PIL的API参考

- 13.ImageOps Module
- 14.ImagePalette Module
- 15.ImagePath Module
- 16.ImageQt Module
- 17.ImageSequence Module
- 18.ImageStat Module
- 19.ImageTk Module
- 20.ImageWin Module (Windows-only)
- 21.ExifTags Module
- 22.TiffTags Module
- 23.PSDraw Module
- 24.PixelAccess Class
- 25.PyAccess Module
- 26.PIL Package (autodoc of remaining modules)”
- 27.Plugin reference

1.Image Module

Image模块提供类一个同名的Image类，是PIL中最重要的一个模块，它提供了诸多图像操作，诸如创建、打开、显示、保存、合成、裁剪、滤波等功能，还可以获取图像等直方图和通道。

1.1简单实例

下面的脚本程序加载了一个图像，将其旋转了45度，并用外部查看器显示它（一般在Unix系统使用xv，Windows系统使用paint程序查看）。

```
#打开、旋转和显示一个图像（使用默认查看器）
from PIL import Image
im=Image.open('example.jpg')
im.rotate(45).show()
```

下面的脚本程序为当前文件夹下所有的JPEG图像建立了一个128x128尺寸的缩略图。

```
from PIL import Image
import glob,os
size=(128,128)
for infile in glob.glob('*.jpg'):
    file,ext=os.path.splitext(infile)
    im=Image.open(infile)
    im.thumbnail(size,Image.ANTIALIAS)
    im.save(infile+'.thumbnail','JPEG')
```

1.2函数

1.2.1打开图像

`PIL.Image.open(file,mode='r')`

打开和识别给定的图像文件。这是一个懒人方法，用这个函数识别文件，文件仍然是打开的，并且实际图像在你进行操作（或者称为load()方法）之前并没有读入进来。

file：一个文件名，或者文件对象。这个文件对象必须支持read()、seek()、tell()方法，而且将会以二

进制形式打开

mode：如果使用mode参数，mode必须是'r'；

return：Image对象

```
from PIL import Image
im1=Image.open("test1.jpg")
im2=Image.open("test2.jpg","r")
```

1.2.2图像处理

`PIL.Image.alpha_composite(im1,im2)`

阿尔法混合两个图像。

im1：第一个图像

im2：第二个图像，必须和im1有相同的颜色模式和尺寸

return：Image对象

`PIL.Image.blend(im1,im2,alpha)`

使用alpha常数，用两个图像的插值创建一个新的图像，公式如下：

out=im1*(1.0-alpha)+im2*alpha

im1:第一个图像

im2:第二个图像，必须和im1有相同的色彩模式和尺寸

alpha：插值因子，如果alpha=0.0，返回im1，如果alpha=1.0，返回im1.

return：Image对象

```
from PIL import Image
im1=Image.open("test1.jpg")
im2=Image.open("test2.jpg")
im=Image.blend(im1,im2,0.3)
im.show()
```

`PIL.Image.composite(im1,im2,mask)`

使用一个透明度掩码创建复合图像。

im1:第一个图像

im2:第二个图像，必须与im1有相同的模式和尺寸

mask：一个掩码图像，可以是“1”、“L”或者“RGB”模式，必须和其他两个图像尺寸相同

```
from PIL import Image
im1=Image.open("test1.jpg")
im2=Image.open("test2.jpg")
r,g,b=im1.split()
im=Image.composite(im1,im2,g)
im.show()
```

`PIL.Image.eval(im,*args)`

将一个函数应用到图像的每个像素点，多通道的图像，这个函数将被应用到每一个通道上。

函数对每一个像素进行求值，因此不能使用随机组件或者其他生成器。

im：输入的图像

*args：一个函数对象，取一个整型参数

return：Image对象

```
from PIL import Image
im1=Image.open("test1.jpg")
def fun(x):
    return x*0.5
im=Image.eval(im1,fun)
im.show()
im1.show()
```

