，我们会遇到各式各样的颜色空间，比如RGB、HLS、HSV、HSB、YCrCb、CIE XYZ、CIE Lab ，那么它们的区别和应用场所又在哪里呢？

1）RGB是生活中最常见的颜色空间，其中，R代表红色通道，G代表绿色通道，B代表蓝色通道，它们之间的相互搭配组合256\*256\*256，几乎可以包括人类视力所能感知的所有颜色。应用场所：一般的彩色图片都是用RGB三通道来表示，另外，在深度学习领域，很多图片的输入也是RGB三通道的，比如目标跟踪、图像超分等。

2）在HLS颜色空间里，其中，H代表色相通道，L代表饱和度通道，S代表亮度通道

3）HSV、HSB是同样的一个颜色空间，其中，H代表色相，S代表饱和度，V代表明度。如果我们不仔细辨别HSV和HSB，如果它们就是一个东西，在原理和表现上，HSL 和 HSB 中的 H（色相） 完全一致，但二者的 S（饱和度）不一样， L 和 B （明度 ）也不一样：

HSB 中的 S 控制纯色中混入白色的量，值越大，白色越少，颜色越纯；

HSB 中的 B 控制纯色中混入黑色的量，值越大，黑色越少，明度越高

HSL 中的 S 和黑白没有关系，饱和度不控制颜色中混入黑白的多寡；

HSL 中的 L 控制纯色中的混入的黑白两种颜色。

另外，有了RGB通道，又为什么要引入HSL和HSB通道呢？因为，人去描述一个颜色的时候，不仅仅会说是什么颜色，也会说其明暗，因此他们的应用场景常见如艺术家。

4）YCrCb，它的另外一个大名是YUV，其中，其中“Y”表示明亮度，也就是灰阶值；而“U”和“V” 表示的则是色度，作用是描述影像色彩及饱和度，用于指定像素的颜色。YUV常用于视频传输中，而且有一个很好的优点是可以兼容黑白电视机（只传Y通道即可）

5）在CIE XYZ颜色空间里面，其中X、Y、Z是R、G、B的线性变换的结果，因为Y不仅仅包含绿色，并且由于人眼感受绿色，要比红色和蓝色要亮，因此Y也被定义为亮度。其作用就是在RGB系统的基础上，用数学方法，选用三个理想的原色来代替实际的三原色，从而将CIE-RGB系统中的光谱三刺激值和色度坐标r、g、b均变为正值

6）在CIE Lab颜色通道里面，它由三个要素组成，一个要素是亮度（L），a 和b是两个颜色通道。a包括的颜色是从深绿色（低亮度值）到灰色（中亮度值）再到亮粉红色（高亮度值）；b是从亮蓝色（低亮度值）到灰色（中亮度值）再到黄色（高亮度值）。因此，这种颜色混合后将产生具有明亮效果的色彩。这个颜色通道是不依赖光线，也不依赖于颜料

RGB颜色  
众所周知，红、绿、蓝是三原色，分别对应red，green，blue，也就是我们常说的RGB颜色。每种颜色都是由三原色由一定的比例调和而成的，三原色的比例不同，配出的颜色当然也不同。

RGB颜色用三个数字分别代表三种颜色所占的比例，也叫作三个通道，取值是0到255之间的整数。  
比如某个颜色用RGB颜色表示是（255,250,250），如果要转换成HTML颜色，三个数字每一个转换成对应的16进制数，是一个两位数，最终会变成6位数。255——FF，250——FA，250——FA，因此最终的结果是#FFFAFA，转换成功！

那么HTML颜色转换成RGB颜色的过程正好相反，就是把三个16进制数转换成十进制的过程

**RGB color space**

屏幕上显示的图片就是RGB的色彩空间

来源与使用阴极射线管(CRT)的彩色电视

R - red

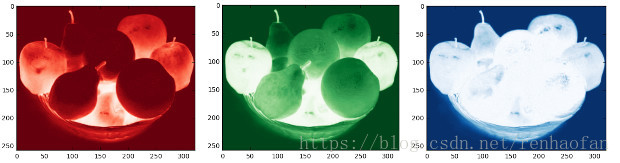
G - green

B - blue

一个原始的图片如下:



把图像分成RGB三通道，如下：



换成另个方向来讲，就是这三张图片如果叠放在一起，就是最上面的那么图片了．这三张图片就是用三个通道，通道的衡量不一样，效果也不一样．所以这也是为什么会有其他色彩空间的原因

**RGBA**

Ａ通道就是图像的透明度

**sRGB**

.sRGB是RGB的一种特定类型。

**Adobe RGB**

Adobe RGB是包含sRGB与CMYK的色彩空间。

**YIQ color space**

美国和日本彩色电视(NTSC标准)曾经使用的一个色彩空间

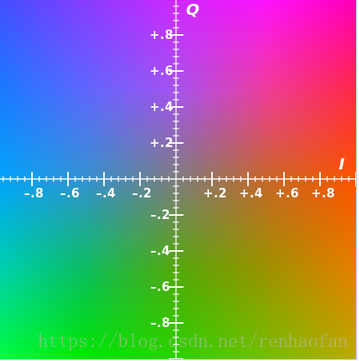
相对与YUV色彩空间旋转了33度

Y – 亮度信号(luminance)，也就是亮度(brightness)

