[**图像解码之二——使用libpng解码png图片**](http://www.cnblogs.com/xiaoxiaoboke/archive/2012/02/13/2349765.html)

上文《[图像解码之一——使用libjpeg解码jpeg图片](http://my.unix-center.net/~Simon_fu/?p=1026)》介绍了使用libjpeg解码jpeg图片。png图片应用也非常广泛，本文将会简单介绍怎样使用[开源libpng库](http://www.libpng.org/pub/png/libpng.html)解码png图片。

**libpng的数据结构**

    png\_structp变量是在libpng初始化的时候创建，由libpng库内部使用，代表libpng的是调用上下文，库的使用者不应该对这个变量进行访问。调用libpng的API的时候，需要把这个参数作为第一个参数传入。

    png\_infop变量，初始化完成libpng之后，可以从libpng中获得该类型变量指针。这个变量保存了png图片数据的信息，库的使用者可以修 改和查阅该变量，比如：查阅图片信息，修改图片解码参数。在早期的版本中直接访问该变量的成员，最新的版本建议是通过API来访问这些成员。

**libpng的使用**

0、判断是否为libpng数据

    这步是可选的，在利用libpng继续数据处理之前，可以调用png\_sig\_cmp函数来检查是否为png数据，请参阅[libpng手册](http://www.libpng.org/pub/png/libpng-manual.txt)了解详细内容。

1、初始化libpng

1: /\* Create and initialize the png\_struct with the desired error handler

2: \* functions. If you want to use the default stderr and longjump method,

3: \* you can supply NULL for the last three parameters. We also supply the

4: \* the compiler header file version, so that we know if the application

5: \* was compiled with a compatible version of the library. REQUIRED

6: \*/

7: png\_ptr = png\_create\_read\_struct(PNG\_LIBPNG\_VER\_STRING,

8: png\_voidp user\_error\_ptr, user\_error\_fn, user\_warning\_fn);

    初始化libpng的时候，用户可以指定自定义错误处理函数，如果不需要指定自定义错误处理函数，则传NULL即可。 png\_create\_read\_struct函数返回一个png\_structp变量，前面已经提到该变量不应该被用户访问，应该在以后调用 libpng的函数时传递给libpng库。

    如果你需要提供自定义内存管理模块则需要调用png\_create\_read\_struct\_2来完成对libpng的初始化：

1: png\_structp png\_ptr = png\_create\_read\_struct\_2

2: (PNG\_LIBPNG\_VER\_STRING, (png\_voidp)user\_error\_ptr,

3: user\_error\_fn, user\_warning\_fn, (png\_voidp)

4: user\_mem\_ptr, user\_malloc\_fn, user\_free\_fn)

2、创建图像信息——png\_infop变量

1: /\* Allocate/initialize the memory for image information. REQUIRED. \*/

2: info\_ptr = png\_create\_info\_struct(png\_ptr);

3: if (info\_ptr == NULL)

4: {

5: png\_destroy\_read\_struct(&png\_ptr, png\_infopp\_NULL, png\_infopp\_NULL);

6: return (ERROR);

7: }

    如前面所说，用户将会通过png\_infop变量来获得图片的信息，设置图片解码参数等。

3、设置错误返回点

    上文libjpeg解码jpeg图片中提到用setjmp/longjmp函数来处理异常。libpng库默认集成这种机制来完成异常处理，如下代码初始化错误返回点：

1: /\* Set error handling if you are using the setjmp/longjmp method (this is

2: \* the normal method of doing things with libpng). REQUIRED unless you

3: \* set up your own error handlers in the png\_create\_read\_struct() earlier.

4: \*/

5: if (setjmp(png\_jmpbuf(png\_ptr)))

6: {

7: /\* Free all of the memory associated with the png\_ptr and info\_ptr \*/

8: png\_destroy\_read\_struct(&png\_ptr, &info\_ptr, png\_infopp\_NULL);

9: /\* If we get here, we had a problem reading the file \*/

10: return (ERROR);

11: }

    正如上面注释中提到情况，只有在初始化libpng的时候未指定用户自定义的错误处理函数情况下，才需要设置错误返回点。如果设置了用户自定义的错误处理函数，libpng将会调用用户自定义错误处理函数，而不会返回到这个调用点。

    当libpng库出现错误的时候，libpng将会自动调用longjmp函数返回到这个点。在这个点我们可以进行必要的清理工作。

4、设置libpng的数据源

    我在上文《[图像解码之一——使用libjpeg解码jpeg图片](http://my.unix-center.net/~Simon_fu/?p=1026)》中提到，一个好的代码库应该能够运行用户输入各式各样的数据，而不能把输入数据定死。libpng在这方面做得非常的好，它提供了默认的文件输入流的支持，并且提供了用户自定义回调函数来完成png数据的输入。

