Giflib应用

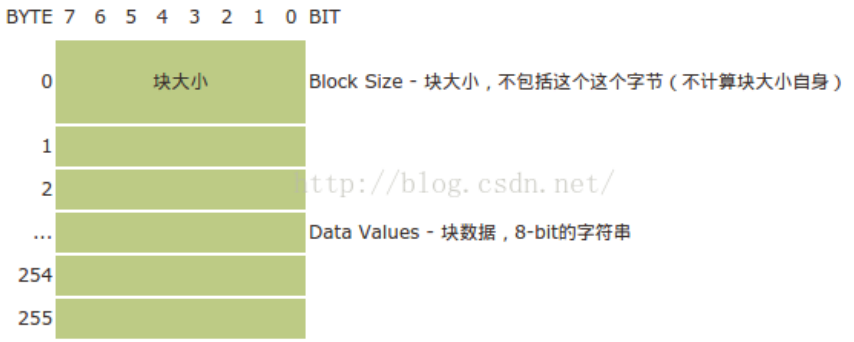
gif文件格式

GIF(Graphics Interchange Format，图形交换格式)文件是由 CompuServe公司开发的图形文件格式，版权所有，任何商业目的使用均须 CompuServe公司授权。

> *简单理解：*
>
> - *它就是颜色表和数据索引的组合，有全局颜色表和每帧数据的颜色表，每帧数据记录的不是RGB而是颜色表中的索引，解码帧的时候是拿索引查询颜色表中的数据，然后就能知道那个像素点的颜色*
> - *可以理解为gif里面就是包含了很多帧图片和帧与帧之间(10ms)的延时信息的文件*

GIF文件结构

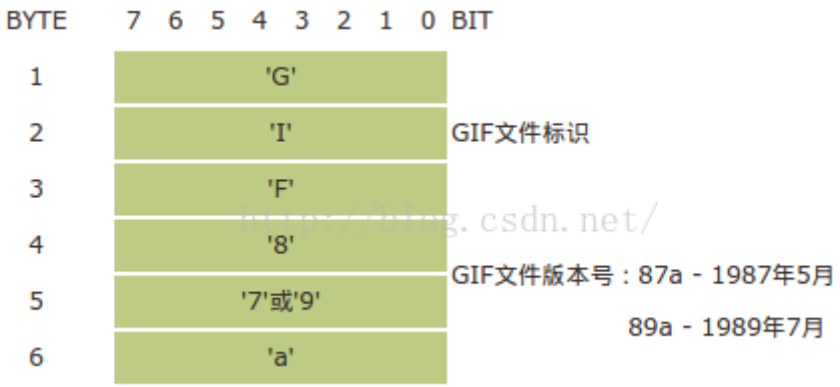
GIF文件内部是按块划分的，包括控制块（ Control Block ）和数据块（Data Sub-blocks）两种。控制块是控制数据块行为的，根据不同的控制块包含一些不同的控制参数；数据块只包含一些8-bit的字符流，由它前面的控制块来决定它的功能，每个数据块大小从0到255个字节，数据块的第一个字节指出这个数据块大小（字节数），计算数据块的大小时不包括这个字节，所以一个空的数据块有一个字节，那就是数据块的大小0x00。



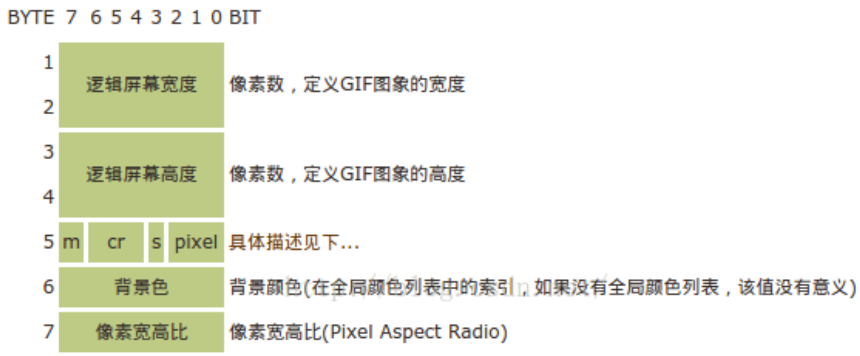
一个GIF文件的结构可分为文件头(File Header)、GIF数据流(GIF Data Stream)和文件终结器(Trailer)三个部分。文件头包含GIF文件署名(Signature)和版本号(Version)；GIF数据流由控制标识符、图象块(Image Block)和其他的一些扩展块组成；文件终结器只有一个值为0x3B的字符（';'）表示文件结束。下表显示了一个GIF文件的组成结构：