I –chrominance 色度, 色彩从橙色到青色

Ｑ – chrominance 色度, 色彩从紫色到黄绿色

可以看出I,Q是色彩通道，归一化到1.0的visualization



这种表达是有用的，Y通道提供了单色显示所需要的全部的信息,充分利用人眼的特性，因为人眼对于亮度是非常敏感的，亮度代表了觉察到的光的能量。

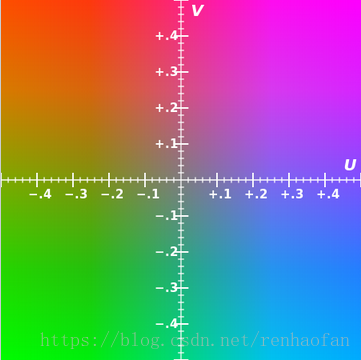
**YUV color space**

澳大利亚，欧洲除法国以外(PAL,法国使用SECAM)的赛色电视使用的色彩空间

Y – luminance 明亮度

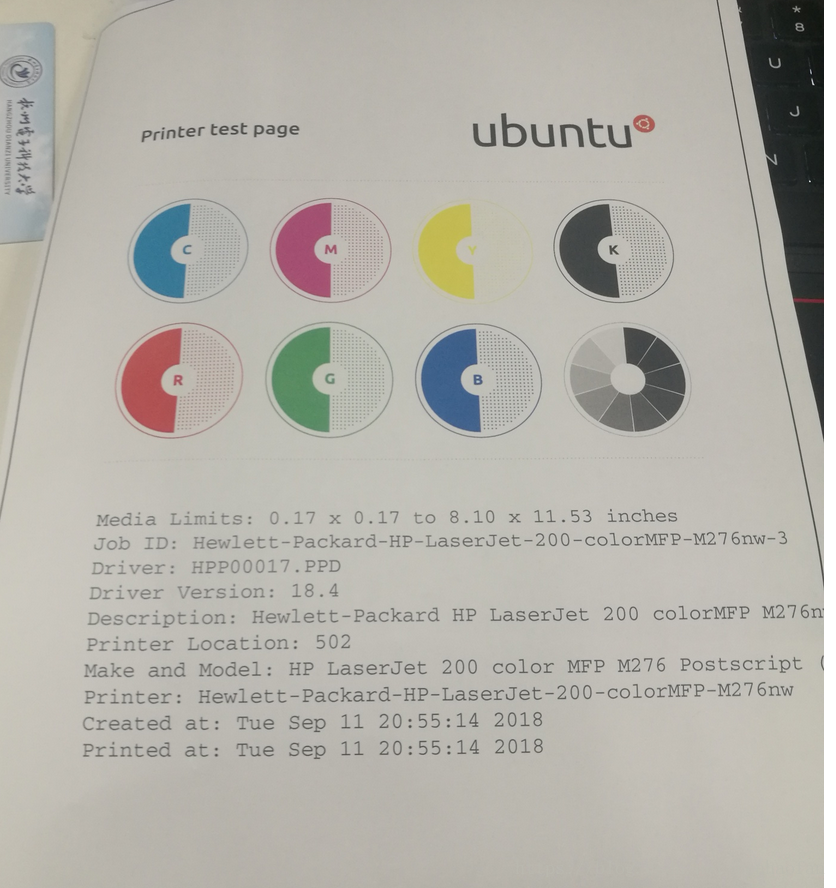
U – chrominance 色度 U = B′ − Y′ (blue − luma)

V – chrominance 色度 V = R′ − Y′ (red − luma)



**CMY color space**

在讲这个之前，我们先看一张在ubuntu下配置打印机成功后，他默认打出来的测试页是什么



从这个图片你也猜到了，CMY肯定是和印刷的有关系，事实也确实是这样的

C – cyan 青

M – mayenta 深红

Y – yellow 黄色

但是由于三种颜色混合在一起产生的黑色不是纯正的

**CMYK**

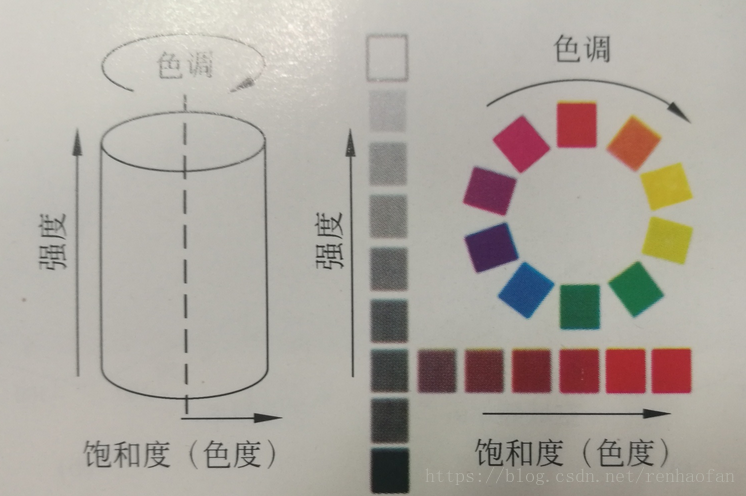
所以就添加了第四个通道的K，来生成黑色

**HSV/HSB color space**

H – hue 色调，是那一种颜色

S – Saturation 饱和度, 一种颜色，可以看成是某种光谱色与白色混合的结果。其中光谱色所占的比例愈大，颜色接近光谱色的程度就愈高，颜色的饱和度也就愈高。饱和度高，颜色则深而艳。光谱色的白光成分为0，饱和度达到最高。

V – value 就是亮度(Brightness)，表示颜色明亮的程度，通俗说就是接收光的多少



面向人眼。

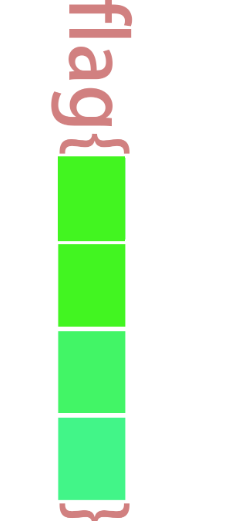
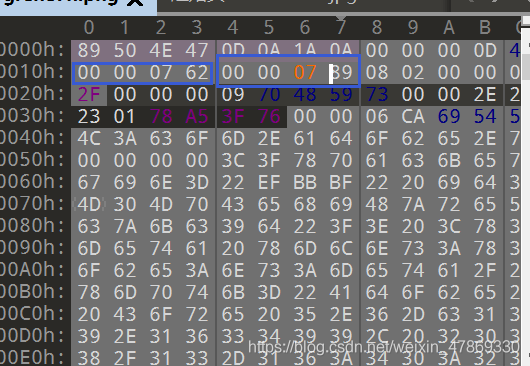
之所以有这种模式，是因为一个画家画画，如果想获得一些另外的颜色，就可以通过颜色的混合。r如果它想把“红旗的红”变成粉红色，就可以把红旗的红色与白色混合，降低饱和度来获得。