    对于文件流数据数据设置代码如下：

1: /\* One of the following I/O initialization methods is REQUIRED \*/

2: def streams /\* PNG file I/O method 1 \*/

3: /\* Set up the input control if you are using standard C streams \*/

4: png\_init\_io(png\_ptr, fp);

    用户自定义回调函数设置libpng数据源的代码如下：

1: /\* If you are using replacement read functions, instead of calling

2: \* png\_init\_io() here you would call:

3: \*/

4: png\_set\_read\_fn(png\_ptr, (void \*)user\_io\_ptr, user\_read\_fn);

5: /\* where user\_io\_ptr is a structure you want available to the callbacks \*/

    如果你已经使用png\_sig\_cmp函数来检查了png数据，需要调用png\_set\_sig\_bytes函数来告诉libpng库，这样库处理数据的时候将会跳过相应的数据，具体请参考[libpng手册](http://www.libpng.org/pub/png/libpng-manual.txt)。

5、png图像处理

    这步有两种设置方案一种称为高层处理，一种称为底层处理。

**高层处理**

    当用户的内存足够大，可以一次性读入所有的png数据，并且输出数据格式为如下libpng预定义数据类型时，可以用高层函数，下libpng预定义数据类型为：

PNG\_TRANSFORM\_IDENTITY          No transformation  
  
PNG\_TRANSFORM\_STRIP\_16          Strip 16-bit samples to

                                              8 bits

PNG\_TRANSFORM\_STRIP\_ALPHA     Discard the alpha channel

PNG\_TRANSFORM\_PACKING           Expand 1, 2 and 4-bit

                                              samples to bytes

PNG\_TRANSFORM\_PACKSWAP        Change order of packed

                                              pixels to LSB first

PNG\_TRANSFORM\_EXPAND            Perform set\_expand()

PNG\_TRANSFORM\_INVERT\_MONO   Invert monochrome images

PNG\_TRANSFORM\_SHIFT              Normalize pixels to the

                                              sBIT depth

PNG\_TRANSFORM\_BGR                 Flip RGB to BGR, RGBA

                                              to BGRA

PNG\_TRANSFORM\_SWAP\_ALPHA     Flip RGBA to ARGB or GA

                                              to AG

PNG\_TRANSFORM\_INVERT\_ALPHA  Change alpha from opacity

                                              to transparency

PNG\_TRANSFORM\_SWAP\_ENDIAN   Byte-swap 16-bit samples

PNG\_TRANSFORM\_GRAY\_TO\_RGB   Expand grayscale samples

                                              to RGB (or GA to RGBA)

    高层读取函数如下：

1: /\*

2: \* If you have enough memory to read in the entire image at once,

3: \* and you need to specify only transforms that can be controlled

4: \* with one of the PNG\_TRANSFORM\_\* bits (this presently excludes

5: \* dithering, filling, setting background, and doing gamma

6: \* adjustment), then you can read the entire image (including

7: \* pixels) into the info structure with this call:

8: \*/

9: png\_read\_png(png\_ptr, info\_ptr, png\_transforms, png\_voidp\_NULL);

    该函数将会把所有的图片数据解码到info\_ptr数据结构中。png\_transforms为整型参数，为上面libpng预定义的数据类型进行or操作得到。调用了该函数，就不可以再调用png\_set\_transform函数来设置输出数据。

    该函数相当于调用底层函数（下文将会介绍）如下调用顺序：

a)调用png\_read\_info函数获得图片信息。

b)根据png\_transforms所指示的，调用png\_set\_transform设置输出格式转换的函数。

c)调用png\_read\_image来解码整个图片的数据到内存。

d)调用png\_read\_end结束图片解码。

    当你调用png\_read\_png之后，则可以调用如下函数得到png数据：

1: row\_pointers = png\_get\_rows(png\_ptr, info\_ptr);

**底层处理**

**a)读取输入png数据的图片信息：**

1: /\* The call to png\_read\_info() gives us all of the information from the

2: \* PNG file before the first IDAT (image data chunk). REQUIRED

3: \*/

4: png\_read\_info(png\_ptr, info\_ptr);

    该函数将会把输入png数据的信息读入到info\_ptr数据结构中。

**b)查询图像信息**

    前面提到png\_read\_info将会把输入png数据的信息读入到info\_ptr数据结构中，接下来需要调用API查询该信息。

1: png\_get\_IHDR(png\_ptr, info\_ptr, &width, &height, &bit\_depth, &color\_type,