GIF署名用来确认一个文件是否是GIF格式的文件，这一部分由三个字符组成："GIF";文件版本号也是由三个字节组成,可以为"87a"或"89a".具体描述见下表:

![](.\assets\gif3.png



逻辑屏幕标识符(Logical Screen Descriptor)



m - 全局颜色列表标志(Global Color Table Flag)，当置位时表示有全局颜色列表，pixel值有意义.

cr - 颜色深度(Color ResoluTion)，cr+1确定图象的颜色深度.

s - 分类标志(Sort Flag)，如果置位表示全局颜色列表分类排列.

pixel - 全局颜色列表大小，pixel+1确定颜色列表的索引数（2的pixel+1次方）.

背景颜色索引 - 为背景颜色指向全局色表。背景颜色是指那些没有被图像覆盖的视屏部分的颜色。若全局色表标志位置为0，则该字段也被赋值0，并且被忽略。

象素高宽比 - 用于计算原图像中像素的近似高宽比。如果该字段的值为非0，则象素的高宽比由下面的公式计算： 高宽比 = (象素高宽比 + 15) / 64 该字段的取值范围从最宽的比值4：1到最高的比值1：4，递增的步幅为1/64。取值： 0 - 没有比值，1～255 - 用于计算的值。

全局色表标志 - 指示有没有全局色表，如果该标志位置1，则全局色表会紧接在该块之后出现。该位也用于解释是否选用背景颜色索引字段。若该位置1，则背景颜色索引字段的值将指向背景颜色表。

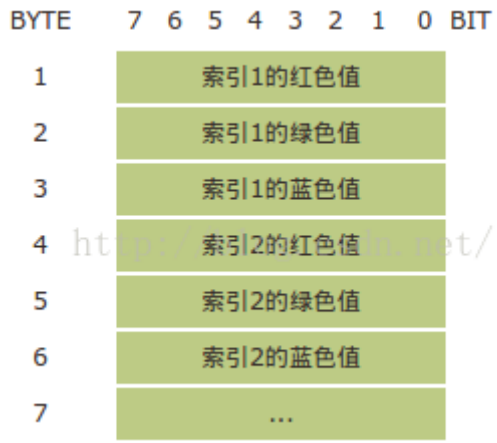
色彩方案 - 提供给原始图像的每个颜色的位数减1。这个值代表图像中所使用的整个调色板的大小，而不是图像中所使用的颜色的数量。例如，若该字段的值为3，则图像中所使用的调色板的每个色值占4位。

短标志 - 表明全局色表是否被排序。如果该位置1，则全局色表按照重要性递减的原则进行了排序。典型地，是按照颜色的使用频度进行递减排序，使用频度最高的颜色排在色表的最前面。这样便可帮助解码器选择最好的颜色子集来成象。

全局色表的尺寸 - 如果全局色表标志位置1，则该字段的值记录全局色表中所占用的字节数。

全局颜色列表(Global Color Table)

必须紧跟在逻辑屏幕标识符后面，每个颜色列表索引条目由三个字节组成，按R、G、B的顺序排列。



图象标识符(Image Descriptor)包含处理一个基于图像的表的必要参数，它是一幅图所必需的，图象标识符以0x2C(',')字符开始。一幅图像对应一个图象标识符，图象标识符后面跟着对应的图像数据。

BYTE	7	6	5	4	3	2	1	0	BIT	
1	0	0	1	0	1	1	0	0	图象标识符开始，固定值为','	
2	X方向偏移量									
3										
4	Y方向偏移量									
5										
6	图象宽度									必须限定在逻辑屏幕尺寸范围内
7										
8	图象高度									
9										
10	m	i	s	r	pixel				m - 局部颜色列表标志(Local Color Table Flag)	