图像增加领域

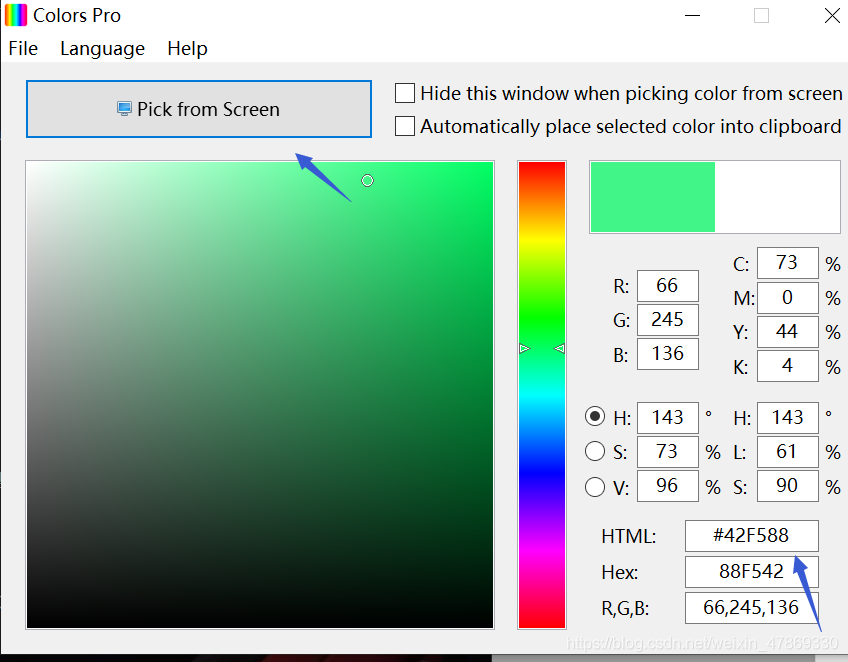
这个色彩空间只对V分量增强，彩色信息就不会受到影响。

**例题：**

爱是一道光  
用010修改宽高  
第二行前面是宽，后面是高



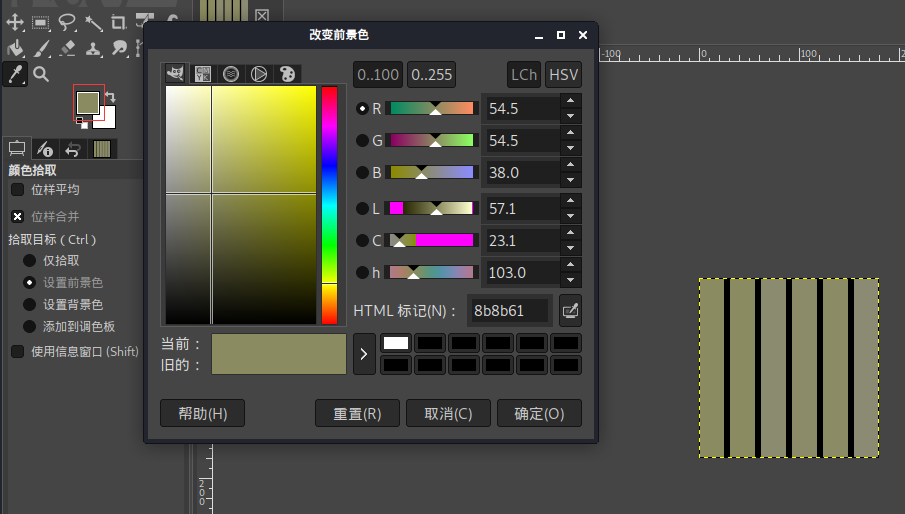
偷懒方法：宽=高，高稍微大点

rgb颜色识别  


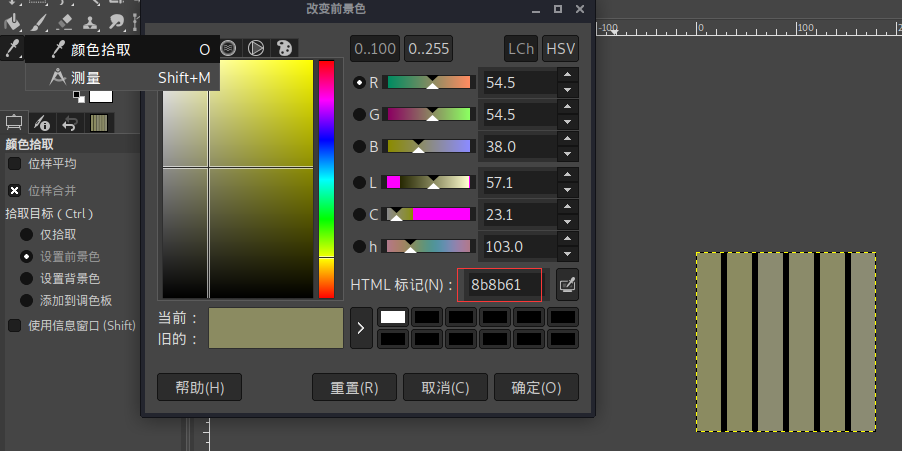
四个都识别，后两位即为flag

**2、**用GIMP查看图片

双击红框查看颜色RGB值

[](https://image-1303962289.cos.ap-beijing.myqcloud.com/image/20201207091010.png)

再选颜色拾取，分别查看栅栏隔开的六个颜色

[](https://image-1303962289.cos.ap-beijing.myqcloud.com/image/20201207091118.png)

8b8b61

8b8b61

8b8b70

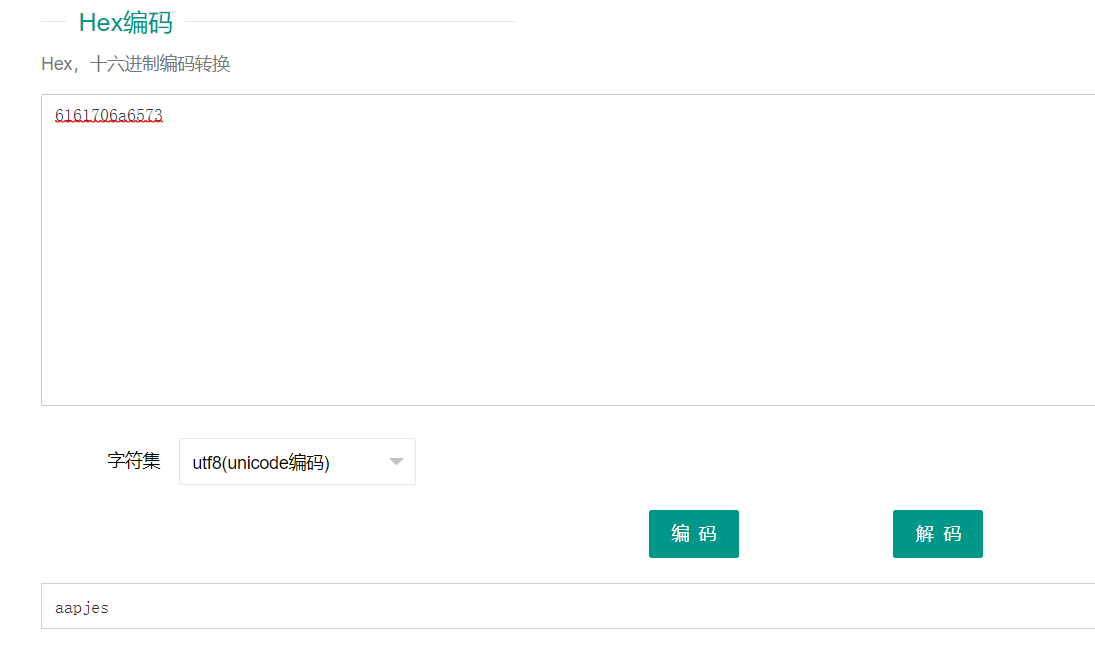
8b8b6a

8b8b65

8b8b73

6161706a6573

hex解码

[](https://image-1303962289.cos.ap-beijing.myqcloud.com/image/20201207091325.png)

1. **PNG、JPG、GIF文件组成及相应考点**

**PNG：**

根据PNG文件的定义来说，其文件头位置总是由位固定的字节来描述的：

|  |  |
| --- | --- |
| 十进制数 | 137 80 78 71 13 10 26 10 |
| 十六进制数 | 89 50 4E 47 0D 0A 1A 0A |

其中第一个字节0x89超出了ASCII字符的范围，这是为了避免某些软件将PNG文件当做文本文件来处理。文件中剩余的部分由3个以上的PNG的数据块（Chunk）按照特定的顺序组成，因此，一个标准的PNG文件结构应该如下：

|  |  |  |  |
| --- | --- | --- | --- |
| PNG文件标志 | PNG数据块 | …… | PNG数据块 |

**PNG数据块（Chunk）**

PNG定义了两种类型的数据块，一种是称为关键数据块(critical chunk)，这是标准的数据块，另一种叫做辅助数据块(ancillary chunks)，这是可选的数据块。关键数据块定义了4个标准数据块，每个PNG文件都必须包含它们，PNG读写软件也都必须要支持这些数据块。虽然PNG文件规范没有要求PNG编译码器对可选数据块进行编码和译码，但规范提倡支持可选数据块。