`PIL.Image.merge(mode,bands)`

将一系列单通道图像融合为一个多通道图像。

mode：要生成的图像的模式

bands：一个包含了多个单通道图像的序列，每一个单通道图像的尺寸必须相同

return：Image对象

```
from PIL import Image
im1=Image.open("test1.jpg")
im2=Image.open("test2.jpg")
r1,g1,b1=im1.split()
r2,g2,b2=im2.split()
ims=[r1,g2,b2]
im=Image.merge("RGB",ims)
im.show()
```

1.2.3创建图像

`PIL.Image.new(mode,size,color=0)`

使用给定的模式和尺寸创建图像。

mode：新图像的模式

size：新图像的尺寸

color：新图像的颜色，默认是黑色。给定新图像颜色时，如果是单通道图像，color的值必须是integer或者floating point值，如果是多通道图像，color为元组。创建RGB图像时，你也可以使用ImageColor模块支持的颜色字符串。如果color值为None，则图像不会初始化。

return：Image对象

```
from PIL import Image
#创建一个128*128尺寸的红色图像
im=Image.new("RGB",(128,128),"#FF0000")
im.show()
#创建一个128*128尺寸的黑色图像
im=Image.new("RGB",(128,128))
#创建一个128*128尺寸的红色图像
im=Image.new("RGB",(128,128),"red")
```

`PIL.Image.fromarray(obj,mode=None)`

从导出数组接口的对象创建图像内存。

obj：数组接口的对象

mode：使用的模式

return：图像内存

`PIL.Image.frombytes(mode,size,data,decoder_name='raw',*args)`

从缓存区中的像素数据创建图像内存的副本，最简单的使用格式需要（mode,size,data）这三个参数。你可以使用任何PIL支持的解码器。注意这个函数只能解码像素数据，而不是完整的图像。如果需要一个字符串中的

完整的图像文件，可使用BytesIO对象打包它，再用open()加载。

data：一个包含给定模式的raw数据的字节缓存

decoder_name:解码器名称

args：给定解码器的参数

return：Image对象

`PIL.Image.frombuffer(mode,size,data,decoder_name='raw',*args)`

在字节缓存中创建引用像素数据的图像内存。这个函数与frombytes类似，但是这个函数使用了字节缓存的数据，这意味着对原始缓存对象的修改将会反映在这个图像上。并不是所有的模式都支持共享内存，这个函数支持L、RGB、RGBA和CMYK模式。注意这个函数只能解码像素数据，而不是完整的图像。如果需要一个字符串中的完整的图像文件，可使用BytesIO对象打包它，再用open()加载。

mode：图像的模式

size：图像的尺寸

data：包含了给定模式的raw数据的字节或者其他缓存对象

decoder_name：使用的解码器

args：给定解码器的参数，在使用默认解码器（raw）时，建议提供完整的参数

frombuffer(mode,size,data,"raw",mode,0,1)

return：Image对象

1.2.4注册插件

这一类函数用于开发插件，应用软件的开发者可以忽略它们。

`PIL.Image.register_open(id,factory,accept=None)`

`PIL.Image.register_mine(id,minetype)`

`PIL.Image.register_save(id,driver)`

`PIL.Image.register_extension(id,extension)`

1.3Image类

1.3.1Image类实例方法

这个类用于创建一个Image对象，尽量不要直接调用图像构造器。

- open()
- new()
- frombytes()

Image类的实例有以下方法。除非有特殊说明，否则所有的方法都返回一个保存着结果图像的新的Image类。

`Image.convert(mode=None,matrix=None,dither=None,palette=0,colors=256)`

返回一个转化过的图像副本。在P模式下，这个方法通过调色板解释像素点。如果mode被漏掉了，一个mode将会被选择来保证图像中所有的信息以及调色板都能被描绘出来。

现在的版本支持L、RGB、CMYK模式之间所有可能的转化。matrix参数只支持L和RGB模式。当将一个彩色图像转化为灰度图像时，将使用ITU-R 601-2 luma变换：