置位时标识紧接在图象标识符之后有一个局部颜色列表，供紧跟在它之后的一幅图象使用；值否时使用全局颜色列表，忽略pixel值。

i - 交织标志(Interlace Flag)，置位时图象数据使用交织方式排列（详细描述...），否则使用顺序排列。

s - 分类标志(Sort Flag)，如果置位表示紧跟着的局部颜色列表分类排列.

r - 保留，必须初始化为0.

pixel - 局部颜色列表大小(Size of Local Color Table)，pixel+1就为颜色列表的位数

局部颜色列表(Local Color Table)

如果上面的局部颜色列表标志置位的话，则需要在这里（紧跟在图象标识符之后）定义一个局部颜色列表以供紧接着它的图象使用，注意使用前应线保存原来的颜色列表，使用结束之后回复原来保存的全局颜色列表。如果一个GIF文件即没有提供全局颜色列表，也没有提供局部颜色列表，可以自己创建一个颜色列表，或使用系统的颜色列表。局部颜色列表的排列方式和全局颜色列表一样：RGBRGB.....

图形控制扩展(Graphic Control Extension)

这一部分是可选的（需要89a版本），可以放在一个图象块(图象标识符)或文本扩展块的前面，用来控制跟在它后面的第一个图象（或文本）的渲染(Render)形式，组成结构如下：

BYTE	7	6	5	4	3	2	1	0	BIT	
1	扩展块标识								Extension Introducer - 标识这是一个扩展块，固定值0x21	
2	图形控制扩展标签								Graphic Control Label - 标识这是一个图形控制扩展块，固定值0xF9	
3	块大小								Block Size - 不包括块终结器，固定值4	
4	保留			处置方法			i	t	i - 用户输入标志；t - 透明色标志。详细描述见下...	
5	延迟时间								Delay Time - 单位1/100秒，如果值不为1，表示暂停规定的时间后再继续往下处理数据流	
6										
7	透明色索引								Transparent Color Index - 透明色索引值	
8	块终结器								Block Terminator - 标识块终结，固定值0	

处置方法(Disposal Method)：指出处置图形的方法，当值为：

0 - 不使用处置方法

1 - 不处置图形，把图形从当前位置移去

2 - 回复到背景色

3 - 回复到先前状态

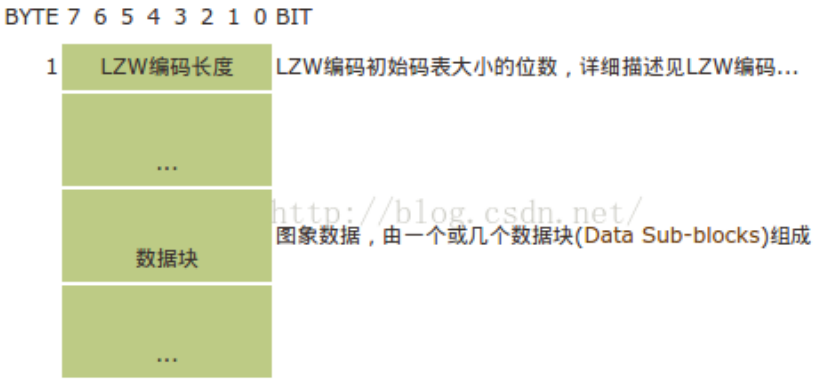
4-7 - 自定义

用户输入标志(Use Input Flag)：指出是否期待用户有输入之后才继续进行下去，置位表示期待，值否表示不期待。用户输入可以是按回车键、鼠标点击等，可以和延迟时间一起使用，在设置的延迟时间内用户有输入则马上继续进行，或者没有输入直到延迟时间到达而继续

透明颜色标志(Transparent Color Flag)：置位表示使用透明颜色

扩充引入 - 用于识别一个扩充块的开始，该字段为固定值0x21。 图像控制标号 - 识别当前块是否为图形控制扩充。该字段为固定值 0xF9。 块尺寸 - 块中所包含的字节数。从块尺寸字段开始到快结束符（不含结束符）。该字段包含固定值4。 配置方法 - 指示图像显示后的处理方法。值: 0 - 无指定的配置，解码器不需要做任何处理。 1 - 不做配值。图像将被留在原位置。 2 - 恢复背景颜色。图像所占的区域必须备恢复为背景颜色。 3 - 恢复以前的颜色。解码器需要将图像区域恢复为原来成象的颜色。 4-7 - 未定义。 用户输入标志 - 说明在继续处理之前是否需要用户输入。可以和输入延时一起使用。 透明标志 - 表明在透明索引字段是否给定透明索引。 延时 - 如果不为0, 该字段指定以1/100秒为单位的时延数。 透明索引 - 如果遇到透明索引，则显示设备的相关象素不被改变，继续处理下一个象素。 块终止符 - 这个0长度字段标志着图像控制扩充得结束。