下表就是PNG中数据块的类别，其中，关键数据块部分我们使用深色背景加以区分。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PNG文件格式中的数据块 | | | | |
| 数据块符号 | 数据块名称 | 多数据块 | 可选否 | 位置限制 |
| IHDR | 文件头数据块 | 否 | 否 | 第一块 |
| cHRM | 基色和白色点数据块 | 否 | 是 | 在PLTE和IDAT之前 |
| gAMA | 图像γ数据块 | 否 | 是 | 在PLTE和IDAT之前 |
| sBIT | 样本有效位数据块 | 否 | 是 | 在PLTE和IDAT之前 |
| PLTE | 调色板数据块 | 否 | 是 | 在IDAT之前 |
| bKGD | 背景颜色数据块 | 否 | 是 | 在PLTE之后IDAT之前 |
| hIST | 图像直方图数据块 | 否 | 是 | 在PLTE之后IDAT之前 |
| tRNS | 图像透明数据块 | 否 | 是 | 在PLTE之后IDAT之前 |
| oFFs | (专用公共数据块) | 否 | 是 | 在IDAT之前 |
| pHYs | 物理像素尺寸数据块 | 否 | 是 | 在IDAT之前 |
| sCAL | (专用公共数据块) | 否 | 是 | 在IDAT之前 |
| IDAT | 图像数据块 | 是 | 否 | 与其他IDAT连续 |
| tIME | 图像最后修改时间数据块 | 否 | 是 | 无限制 |
| tEXt | 文本信息数据块 | 是 | 是 | 无限制 |
| zTXt | 压缩文本数据块 | 是 | 是 | 无限制 |
| fRAc | (专用公共数据块) | 是 | 是 | 无限制 |
| gIFg | (专用公共数据块) | 是 | 是 | 无限制 |
| gIFt | (专用公共数据块) | 是 | 是 | 无限制 |
| gIFx | (专用公共数据块) | 是 | 是 | 无限制 |
| IEND | 图像结束数据 | 否 | 否 | 最后一个数据块 |

为了简单起见，我们假设在我们使用的PNG文件中，这4个数据块按以上先后顺序进行存储，并且都只出现一次。

**数据块结构**

PNG文件中，每个数据块由4个部分组成，如下：

|  |  |  |
| --- | --- | --- |
| 名称 | 字节数 | 说明 |
| Length (长度) | 4字节 | 指定数据块中数据域的长度，其长度不超过(231－1)字节 |
| Chunk Type Code (数据块类型码) | 4字节 | 数据块类型码由ASCII字母(A-Z和a-z)组成 |
| Chunk Data (数据块数据) | 可变长度 | 存储按照Chunk Type Code指定的数据 |
| CRC (循环冗余检测) | 4字节 | 存储用来检测是否有错误的循环冗余码 |

CRC(cyclic redundancy check)域中的值是对Chunk Type Code域和Chunk Data域中的数据进行计算得到的。CRC具体算法定义在ISO 3309和ITU-T V.42中，其值按下面的CRC码生成多项式进行计算：

x32+x26+x23+x22+x16+x12+x11+x10+x8+x7+x5+x4+x2+x+1

PS:(CRC: 一种校验算法。仅仅用来校验数据的正确性的)

**IHDR**

文件头数据块IHDR(header chunk)：它包含有PNG文件中存储的图像数据的基本信息，并要作为第一个数据块出现在PNG数据流中，而且一个PNG数据流中只能有一个文件头数据块。

文件头数据块由13字节组成，它的格式如下表所示。

|  |  |  |
| --- | --- | --- |
| 域的名称 | 字节数 | 说明 |
| Width | 4 bytes | 图像宽度，以像素为单位 |
| Height | 4 bytes | 图像高度，以像素为单位 |
| Bit depth | 1 byte | 图像深度：  索引彩色图像：1，2，4或8  灰度图像：1，2，4，8或16  真彩色图像：8或16 |
| ColorType | 1 byte | 颜色类型： 0：灰度图像, 1，2，4，8或16  2：真彩色图像，8或16  3：索引彩色图像，1，2，4或8  4：带α通道数据的灰度图像，8或16  6：带α通道数据的真彩色图像，8或16 |
| Compression method | 1 byte | 压缩方法(LZ77派生算法) |
| Filter method | 1 byte | 滤波器方法 |
| Interlace method | 1 byte | 隔行扫描方法： 0：非隔行扫描  1： Adam7(由Adam M. Costello开发的7遍隔行扫描方法) |

**PLTE**

调色板数据块PLTE(palette chunk)包含有与索引彩色图像(indexed-color image)相关的彩色变换数据，它仅与索引彩色图像有关，而且要放在图像数据块(image data chunk)之前。

PLTE数据块是定义图像的调色板信息，PLTE可以包含1~256个调色板信息，每一个调色板信息由3个字节组成：

|  |  |  |
| --- | --- | --- |
| 颜色 | 字节 | 意义 |
| Red | 1 byte | 0 = 黑色, 255 = 红 |
| Green | 1 byte | 0 = 黑色, 255 = 绿色 |
| Blue | 1 byte | 0 = 黑色, 255 = 蓝色 |

因此，调色板的长度应该是3的倍数，否则，这将是一个非法的调色板。

对于索引图像，调色板信息是必须的，调色板的颜色索引从0开始编号，然后是1、2……，调色板的颜色数不能超过色深中规定的颜色数（如图像色深为4的时候，调色板中的颜色数不可以超过2^4=16），否则，这将导致PNG图像不合法。

真彩色图像和带alpha通道数据的真彩色图像也可以有调色板数据块，目的是便于非真彩色显示程序用它来量化图像数据，从而显示该图像。

**IDAT**

图像数据块IDAT(image data chunk)：它存储实际的数据，在数据流中可包含多个连续顺序的图像数据块。

IDAT存放着图像真正的数据信息，因此，如果能够了解IDAT的结构，我们就可以很方便的生成PNG图像。

**IEND**

图像结束数据IEND(image trailer chunk)：它用来标记PNG文件或者数据流已经结束，并且必须要放在文件的尾部。

如果我们仔细观察PNG文件，我们会发现，文件的结尾12个字符看起来总应该是这样的：

00 00 00 00 49 45 4E 44 AE 42 60 82

不难明白，由于数据块结构的定义，IEND数据块的长度总是0（00 00 00 00，除非人为加入信息），数据标识总是IEND（49 45 4E 44），因此，CRC码也总是AE 42 60 82。

**JPG：**

JPEG是一个压缩标准，又可分为标准 JPEG、渐进式JPEG及JPEG2000三种:  
  
①标准JPEG：以24位颜色存储单个光栅图像，是与平台无关的格式，支持最高级 别的压缩，不过，这种压缩是有损耗的。此类型图片在网页下载时只能由上而下依序显示图片，直到图片资料全部下载完毕，才能看到全貌。  
②渐进式 JPEG：渐进式JPG为标准JPG的改良格式，支持交错，可以在网页下载时，先呈现出图片的粗略外观后，再慢慢地呈现出完整的内容，渐进式JPG的文件 比标准JPG的文件要来得小。  
③JPEG2000：新一代的影像压缩法，压缩品质更好，其压缩率比标准JPEG高约30％左右，同时支持有损 和无损压缩。一个极其重要的特征在于它能实现渐进传输，即先传输图像的轮廓，然后逐步传输数据，让图像由朦胧到清晰显示。  
  