L=R*299/1000+G*587/1000+B*144/1000

将L和RGB模式图像转化成1模式的默认方法是使用Floyd-Steinberg抖动来计算近似原始图像的亮度值。如果diter值为NONE，所有的非零值将会被设置为255。如果想使用其他的阈值，请使用Point()方法。

mode：要转化成的模式

matrix：一个可选的转化矩阵。如果使用这个参数，必须是4或者16元组的浮点值

dither：抖动方法，用于将RGB转化成P，或者将RGB、L模式转化成1模式时。默认值是NONE或者FLOYDSTEINBERG

palette：用于将RGB模式转化到P模式的调色板，可选的palette为WEB和ADAPTIVE

colors：ADAPTIVE调色板使用的颜色数，默认是256

return：Image对象

下面的例子将RGB图像（依据ITU-R 709的线性校准，使用D65亮度）转化成CIE XYZ颜色空间：

```
rgb2xyz=(0.412453,0.357580,0.180423, 0,
    0.212671, 0.715160, 0.072169, 0,
    0.019334, 0.119193, 0.950227, 0 )
out=im.convert("RGB",rgb2xyz)
```

`Image.copy()`

复制图像，这个方法用于当你想往图像上粘贴东西又想保持原始图像时。

return：Image对象

`Image.crop(box=None)`

从图像中返回一个矩形区域。这是一个懒人方法，对原始图像的改变可能反映也可能不反映到剪切区域中，要想切断这一联系，可以使用load()方法。

box：要剪切的矩形空间，4元组，（左，上，右，下）

return：Image对象

`Image.draft(mode,size)`

配置图像文件加载器以保证返回一个尽可能接近给定模式和尺寸的图像。例如，你可以使用这个方法在加载时将一个彩色的图像转化为灰度图像，或者从PCD文件提取一个128x192版本图像。

注意这个方法在正确的地方才有效。如果图像已经加载，这个方法将无效。

`Image.filter(filter)`

使用给定的滤波器对图像进行滤波。可用的滤波器列表详见ImageFilter模块。

filter：滤波器内核

return：Image对象

`Image.getbands()`

返回一个包含图像各个通道名称的元组。例如，使用在RGB图像上时将返回("R","G","B")

return：包含通道名称的元组

`Image.getbbox()`

计算图像中非零区域的边界框。

return：边界框以4元组形式返回，（左，上，右，下），如果图像是空的，返回None

`Image.getcolors(maxcolors=256)`

返回图像中使用的颜色的列表。

maxcolors：颜色值的最大值，如果超过这个值，返回None。默认值是256.

return：一个未排序的(count,pixel)值列表

`Image.getdata(band=None)`

返回图像内容到一个包含像素值的序列对象。这个序列对象是扁平化（一维化）的，所以行1的值直接跟在行0后面。

注意这个方法返回的序列对象是一个PIL的内部数据类型，只支持某些序列操作。如果想将其转化成普通的序列，请使用list()函数。

band：要返回的通道名，默认是返回所有通道。如果要返回单一通道，请传递这个通道的索引值，例如0返回RGB模式的R通道数据

return：一个序列式的对象

`Image.getextrema()`

返回图像各个通道的最大值和最小值。

return：对于单通道图像，返回一个包含最大值和最小值的2元组，对于多通道的图像，返回一个包含着多个2元组的元组。

`Image.getpixel(xy)`

返回给定坐标的像素值。

xy：一个坐标，（x，y）

return：像素点的值，如果是多通道图像返回一个元组

`Image.histogram(mask=None,extrema=None)`

返回图像的直方图，这个直方图以像素数量值的列表形式返回，每一个数量值代表一个像素值在源图像中的数

量。如果图像是多通道的，每一个通道的直方图将会链接起来（例如RGB的直方图包含768个值）。这个方法将1模式的图像当成L模式来处理。如果给定mask的值，这个方法返回mask图像非零区域的直方图。mask图像必须和原图有相同的尺寸，可以是1模式或者L模式的图像。

mask：一个可选的掩码图像

return：一个包含着像素值数量值的列表

`Image.paste(im,box=None,mask=None)`

将另一个图像粘贴到这个图像。如果两个图像的模式不同，被粘贴的图像将会被转化成和这个图像一样的模式。如果不使用Image对象，也可以使用包含像素值的整数值或者元组。这个方法将会用给定的颜色填满这个区域。当创建RGB模式图像时，你也可以使用ImageColor模块支持的颜色字符串。