基于颜色列表的图象数据(Table-Based Image Data)，由两部分组成：LZW编码长度(LZW Minimum Code Size)和图象数据(Image Data)。



LZW压缩算法在GIF中的应用

GIF图象数据使用了LZW压缩算法（详细介绍请看后面的『LZW算法和GIF数据压缩』），大大减小了图象数据的大小。图象数据在压缩前有两种排列格式：（由图象标识符的交织标志控制)。连续方式按从左到右、从上到下的顺序排列图象的光栅数据；交织图象按下面的方法处理光栅数据：

创建四个通道(pass)保存数据，每个通道提取不同行的数据： 第一通道(Pass 1)提取从第0行开始每隔8行的数据； 第二通道(Pass 2)提取从第4行开始每隔8行的数据； 第三通道(Pass 3)提取从第2行开始每隔4行的数据； 第四通道(Pass 4)提取从第1行开始每隔2行的数据；

行	通道1	通道2	通道3	通道4
0	1			
1				4
2			3	
3				4
4		2		
5				4
6			3	
7				4
8	1			
9				4
10			3	
11				4
12		2		
13				4
14			3	
15				4
16	1			
17				4
18			3	
19				4
20		2		

giflib API介绍

giflib中数据类型

```
typedef struct GifFileType {
    GifWord SWidth, SHeight;         /* Size of virtual
canvas */
    GifWord SColorResolution;        /* How many colors
can we generate? */
    GifWord SBackGroundColor;        /* Background color
for virtual canvas */
    GifByteType AspectByte;      /* Used to compute pixel
aspect ratio */
    ColorMapObject *SColorMap;       /* Global colormap,
NULL if nonexistent. */
    int ImageCount;                  /* Number of current
image (both APIs) */
    GifImageDesc Image;              /* Current image
(low-level API) */
    SavedImage *SavedImages;         /* Image sequence
(high-level API) */
    int ExtensionBlockCount;         /* Count extensions
past last image */
    ExtensionBlock *ExtensionBlocks; /* Extensions past
last image */
    int Error;               /* Last error condition
reported */
    void *UserData;                  /* hook to attach
user data (TVT) */
```

```
    void *Private;                  /* Don't mess with
this! */
} GifFileType;
```

> 1. *以S开头的变量标识屏幕（Screen）。SaveImages变量用来存储已经读取过得图像数据。*
> 2. *Private变量用来保存giflib私有数据，用户不应该访问该变量。*
> 3. *其他变量都标识当前图像。*

初始化giflib

- 文件流初始化giflib的代码如下：

```
    if ((GifFile = DGifOpenFileName(*FileName)) == NULL) {
        PrintGifError();
        exit(EXIT_FAILURE);
    }
```

- 也可以以文件句柄方式初始化giflib，例如：

```
    if ((GifFile = DGifOpenFileHandle(0)) == NULL) {
        PrintGifError();
        exit(EXIT_FAILURE);
    }
```

- 重点介绍一下怎样用自定义输入回调函数来初始化giflib，因为这个可以适配各式各样的数据输入方式比如网络等，参考代码如下：

```
    if ((GifFile = DGifOpen(&gif, gif_input_cb)) == NULL)
{
        PrintGifError();
        exit(EXIT_FAILURE);
    }
    gif_input_cb为自定义输入回调函数，该函数负责giflib的数据输
入。
```

我们另外需要注意以上三个函数都返回一个GifFileType类型的指针，该指针以后在调用giflib的函数时，用作第一个参数传入。

初始化屏幕

所有的gif图像共享一个屏幕（Screen），这个屏幕和我们的电脑屏幕不同，只是一个逻辑概念。所有的图像都会绘制到屏幕上面。

首先我们需要给屏幕分配内存：

```
    if ((ScreenBuffer = (GifRowType *)
            malloc(GifFile->SHeight * sizeof(GifRowType
*))) == NULL)
            GIF_EXIT("Failed to allocate memory required,
aborted.");
```