以一幅24 位彩色图像为例，JPEG的压缩分为四个步骤：  
  
①颜色转换：在将彩色图像进行压缩之前，必须先对颜色模式进行数据转换。转换完成之后 还需要进行数据采样。  
②DCT变换：是将图像信号在频率域上进行变换，分离出高频和低频信息的处理过程，然后再对图像的高频部分（即图像细 节）进行压缩。首先以象素为单位将图像划分为多个8×8的矩阵，然后对每一个矩阵作DCT 变换。把8×8的象素矩阵变成8×8的频率系数矩阵（所谓频率 就是颜色改变的速度），频率系数都是浮点数。  
③量化：由于下面第四步编码过程中使用的码本都是整数，因此要对频率系数进行量化，将之转换为整 数。数据量化后，矩阵中的数据都是近似值，和原始图像数据之间有了差异，这一差异是造成图像压缩后失真的主要原因。这一过程中，质量因子的选取至为重要。 值选得大，可以大幅度提高压缩比，但是图像质量就比较差，质量因子越小图像重建质量越好，但是压缩比越低。  
④编码：编码是基于统计特性的方 法。  
  
 四个步骤都完成后的JPEG文件，其基本数据结构为两大类型：“段”和经过压缩编码的图像数据。  
  
二、数据结构  
  
1.段的一般结构如下表所示：  
  
表1：段的一般结构  
-----------------------------------------------------------------  
名 称   字节数  数据   说明  
-----------------------------------------------------------------  
段 标识   1     FF     每个新段的开始标识  
段类型    1           类型编码（称作“标记码”）  
段长 度   2           包括段内容和段长度本身,不包括段标识和段类型  
段内容               ≤65533字节  
-----------------------------------------------------------------  
说明：  
①JPG文件中所有关于宽度高度长度间隔这一类数据，凡是＞１字节的，均采用Motorola格式，即：高位在前，低位在后。  
② 有些段没有长度描述也没有内容，只有段标识和段类型。文件头和文件尾均属于这种段。  
③段与段之间无论有多少FF都是合法的，这些FF称为“填充字 节”，必须被忽略掉。

2.段类型有30种，但只有10种是必须被所有程序识别的，其它的类型都可以忽略。所以下面只列出这10种类型。  
  
表2：段类型  
---------------------------------------  
名称   标记码   说明  
---------------------------------------  
SOI    D8     文件头  
EOI    D9     文件尾  
SOF0    C0     帧开始（标准JPEG）  
SOF1    C1     同上  
DHT    C4     定义Huffman表（霍夫曼表）  
SOS    DA     扫描行开始  
DQT    DB     定义量化表  
DRI    DD     定义重新开始间隔  
APP0    E0     定义交换格式和图像识别信息  
COM    FE     注释  
-----------------------------------------------------------  
说明：有的文章也将DNL段（标记码＝DC，定义扫描行数）列为必须段。  
  
3.以下按一般JPEG文件的段排列顺序详细介绍各种段的结构：  
  
表3：SOI（文件头）  
-------------------------  
名称  字节数   值  
-------------------------  
段 标识   1   FF  
段类型    1   D8   
------------------------  
说明：这两个字节构成了 JPEG文件头。  
  
表4：APP0（图像识别信息）  
--------------------------------------------------------------------------  
名 称       字节数  值          说明  
--------------------------------------------------------------------------  
段 标识       1    FF  
段类型        1     E0  
段长度        2    0010        如果有 RGB缩略图就＝16＋3n  
（以下为段内容）  
交换格式       5   4A46494600   “JFIF”的ASCII码  
主 版本号      1  
次版本号       1    
密度单位       1               0＝无单位；1＝点数/英 寸；2＝点数/厘米  
X像素密度      2               水平方向的密度     
Y像素密 度     2               垂直方向的密度  
缩略图X像素    1               缩略图水平像素数目    
缩 略图Y像素    1               缩略图垂直像素数目  
（如果“缩略图X像素”和“缩略图Y像素”的值均＞0，那么才有下面的数据）  
RGB缩略图     3×n           n＝缩略图像素总数＝缩略图X像素×缩略图Y像素  
--------------------------------------------------------------------------

说明：  
①JFIF是JPEG File Interchange Format的缩写，即JPEG文件交换格式，另外还有TIFF等格式，很少用  
②“如果有RGB缩略图就＝16＋3n”是什么意思呢？比如说“缩略图X像素”和“缩略图Y像素”的值均为48，就表示有一个48×48像素的 缩略图（ｎ＝48×48），缩略图是24位真彩位图，用３个字节来表示一个像素，所以共占用3n个字节。但大多数JPG文件都没有这个“鸡肋”缩略图。  
  
表 5：COM（注释）  
--------------------------------------------------------------------------  
名 称    字节数    值     说明  
--------------------------------------------------------------------------  
段 标识    1     FF  
段类型     1     FE  
段长度     2         其值＝注释字符的字节数＋2  
段内容              注释字符  
--------------------------------------------------------------------------  
说明：有的JPEG文件没有这个段。

**GIF：**

GIF(Graphics Interchange Format)的原义是“图像互换格式”，是CompuServe公司在 1987 年开发的图像文件格式。GIF文件的数据，是一种基于LZW算法的连续色调的无损压缩格式。其压缩率一般在50%左右，它不属于任何应用程序。GIF格式可以存多幅彩色图像，如果把存于一个文件中的多幅图像数据逐幅读出并显示到屏幕上，就可构成一种最简单的动画。

GIF分为静态GIF和动画GIF两种，扩展名为.gif，是一种压缩位图格式，支持透明背景图像，适用于多种操作系统，“体型”很小，网上很多小动画都是GIF格式。其实GIF是将多幅图像保存为一个图像文件，从而形成动画，最常见的就是通过一帧帧的动画串联起来的搞笑gif图，所以归根到底GIF仍然是图片文件格式。

但GIF只能显示256色。和jpg格式一样，这是一种在网络上非常流行的图形文件格式。

GIF主要分为两个版本，即GIF 89a和GIF 87a

GIF 87a：是在1987年制定的版本

GIF 89a：是1989年制定的版本。在这个版本中，为GIF文档扩充了图形控制区块、备注、说明、应用程序编程接口等四个区块，并提供了对透明色和多帧动画的支持

**格式**

1. 文件头是一个带有识别GIF格式数据流的数据块，用以区分早期版本和新版本。



（2）逻辑屏幕描述区定义了与图像数据相关的图像平面尺寸、彩色深度，并指明后面的调色板数据区属于全局调色板还是局部调色板。若使用的是全局调色板，则生成一个24bit的RGB全局调色板，其中一个基色占用一个字节。

（3）调色板数据区。分通用调色板和局部调色板。其中通用调色板适于文件中所有图像，局部调色板只适于某一个图像。

（4）图像数据区的内容有两类，一类是纯粹的图像数据，一类是用于特殊目的的数据块（包含专用应用程序代码和不可打印的注释信息）。在GIF89a格式的图像文件中，如果一个文件中包含多个图像，图像数据区将依次重复数据块序列。