如果给定mask，这个方法只将mask指定的区域更新，你可以使用1、L或者RGBA模式的图像作为mask。如果mask值是255，给定图像的值将会被完全复制，如果是0，将保持原图的值，其他中间值可以用于透明度效果。

注意如果粘贴RGBA模式图像，alpha通道将会被忽略。你可以把一个RGBA图像同时作为粘贴图像和掩码图像使用。

im：粘贴源图像，或者像素值

box：一个4元组，代表要被粘贴到的区域。如果使用2元组，将会被当成粘贴区域的左上角。如果缺省这个值，源图像将会被粘贴到载体图像的左上角。如果一个图像对象被给定为这个方法的第二个参数，box的值将会为（0，0），而这个图像对象将会被解释成mask值

mask：可选的掩码图像

`Image.point(lut,mode=None)`

通过一个查找表或者函数会议这个图像。

lut：一个查找表，每个通道的查找表包含256个值（如果self.mode=T mode=L，值为65336个）。也可以是一个函数，这个函数必须是单参数的。函数将会尽可能应用于所有像素值，得到的表将会被应用到图像的所有通道。

mode：输出图像的模式，默认值与输入图像相同。在现在版本中，只有当源图像为L或P模式输出图像为1

模式时，或者源图像为I模式输出图像为L模式时才会用到这个参数。

return：Image对象

`Image.putalpha(alpha)`

添加或者替换原图的alpha通道。如果图像没有alpha通道，它将会被转化称LA或者RGBA模式。

alpha：新的alpha通道，可以是1或者L模式的图像，要与原图的尺寸相同，也可以是一个正数或者其他形式的颜色值。

`Image.putdata(data,scale=1.0,offset=0.0)`

将像素数据复制到这个图像。这个方法从序列对象复制数据，从图像的左上角开始，知道图像结束或者序列结束。scale和offset用于调整序列值：pixel=value*scale+offset。

data：一个序列对象

scale：可选的缩放值，默认是1.0

offset：可选的便宜值，默认是0.0

`Image.putpalette(data,rawmode='RGB')`

将一个调色板附加到图像上。这个图像必须是P或者L模式，调色板序列必须包含768个整型值，其中每三个值一组表示像素索引的红绿蓝颜色值。也可以用8-bit字符串代替整型序列。

data：一个调色板的序列（列表或者字符串）

`Image.putpixel(xy,value)`

修改给定位置的像素值。单通道的图像要修改称的像素颜色值是一个单一值，多通道则为元组。

注意这个方法速度相对来说比较慢，如果要进行大量替换请使用paste()或者ImageDraw模块。

xy：像素坐标，以（x，y）的形式给定

value：要替换成的像素值

`Image.quantize(colors=256,method=None,kmeans=0,palette=None)`

将图像用指定的颜色数转化成P模式。

colors：期望的颜色数，不大于256

method：0=median cut中值切分，1=maximum coverage最大覆盖，2=fast octree快速八叉树，3=libimagequant

kmean：一个整型数

palette：PIL.ImagingPalette调色板的量化

`Image.resize(size,resample=0)`

返回一个调整尺寸的图像副本。

size：需要的尺寸，2元组（宽，高）

resample：一个可选的重采样滤波器，可以为PIL.Image.NEAREST(最近临接)/BILINER(在2x2环境中线性插值)/BICUBIC(在4x4环境中三次样条插值)。如果这个参数缺省，或者图像为1或者P模式，这个参数将会被设置成PIL.Image.NEARSEST

return：一个Image对象

`Image.rotate(angle,resample=0,expand=0,center=None,translate=None)`

返回一个旋转了的图像副本。围绕图像中心顺时针旋转给定的角度。

angle：顺时针方向要旋转的角度

resample：可选的重采样滤波器。可选值与resize中相同。

expand:可选的扩展标志。如果为真则将输出图像扩展到能保持完整的旋转后图像的尺寸，如果为假或者缺省则使输出图像与原始图像尺寸相同。

center：可选的中心坐标，2元组，默认值是图像中心

translate：可选的post-rotate转化，2元组

`Image.save(fp,format=None,**params)`

用给定的文件名保存图像。如果没有给定format，将会使用文件名的扩展名作为format。关键词选项用于想作者提供附加说明，如果不使用这个选项，它将会被忽略，可用的选项在说明文档中Image File Formats。