另外我们需要以背景颜色（GifFile->SBackGroundColor）初始化屏幕buffer。

```
    Size = GifFile->SWidth * sizeof(GifPixelType);/* Size
in bytes one row.*/
    if ((ScreenBuffer[0] = (GifRowType) malloc(Size)) ==
NULL) /* First row. */
        GIF_EXIT("Failed to allocate memory required,
aborted.");

    for (i = 0; i < GifFile->SWidth; i++)  /* Set its
color to BackGround. */
        ScreenBuffer[0][i] = GifFile->SBackGroundColor;
    for (i = 1; i < GifFile->SHeight; i++) {
       /* Allocate the other rows, and set their color to
background too: */
       if ((ScreenBuffer[i] = (GifRowType) malloc(Size))
== NULL)
           GIF_EXIT("Failed to allocate memory required,
aborted.");

      memcpy(ScreenBuffer[i], ScreenBuffer[0], Size);
   }
```

解码gif数据

我们上面已经提到gif数据是以顺序存放的块来存储的，DGifGetRecordType函数用来获取下一块数据的类型。因此解码gif数据的代码组织如下：

```
    do {
        if (DGifGetRecordType(GifFile, &RecordType) ==
GIF_ERROR) {
            PrintGifError();
           exit(EXIT_FAILURE);
        }

        switch (RecordType) {
            case IMAGE_DESC_RECORD_TYPE:
               break;
           case EXTENSION_RECORD_TYPE:
               break;
           case TERMINATE_RECORD_TYPE:
               break;
           default:            /* Should be traps by
DGifGetRecordType. */
               break;
```

```
        }
    } while (RecordType != TERMINATE_RECORD_TYPE);
```

循环解析gif数据，并根据不同的类型进行不同的处理。

处理图像数据

首先先介绍怎样处理IMAGE_DESC_RECORD_TYPE类型的数据。这代表这是一个图像数据块，这个图像数据需要绘制到前面提到的屏幕buffer上面，相应的代码如下：

```
    if (DGifGetImageDesc(GifFile) == GIF_ERROR) {
        PrintGifError();
        exit(EXIT_FAILURE);
    }
    Row = GifFile->Image.Top; /* Image Position relative
to Screen. */
    Col = GifFile->Image.Left;
    Width = GifFile->Image.Width;
    Height = GifFile->Image.Height;
    GifQprintf("\n%s: Image %d at (%d, %d) [%dx%d]:     ",
       PROGRAM_NAME, ++ImageNum, Col, Row, Width, Height);
   if (GifFile->Image.Left + GifFile->Image.Width >
GifFile->SWidth ||
     GifFile->Image.Top + GifFile->Image.Height > GifFile-
>SHeight) {
       fprintf(stderr, "Image %d is not confined to screen
dimension, aborted.\n",ImageNum);
       exit(EXIT_FAILURE);
   }
   if (GifFile->Image.Interlace) {
       /* Need to perform 4 passes on the images: */
       for (Count = i = 0; i < 4; i++)
           for (j = Row + InterlacedOffset[i]; j < Row +
Height;
                        j += InterlacedJumps[i]) {
               GifQprintf("\b\b\b\b%-4d", Count++);
               if (DGifGetLine(GifFile, &ScreenBuffer[j]
[Col], Width) == GIF_ERROR) {
                   PrintGifError();
                   exit(EXIT_FAILURE);
               }
           }
   }
   else {
       for (i = 0; i < Height; i++) {
           GifQprintf("\b\b\b\b%-4d", i);
          if (DGifGetLine(GifFile, &ScreenBuffer[Row++]
[Col],
               Width) == GIF_ERROR) {
               PrintGifError();
```

```
                exit(EXIT_FAILURE);
            }
        }
    }

    /* Get the color map */
    ColorMap = (GifFile->Image.ColorMap
        ? GifFile->Image.ColorMap
        : GifFile->SColorMap);
    if (ColorMap == NULL) {
        fprintf(stderr, "Gif Image does not have a
colormap\n");
        exit(EXIT_FAILURE);
    }

    DumpScreen2RGB(OutFileName, OneFileFlag,
           ScreenBuffer, GifFile->SWidth, GifFile-
>SHeight);
```