（5）结束标志区的作用主要是标记整个数据流的结束。

**制作**

PC上制作软件主要为Adobe ImageReady 和 fireworks 两个。

WEB上gif在线制作编辑gif5.net，支持图片、视频、FLASH转GIF。

**特点**

GIF格式的图像文件具有如下特点：

（1）GIF格式图像文件的扩展名是“.gif”。

（2）对于灰度图像表现最佳。

（3）具有GIF87a和GIF89a两个版本。

（4）采用改进的LZW压缩算法处理图像数据。

（5）调色板数据有通用调色板和局部调色板之分，有不同的颜色取值。

（6）不支持24bit彩色模式，最多存储256色。

**用途**

①GIF是压缩格式的文件，用于减少文件在网络上传递的时间；

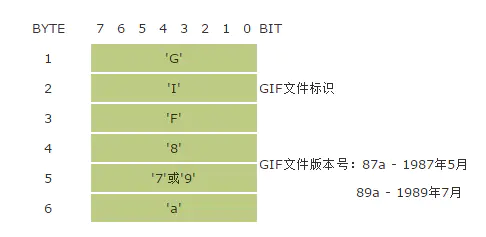
②GIF的位深为1-8bit，单色透明，由一个最多256种颜色的调色板实现，图像大小最多为64K×64K像素。GIF主要是为一个数据流而设计的一种传输格式，而不是作为文件的存当格式，因此它是最复杂的一种图像文件格式；

③支持Bitmap、Grayscale和索引彩色模式。

GIF格式的文件结构整体上分为三部分：文件头、GIF数据流、文件结尾。其中，GIF数据流分为全局配置和图像块。接下来我们将逐一分析GIF格式各部分的作用，

**GIF署名（Signature）和版本号（Version）:**

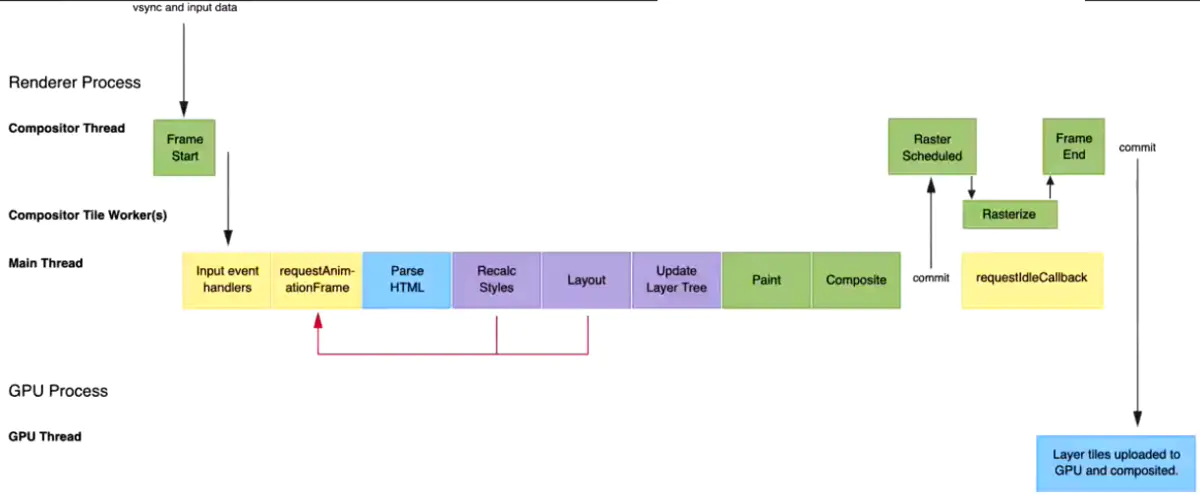
GIF的前6个字节内容是GIF的署名和版本号。我们可以通过前3个字节判断文件是否为GIF格式，后3个字节判断GIF格式的版本。



(帧浏览器)

渲染帧是指浏览器一次完整绘制过程，帧之间的时间间隔是DOM视图更新的 最小间隔 。 由于主流的屏幕刷新率都在60hz，因此渲染一帧的事件就必须控制在16.7ms内才能保证不掉帧。 也就是说每一次渲染都要在 16.7ms 内页面才够流畅不会有卡顿感。 这段时间内浏览器需要完成如下事情： 合成：合成各层渲染结果。 一个帧内要做这么多事情……如果js执行时间过长超过16.7ms，就会block住，那么就会 丢掉一次帧的绘制。 一般 一个帧内的多次DOM改动会被合并渲染。 window.requestAnimationFrame 用于在下一个渲染帧之前执行一个回调函数。 可以用来做 逐帧动画 ，这会使你的动画函数先于浏览器重绘动作。 通常来说，被调用的频率就是60hz。

浏览器的绘制渲染过程可以总结为 一帧数据的渲染  
注意：浏览器并不是因为页面的重绘和回流才会进行数据帧的渲染，一般情况下，浏览器每隔 16ms 刷新一次数据帧。



浏览器一帧的渲染过程

* GPU Process GPU进程，负责绘制（包括重绘和回流）。WebGL是JS面向GPU编程的接口。
  + GPU Thread GPU主线程。
* Renderer Process 渲染进程，又分为三个线程：
  + Compositor Thread 编排/排版线程，负责启动主线程；
  + Compositor Tile Worker(s) 栅格化线程，简单理解为 把CPU数据处理成GUP数据；
  + Main Thread 主线程。

浏览器接收到一帧数据Frame start，由编排线程决定是否启动主线程，由主线程负责执行这一套渲染的逻辑，最终交给 GPU进程 绘制到页面上。

如果一帧数据不需要经过诸如【页面的重绘重排、JS计算、JS事件、CSS动画...】等等的逻辑相关的处理，编排线程是不会启动主线程的，而是直接会绘制到页面上，如<input type="text" />上的光标闪烁，渲染完成之后的页面滚动操作。对于这些不经过渲染主进程的操作，自然也不会产生阻塞。

以<input type="text" />为例，在输入数据时，如果没有注册任何JS事件，那么也就不会涉及逻辑相关的操作，即不会启动主线程。如果注册了事件，原生交互操作会直接提交给GPU进程去绘制，然后会启动渲染主线程。

**启动渲染主线程，依次可分为一下几个阶段：**

* input event handlers 首先处理用户输入/设定的各种操作。
* requestAnimationFrame 它也是一个暴露给开发者的接口，通常用于执行动画，但此刻执行的是上一帧注册的（动画）事件！
* Parse HTML 解析HTML、CSS、JS，最终生成 DOM Tree  
  如果JS中调用了requestAnimationFrame()，此时只会注册该事件，放在下一帧执行。之所以不立即执行，且在页面解析之前再执行上一帧设置的事件，是为了避免（动画）事件的执行导致页面反复的重排重绘。当然，也正是因为每一帧的图像不同，才有了一个动画交互的效果。  
  解析HTML的过程中，遇到JS就调用（交给）V8引擎处理，所以V8引擎和JS的执行是在 **渲染主线程** 中完成的。另外JS可能会操作DOM、修改CSS，从而导致页面重排，所以 JS的执行是同步的，即 会阻塞页面渲染！
* Recalc Styles 重新计算样式，生成 CSS Tree
* Layout 合并 DOM Tree 和 CSS Tree，生成 Layer Tree  
  当然，Layer Tree与 DOM Tree 不一定是一一对应的，比如对某个DOM元素设置了隐藏，使用CSS伪元素。
* Update Layer Tree 更新Layer Tree，生成 [层叠上下文](https://links.jianshu.com/go?to=https://developer.cdn.mozilla.net/zh-CN/docs/Web/Guide/CSS/Understanding_z_index/The_stacking_context" \t "_blank)
* Paint 遍历层级，计算层叠上下文，形成一个层级记录表，记录图层绘制的先后顺序。
* Composite 根据Layer Tree和层级记录表，把页面处理成多个图块，并把数据提交给排版线程。  
  早期浏览器每次只展示视口可见的页面，滚动时又重新渲染可见的部分；后期做了优化，把整个页面拆分成多个图块，视口需要展示哪部分页面，则把涉及的图块拼接到一起（合成）。  
  在之后的渲染中（如下拉滚动条），如果当前视口需要展示的图块都没有涉及逻辑相关的处理，那么就不会再启动渲染主线程，而是由 **排版线程** 合成缓存中的这些图块，并提交到数据栅格化阶段。
* Raster Scheduled、Rasterize 排版线程组织页面图块，栅格化数据，提交给GPU进程去绘制；  
  一个数据帧渲染结束，把栅格化的数据提交给 GPU**进程** 去绘制页面。
* requestIdleCallback 它也是一个暴露给开发者的接口，在一个数据帧渲染完成并提交之后，如果还有剩余时间，才会处理此API设置的事件。