2: &interlace\_type, int\_p\_NULL, int\_p\_NULL);

**c)设置png输出参数（转换参数）**

    这步非常重要，用户可以指定输出数据的格式，比如RGB888，ARGB8888等等输出数据格式。通过png\_set\_xxxxx函数来实现，例如如下代码：

1: // expand images of all color-type and bit-depth to 3x8 bit RGB images

2: // let the library process things like alpha, transparency, background

3: if (bit\_depth == 16)

4: png\_set\_strip\_16(png\_ptr);

5: if (color\_type == PNG\_COLOR\_TYPE\_PALETTE)

6: png\_set\_expand(png\_ptr);

7: if (bit\_depth<8)

8: png\_set\_expand(png\_ptr);

9: if (png\_get\_valid(png\_ptr, info\_ptr, PNG\_INFO\_tRNS))

10: png\_set\_expand(png\_ptr);

11: if (color\_type == PNG\_COLOR\_TYPE\_GRAY ||

12: color\_type == PNG\_COLOR\_TYPE\_GRAY\_ALPHA)

13: png\_set\_gray\_to\_rgb(png\_ptr);

    如上代码将会把图像转换成RGB888的数据格式。这种转换函数还很多，请参阅[libpng手册](http://www.libpng.org/pub/png/libpng-manual.txt)了解他们的作用。

    虽然有很多设置输出参数的函数可以调用，但是用户的需求是无限的，很多输出格式libpng并不是原生支持的，比如YUV565，RGB565，YUYV 等等。幸好libpng提供了自定义转换函数的功能，可以让用户注册转换回调函数给libpng库，在libpng对输出数据进行转换的时候，先对 png\_set\_xxxxx函数设置的参数进行转换，最后将会调用用户自定义的转换函数进行转换。

1: png\_set\_read\_user\_transform\_fn(png\_ptr,

2: read\_transform\_fn);

    read\_transform\_fn为用户自定义的数据转换函数。具体实现可以参考pngtest.c中的实现。

    另外你可以通过png\_set\_user\_transform\_info告诉libpng你的转换函数的用户自定义数据结构和输出数据的详细信息，比如颜 色深度，颜色通道（channel）等等。你可能会问为什么要告诉libpng呢？libpng将会根据这些信息来更新png图片详细信息，后面会介绍。 定义如下：

1: png\_set\_user\_transform\_info(png\_ptr, user\_ptr,

2: user\_depth, user\_channels);

    usr\_ptr是用户自定义的数据结构，在用户自定义转换函数read\_transform\_fn中可以通过png\_get\_user\_transform\_ptr函数得到该数据结构，例如：

1: voidp read\_user\_transform\_ptr =

2: png\_get\_user\_transform\_ptr(png\_ptr);

**d)更新png数据的详细信息**

    经过前面的设置png数据的图片信息肯定会有一些变化，则需要调用png\_read\_update\_info函数更新图片的详细信息：

1: png\_read\_update\_info(png\_ptr, info\_ptr);

    该函数将会更新保存于info\_ptr变量中的图片数据信息，然后可以再调用png\_get\_IHDR重新查询图片信息。

**e)读取png数据**

    可以到用png\_read\_image函数，一次性把所有的数据读入内存，例如：

1: png\_read\_image(png\_ptr, row\_pointers);

    也可以调用png\_read\_rows一次读入1行或多行到内存中，比如：

1: for (y = 0; y < height; y++)

2: {

3: png\_read\_rows(png\_ptr, &row\_pointers[y], png\_bytepp\_NULL, 1);

4: }

**f)结束读取数据**

通过png\_read\_end结束读取png数据，代码如下：

1: /\* Read rest of file, and get additional chunks in info\_ptr - REQUIRED \*/

2: png\_read\_end(png\_ptr, info\_ptr);

6、释放libpng的内存

    调用png\_destroy\_read\_struct来释放libpng的内存，代码如下：

1: /\* Clean up after the read, and free any memory allocated - REQUIRED \*/

2: png\_destroy\_read\_struct(&png\_ptr, &info\_ptr, png\_infopp\_NULL);

    至此png数据解码，全部完成了。

**总结：**

    通过上面的介绍，我们可以发现相对于libjpeg，libpng的扩展性非常好。我们基本上没有任何修改libpng库的需求，它的对外接口提供了足够的灵活性，允许我们扩展。从这个角度来讲，libpng库非常值得我们学习它的对外接口的定义。