这里面有几点需要注意的是：

- gif数据交织处理，参照上面的代码。
- 另外注意调色板的选择，如果当前图像数据中有局部调色板就用局部调色板来解码数据，否则用全局调色板来解码数据。
- 屏幕数据的解码，根据你的显示要求选择输出格式。

解析屏幕数据，假设需要把数据转换成ARGB8888的格式，代码如下：

```
    static void DumpScreen2RGBA(UINT8* grb_buffer,
GifRowType *ScreenBuffer, int ScreenWidth, int
ScreenHeight)
    {
        int i, j;
        GifRowType GifRow;
        static GifColorType *ColorMapEntry;
        unsigned char *BufferP;

        for (i = 0; i < ScreenHeight; i++) {
            GifRow = ScreenBuffer[i];
            BufferP = grb_buffer + i * (ScreenWidth * 4);
            for (j = 0; j < ScreenWidth; j++) {
                ColorMapEntry = &ColorMap-
>Colors[GifRow[j]];
                *BufferP++ = ColorMapEntry->Blue;
                *BufferP++ = ColorMapEntry->Green;
                *BufferP++ = ColorMapEntry->Red;
                *BufferP++ = 0xff;
            }
       }
    }
```

解析屏幕数据的时候需要住处理透明色。这里先埋一个伏笔，等介绍完扩展块，再来重新实现这个函数。

处理扩展块

上面提到扩展块主要实现一些辅助功能，扩展块影响其后的图像数据解码。这里面比较重要的扩展块是图像控制扩展块（Graphic control extension）。可以参考giflib文档中的gif89.txt文件了解图像控制扩展块的详细内容。这个扩展块中有两个内容我们比较关心：

延时时间（delay time）——后面的图像延时多长时间再显示，如果解码线程不是主线程的话，可以在这里延时一下再处理后面的数据。

透明色（transparent color）——在后面的图像解码时，遇到同样的颜色值，则跳过不解码，继续处理后续的点。

如下是一种可供参考的处理方式：

```
    UINT32 delay = 0;
    if( ExtCode == GIF_CONTROL_EXT_CODE
        && Extension[0] == GIF_CONTROL_EXT_SIZE) {
        delay = (Extension[3] << 8 | Extension[2]) * 10;
        /* Can sleep here */
    }

    /* handle transparent color */
    if( (Extension[1] & 1) == 1 ) {
       trans_color = Extension[4];
   }
   else
       trans_color = -1;
```

这里GIF_CONTROL_EXT_CODE为0xF9表明该扩展块是一个图像控制扩展块，GIF_CONTROL_EXT_SIZE为4，图像控制扩 展块的大小。我们可以看到解析出delay信息之后，就地delay。解析出透明颜色值之后，则标识透明色，否则标识为-1。解析图片的时候可以根据透明 色的值进行相应的处理，参考如下解析图像的函数：

```
    static void DumpScreen2RGBA(UINT8* grb_buffer,
GifRowType *ScreenBuffer, int ScreenWidth, int
ScreenHeight)
    {
        int i, j;
        GifRowType GifRow;
        static GifColorType *ColorMapEntry;
        unsigned char *BufferP;

       for (i = 0; i < ScreenHeight; i++) {
           GifRow = ScreenBuffer[i];
           BufferP = grb_buffer + i * (ScreenWidth * 4);
           for (j = 0; j < ScreenWidth; j++) {
```

```
                if( trans_color != -1 && trans_color ==
GifRow[j] ) {
                    BufferP += 4;
                    continue;
                  }

                ColorMapEntry = &ColorMap-
>Colors[GifRow[j]];
                *BufferP++ = ColorMapEntry->Blue;
                *BufferP++ = ColorMapEntry->Green;
                *BufferP++ = ColorMapEntry->Red;
                *BufferP++ = 0xff;
            }
        }
   }
```

注意我们这里假设grb_buffer已经正确的初始化

giflib 源码获取

- 从Android源码中获取

> *\external\giflib 下面*

- 从官网下载

> *git clone https://git.code.sf.net/p/giflib/code giflib-code*