你可以用一个文件对象来代替文件名，如果这样做，必须指定format。文件对象必须能使用seek、tell、write方法，而且以二进制形式打开。

fp：文件名（字符串），Pathlib.Path对象或者file对象

format：图像的格式

options：图像作者的额外选项

`Image.seek(frame)`

在序列文件中寻址到给定的帧。如果你寻找的帧超出了队列，这个方法将引发一个EOFError。当打开一个序列时，这个库将会自动寻址到0号帧。

注意在现在到版本中，大部分格式到序列只允许你寻址到下一帧。

frame：帧号，从0开始

`Image.show(title=None,command=None)`

显示图像。这个方法主要用于排错。在Unix平台上，这个方法将会将图像保存到一个临时的PPM文件，并调用xv或者display应用程序来打开它，具体使用哪个取决于那一个能被找到。在macOS上，这个方法将会将图像保存成一个BMP的临时文件，然后用本地的预览程序打开。在Windows系统，将会保存成一个临时的BMP文件，并使用标准的BMP显示程序显示它。

title：可选的用于图像窗口的标题

command：用于显示图像的命令

`Image.split()`

将图像切分成多个单通道。这个方法返回一个单图像通道的的元组。例如，切分一个RGB图像，将创建三个新的图像，每一个包含一个原始通道的副本（red，green，blue）。

return：一个包含多个通道的元组

`Image.tell()`

返回当前帧的号。

return：帧号，从0开始

`Image.thumbnail(size,resample=3)`

将图像转成缩略图。这个方法将图像本身修改成不大于给定尺寸的某个版本的缩略图。这个方法计算一个合适的缩略图尺寸来保持图像的外形，它将会调用draft()方法来配置文件阅读器，并且最终改变图像的尺寸。

注意这个方法修改了Image对象。如果你想继续使用完整的图像，请在原始图像的副本（copy()）上使用这个方法。

size：需要的尺寸

resample：可选的重采样滤波器。可用的滤波器与resize()相同，默认值是BICUBIC（2.5.0版本中是NEAREST）。

`Image.tobitmap(name='image')`

返回图像转化成的X11 bitmap

`Image.tobytes(encoder_name='raw',*args)`

返回保存成字节对象的图像

`Image.tostring(*args,**kw)`

返回保存成字符串对象的图像

`Image.transform(size,method,data=None,resample=0,fill=1)`

图像变形。这个方法使用给定变形方法，创建一个与原图模式相同的给定尺寸的新图像。

size：输出图像的尺寸

method：变形方法，可选的有PIL.Image.EXTENT(矩阵剖分)/AFFINE(仿射变换)/PERSPECTIVE(透视变换)/QUAD(将正方形映射为矩形)/MESH(在一个操作中绘制多个源四边形)

data：变形方法的附加信息

resample：可选的重采样滤波器。可用的滤波器与resize()相同，默认值是NEAREST

return：Image对象

`Image.transpose(methond)`

图像移动。（以90度的幅度翻转或者旋转）

method：可选的有PIL.Image.FLIP_LEFT_RIGHT/FLIP_TOP_BOTTOM/ROTATE_90/ROTATE_180/ROTATE_270/TRANSPOSE

return：翻转或者旋转后的图像副本

`Image.verify()`

核对图像的内容。对于从文件中读取的图像，这个方法不经过解码就试图验证文件是否损坏。如果这个方法发现了任何问题，他将会引发一个合适的例外。如果你想在使用这个方法后加载图像，图像必须被重新打开。