**MDN：** window.requestIdleCallback() 在浏览器的空闲时段内调用的函数排队。这使开发者能够在主事件循环上执行后台和低优先级工作，而不会影响延迟关键事件，如动画和输入响应。函数一般会按先进先调用的顺序执行。然而，如果回调函数指定了执行超时时间，则有可能为了在超时前执行函数而打乱执行顺序。

综上所述，在一帧数据的渲染过程中，JavaScript执行的时机有三处，分别时：requestAnimationFrame、Parse HTML、requestIdleCallback。当然，这只是简单的JS代码，对于异步、postMessage() 等情况，执行方式又是不同的！

#### 卡顿与丢帧

对于一般的电脑，屏幕的图像（页面）刷新频率大概是60Hz，即1s刷新60次（约16ms一次）。实际上，浏览器内核支撑自身体系运行也需要消耗一些时间，所以留给我们的时间差不多只有10ms。如果一段JS代码的运行时间过多，导致一帧数据的解析与渲染超过16ms，就会造成丢帧，现象则是页面滚动卡顿，动画不流畅等等。

1. **工具介绍 stegdetect stegsolve常见用法**
2. **Stegdetect**

下载地址：<https://pan.baidu.com/s/1tQqG2J13cmj7TCIw0Z5CBQ>

提取码：v9n0

Stegdetect程序主要用于分析JPEG文件，可以检测到通过JSteg、JPHide、OutGuess、Invisible Secrets、F5、appendX和Camouflage等这些隐写工具隐藏的信息

Stegdetect通过统计测试来分析图像文件中是否包含隐藏内容。它运行静态测试以判断隐藏的内容是否存在。此外，它还会尝试识别隐藏内容是通过哪个隐写工具嵌入的。

Stegdetect的主要选项如下：

q – 仅显示可能包含隐藏内容的图像

n – 启用检查JPEG文件头功能，以降低误报率。如果启用，所有带有批注区域的文件将被视为没有被嵌入信息。

如果JPEG文件的JFIF标识符中的版本号不是1.1，则禁用OutGuess检测。

s – 修改检测算法的敏感度，该值的默认值为1。检测结果的匹配度与检测算法的敏感度成正比，

算法敏感度的值越大，检测出的可疑文件包含敏感信息的可能性越大。

d – 打印带行号的调试信息。

t – 设置要检测哪些隐写工具（默认检测jopi），可设置的选项如下：

j – 检测图像中的信息是否是用jsteg嵌入的。

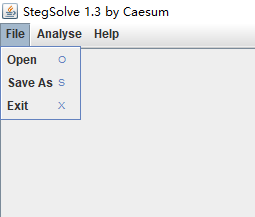
o – 检测图像中的信息是否是用outguess嵌入的。

p – 检测图像中的信息是否是用jphide嵌入的。

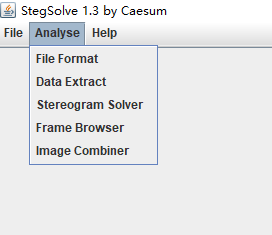
i – 检测图像中的信息是否是用invisible secrets嵌入的。

当然误报率还是有的

1. **Stegsolve**



这个没什么好说的，打开文件 ，保存，退出



 在分析里面从上到下的依次意思是

File Format:文件格式

Data Extract:数据提取

Steregram Solve:立体试图 可以左右控制偏移

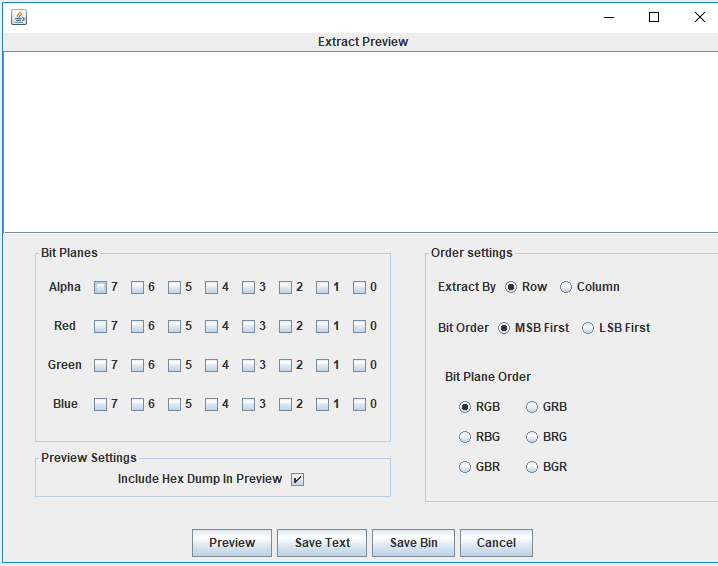
Frame Browser:帧浏览器

Image Combiner:拼图，图片拼接

用法（使用场景）

1.File Format:这里你会看见图片的具体信息有时候有些图片隐写的flag会藏在这里

2.Data Extract:(涉及到数据提取）



 左边一大部分主要是讲了RGBA（Alpha是透明度）的颜色通道

为了方便理解我们分开说

RGB是红绿蓝 但他们的值代表的实际上是亮度

R的数字越大，则代表红色亮度越高；R的数字越小，则代表红色亮度越低。G，B同理

R的亮度各有256个级别，GB同理。即从0到255，合计为256个。从数字0到255的逐渐增高，我们人眼观察到的就是亮度越来越大，红色、绿色或蓝色越来越亮。然而256是2的8次方 所以你会看见上图的7~0  一共8个通道

而Alpha就是透明度 该通道用256级灰度来记录图像中的透明度信息，定义透明、不透明和半透明区域

alpha的值为0就是全透明，alpha 的值为 255 则表示不透明

因此左半部分就理解了

右半部分就是Extra By(额外的)和Bit Order（位顺序）和Bit Plane Order（位平面的顺序）

1）.Extra By(额外的)：分为row（行）和column（纵）

   每个像素用R，G，B三个分量表示，那么一张图片就像一个矩阵，矩阵的每个单位就是（0~255，0~255，0~255）

   也就会有是纵排列和行排列了，一般事先访问行再访问列（如果相反会引起ve使用方法）

￼2）.Bit Order（位顺序）:MSB是一串数据的最高位，LSB是一串数据的最低位。

3）.Bit Plane Order（位平面的顺序)