`Image.load()`

为图像分配内存，加载像素数据。在通常情况下，你并不用调用这个方法，因为Image类在第一次访问时自动加载打开的图像。此方法将关闭与图像关联的文件。

return：一个图像存取对象 PixelAccess类或PIL.PyAcess

`Image.close()`

如果可能，关闭文件指针。这个操作将会破坏图像核心并释放内存。此后图像的数据将无法再被使用。此功能只需要关闭没有自己文件读取和关闭方法的图像。

1.3Image的属性Attributes

Image类的实例含有以下属性：

`PIL.Image.format`

源文件的文件格式。对于由库本身创建的图像（通过工程方法或者通过在现有图像上使用方法），这个属性将被设置为None。

type：string、None

`PIL.Image.mode`

图像模式。是一个说明图像使用的像素格式的字符串。典型的值是1、L、RGB、CMYK。

type：string

`PIL.Image.size`

图像尺寸，以像素为单位。尺寸以2元组的形式给出（宽，高）。

type：(width,height)

`PIL.Image.width`

图像的宽度，以像素为单位。

type：int

`PIL.Image.height`

图像的高度，以像素为单位。

type：int

`PIL.Image.palette`

调色板。如果是P模式，这个属性必须是ImagePalette类的实例，其他情况，这个属性设置成None。

type：ImagePalette、None

`PIL.image.info`

保存与图像相关联的数据的字典。这个字典用于文件处理程序传递的从文件中读取的各种非图像信息。

大多数方法在返回新图像时忽略字典；因为键值不规范，所以不可能知道操作是否影响这个字典。如果你后面需要这些信息，请保留从打开方法返回的信息字典。除非在别处注明，否则这个字典不会影响保存文件。

type：dict

2.ImageChops（“Channel Operations”）Module

ImageChops模块包含了一系列图形等算数操作，比如图像特效、图像组合、算法绘制等。ImageChops大部分操作指用于8bit图像，例如L和RGB。更多的预处理操作在ImageOps模块中。这一次，大部分的通道操作只在8-bit图像中可用（L、RGB）。

2.1函数

大部分通道操作使用一个或者两个图像作为参数，并且返回一个新的图像。除非有其他说明，通道操作的返回值总是被截断在0到MAX（在这个模块中MAX是255）范围内。

`PIL.ImageChops.add(image1,image2,scale=1.0,offset=0.0)`

将两个图像相加，将结果除以缩放，加上偏移量。如果缺省，scale默认值1.0，offset默认值0.0。out=(image1+image2)/scale+offset。

return：Image对象

`PIL.ImageChops.add_modulo(image1,image2)`

在不截断返回值的情况下将两个图像相加。out=(image1+image2)%MAX

return：Image对象

`PIL.ImageChops.blend(image1,image2,alpha)`

使用透明度值混合两个图像，是PIL.Image.Image.blend()的别名。

return：Image对象

`PIL.ImageChops.composite(image1,image2,mask)`

使用透明度掩码复合两个图像，是PIL.Image.Image.composite()的别名。

reutrn：Image对象

`PIL.ImageChops.constant(image,value)`

使用给定的灰度值填充一个通道。

return：Image对象

`PIL.ImageChops.darker(image1,image2)`

逐像素对比两个图像，返回一个更暗的像素值到一个新的图像。out=min(image1,image2)

return：Image对象

`PIL.ImageChops.difference(image1,image2)`

逐像素将两个图像相减，返回结果的绝对值到一个新的图像。out=abs(image1,image2)

return：Image对象

`PIL.ImageChops.duplicate(image)`

复制通道，是PIL.Image.Image.copy()的别名。

return：Image对象

`PIL.ImageChops.invert(image)`

倒置图像的像素值（通道）。out=Max-image.