     整个图像分解为8个位平面，从LSB(最低有效位0)到MSB（最高有效位7）随着从位平面0 到位平面7，位平面图像的特征逐渐变得复杂，细节不断增加。（一般我们的图片如果是RGB那么就是24位 3乘8嘛）

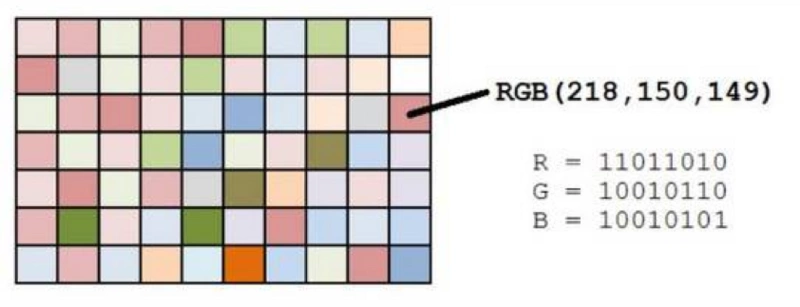
4）Bit Plane Order（位平面的顺序）:一般图片是24位 也就是3个8 大家可以想像成三明治 比如BGR就是B为三明治第一层 G为第二层 R为第三层。

3.Steregram Solve:立体试图 可以左右控制偏移 可以放张图片试一下就知道这个是什么意思了

4.Frame Browser:帧浏览器   主要是对GIF之类的动图进行分解，把动图一帧帧的放，有时候会是二维码

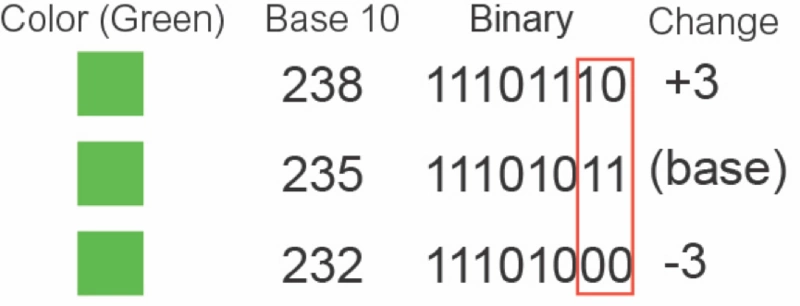
5.Image Combiner:拼图，图片拼接（意思显而易见）

接下来会带大家实战去深入理解一下Data Extract里面ctf经常用到的LSB隐写

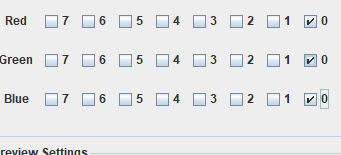


 这个我们之前介绍的很详细

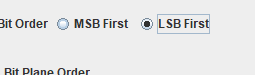
而LSB隐写就是修改RGB颜色分量的最低二进制位也就是最低有效位（LSB），而人类的眼睛不会注意到这前后的变化，（人类的眼睛只能识别一部分颜色的变化）



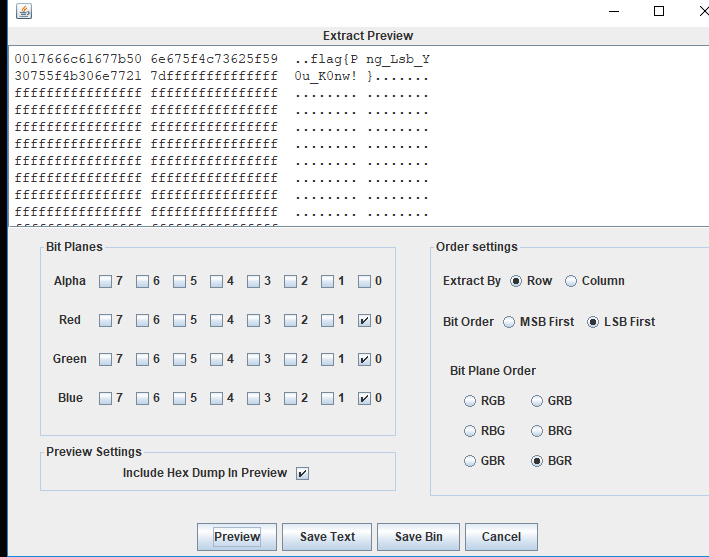
 如果我们修改lsb那么颜色依然和没修改的一样，并且修改的话每个像数可以携带3比特的信息。



 这个作用是在于把最低位的二进制全部提取出来



这个作用在于对提取出来的最低位使用lsb解码算法



解题思路：

## **图像隐写术进行数据隐写分为以下几类：**

1.在图片右击查看属性，在详细信息中隐藏数据

2.将数据类型进行改写（rar或者zip数据改为jpg等格式）

3.根据各种类型图像的固定格式，隐藏数据

在编译器中修改图像开始的标志，改变其原来图像格式

在图像结束标志后加入数据

在图像数据中加入数据，不影响视觉效果情况下修改像素数据，加入信息

4.利用隐写算法将数据隐写到图片中而不影响图像（仅限于jpg图像） 隐写常用的算法有F5，guess jsteg jphide。

## **破解隐写术方法及步骤**

1.查看图像属性详细信息是否有隐藏内容

2.利用winhex或nodepad++打开搜索ctf,CTF，flag,key等关键字是否存在相关信息

3.检查图像的开头标志和结束标志是否正确，若不正确修改图像标志恢复图像，打开查看是否有flag或ctf信息，（往往gif属于动图，需要分帧查看各帧图像组合所得数据 若不是直接的ctf或flag信息 需要考虑将其解码）

jpg图像开始标志：FF D8 结束标志 ：FF D9

gif图像开始标志：47 49 46 38 39 61 (GIF89)结束标志：01 01 00 3B

bmp图片开始标志：42 4D //92 5B 54 00 00 00 00 00 结束标志：00

png图片开始标志：89 50 结束标志：60 82

4.将图片放置在kail系统中，执行binwalk xxx.jpg 查看图片中是否是多个图像组合或者包含其他文件（若存在多幅图像组合，再执行foremost xxx.jpg会自动分离；若检测出其他文件修改其后缀名即可，如zip）

5.使用StegSolve对图像进行分通道扫描，查看是否为LSB隐写

6.在kail下切换到F5-steganography，在java Extract运行

命令：java Extract 123456.jpg图片的绝对地址 -p 123456

判断是否为F5算法隐写

7.在kali系统中使用outguess-master工具（需要安装），检测是否为guess算法隐写

## **算法隐写的具体操作**

1.F5算法隐写

具体操作：在kail下切换到F5-steganography，在java Extract运行

命令：java Extract 123456.jpg图片的绝对地址 -p 123456

2.LSB算法隐写

具体操作：在Stegsolve.jar分析data Extract的red blue green

3.guess算法隐写

具体操作：在kail下切换到outguess目录下，直接用命令即可

命令:outguess -r /root/angrybird.jpg(绝对路径) 123.txt(信息存放的文本)