return：Image对象

`PIL.ImageChops.lighter(image1,image2)`

逐像素对比两个图像，返回更亮的像素值到一个新图像。out=max(image1,image2)。

return：Image对象

`PIL.ImageChops.logical_and(image1,image2)`

逻辑加两个图像。out=(image1 and image2)%MAX

return：Image对象

`PIL.ImageChops.logical_or(image1,image2)`

逻辑或两个图像。out=(image1 or image2)%MAX

return：Image对象

`PIL.ImageChops.multiply(image1,image2)`

将两个图像相互叠映。如果将一个图像与纯黑色图像相乘，结果是黑色图像。如果将图像与纯白图像相乘，结果不变。out=image1*image2/MAX

return：Image对象

`PIL.ImageChops.offset(image,xoffset,yoffset=None)`

返回一个偏移了给定距离的图像副本。数据从边缘绕回。如果yoffset缺省，默认值将会设置成与xoffset相同。

xoffset：水平向的偏移量

yoffset：竖直向的偏移量

return：Image对象

`PIL.ImageChops.screen(image1,image2)`

将两个图像的倒置像素值叠映。out=MAX-(MAX-image1)*(MAX-image2)/MAX

return：Image对象

`PIL.ImageChops.subtract(image1,image2,scale=1.0,offset=0.0)`

将两个图像相减，结果除以缩放量，加上偏移量。scale默认值1.0，offset默认值0.0。

out=(image1-image2)/scale+offset

return：Image对象

`PIL.ImageChops.subtract_modulo(image1,image2)`

将两个图像相减，不截断返回值。out=(image1-image2)%MAx

return：Image对象

3.ImageColor Module

ImageColor模块包含来颜色表和从css3样式颜色到RGB颜色元组的转换器。这个模块使用于PIL.Image.Image.new()和ImageDraw等模块中。

3.1颜色名Color Names

ImageColor模块支持如下字符串格式：

- 16进制颜色说明符，以#rgb或者#rrggbb格式给定，例如#ff0000表示纯红色。
- RGB函数，以rgb(red,green,blue)格式给定，颜色值必须在0到255范围内。或者颜色值也可以用百分比形式给出，例如rgb(255,0,0)也可以写成rgb(100%,0%,0%)，都是纯红色。
- 色彩饱和度(HSL)函数，以hsl(hue,saturation%,lightness%)给定，其中hue是一个0到360之间的角度值(red=0,green=120,blue=240)，饱和度值是0%到100%之间(gray=0%,full color=100%)，亮度值在0%到100%之间(black=0%,normal=50%,white=100%)，例如hsl(0,100%,50%)代表纯红色
- 通用HTML颜色名。ImageColor模块提供了一些X Window系统和大多数网络浏览器支持的，140标准的颜色名。颜色名不区分大小写。例如，red和Red都表示纯红色。

3.2函数

`PIL.ImageColor.getrgb(color)`

将一个颜色值字符串转化成RGB元组。如果语法分析失败，这个方法将会引发一个ValueError例外。

color：一个颜色值字符串

return：(red,green,blue[,alpha])

`PIL.ImageColor.getcolor(color,mode)`

与getrgb()相同，但是将非彩色或者调色板模式的图像的RGB值转化成灰度值。

color：一个颜色值字符串

return：(graylevel[,alpha]) or (red,green,blue[,alpha])

4.ImageCms Module

ImageCms模块使用基于Kevin Cazabon的PyCMS库开发的LittleCMS2颜色管理引擎，提供了对颜色管理的支持。

5.ImageDraw Module

ImageDraw模块为Image对象提供了基本的2D绘图功能，例如可以用它来创建图像、注释或修饰现有的图像，为WEB应用实时产生各种图形等。

更多的PIL高级会图库，详见aggdraw module

5.1简单实例

在图像上画一个灰色的十字

```
from PIL import Image,ImageDraw
im=Image.open('lena.pgm')
draw=ImageDraw.Draw(im)
draw.line((0,0)+im.size,fill=128)
draw.line((0,im.size[1],im.size[0],0),fill=128)
del draw
#写数据
im.save(sys.stdout,"PNG")
```

绘制一个部分不透明的文本

```
from PIL import Image,ImageDraw,ImageFont
base=Image.open('Pillow/test/images/lena.png').convert('RGBA')
txt=Image.new('RGBA',base.size,(255,255,255,0))
fnt=ImageFont.truetype('Pillow/test/fonts/FreeMono.ttf',40)
d=ImageDraw.Draw(txt)
d.text((10,10),"Hello",font=fnt,fill(255,255,255,128))
d.text((10,60),"World",font=fnt,fill(255,255,255,255))
out=Image.alpha_composite(base,txt)
out.show()
```

5.2概念

坐标

图像接口使用了和PIL本身相同的坐标系统，以(0,0)为左上角。

颜色

为了指明颜色，你可以像在PIL.Image.Image.new()或PIL.Image.Image.putpixel()中一样使用数字或者元组。1、L和I模式的图像，使用整型数。RGB图像使用3元组或者颜色名。绘图图层将会自动分配颜色索引，只要你不绘制超过256色。

颜色名

支持的颜色名详见3.1。

字体

PIL可以使用bitmap字体或者OpenType / TrueType字体。

Bitmap字体本保存成PIL自己的格式，每种字体将会一贯地包含两个文件，一个后缀名.pil一个后缀名为.pbm。前者包含字体度量，后者是光栅数据。

要加载bitmap字体，请使用ImageFont模块中的加载函数。

要加载OpenType / TrueType字体，使用ImageFont模块中的Truetype函数。注意这个函数依赖于第三方库，并不是所有的PIL版本中都可用。

5.3函数

`PIL.ImageDraw.Draw(im,mode=None)`

在给定的图像上创建一个可以用于绘图的对象。注意这个图像将会被修改。

im：要在其中进行绘图的图像

mode：可选的颜色模式。对于RGB图像，这个参数可以使用RGB或者RGBA。对于所有其他的模式，这个参数必须和im的模式相同。如果缺省，默认值是im的模式。

5.4方法

`PIL.ImageDraw.Draw.arc(xy,start,end,fill=None)`

在给定的方框中画一个从开始角度到结束角度的圆弧（圆形轮廓的一部分）。

xy：定义方框的四个点，[(x0,y0),(x1,y1)]或者[x0,y0,x1,y1]形式的序列

start：开始画图的角度值，角度从3点钟方向开始顺时针计算

end：结束画图的角度值

fill：绘制圆弧的颜色

`PIL.ImageDraw.Draw.bitmap(xy,bitmap,fill=None)`

在给定的位置绘制位图，使用给定的颜色填充非零区域。这个位图必须是有效的透明遮罩（模式1）或者磨砂形式（模式L或者RGB）。这个函数效果与image.paste(xy,color,bitmap)相同。

`PIL.ImageDraw.Draw.chord(xy.start,end,fill=None,outline=None)`

与arc()类似，但是是用直线链接结束点。

xy：定义绘图方框的四个点，[(x0,y0),(x1,y1)]或者[x0,y0,x1,y1]形式的序列

outline：轮廓的颜色

fill：填充的颜色

`PIL.ImageDraw.Draw.ellipse(xy,fill=None,outline=None)`

在给定的方框中绘制椭圆。

xy：定义绘图方框的四个点，[(x0,y0),(x1,y1)]或者[x0,y0,x1,y1]形式的序列

outline：轮廓的颜色

fill：填充的颜色

`PIL.ImageDraw.Draw.line(xy,fill=None,width=0)`

在给定的坐标之间画线。

xy：2元组序列，[(x,y),(x,y),...]或者[x,y,x,y,x,y....]

fill：画线的颜色

width：线的粗度值，以像素为单位。注意连接处处理得不好，所以宽折线不好看。

`PIL.ImageDraw.Draw.pieslice(xy,start,end.fill=None,outline=None)`

画扇形，绘制弧线并同时绘制从结束点到绘图方框中心的直线。

xy：定义方框的四个点，[(x0,y0),(x1,y1)]或者[x0,y0,x1,y1]形式的序列

start：开始画图的角度值，角度从3点钟方向开始顺时针计算

end：结束画图的角度值

fill：填充的颜色

outline：边框的颜色

`PIL.ImageDraw.Draw.point(xy,fill=None)`

在给定的坐标画单独的点。

xy：2元组序列，[(x,y),(x,y),...]或者[x,y,x,y,x,y....]

fill：画点的颜色

`PIL.ImageDraw.Draw.polygon(xy,fill=None,outline=None)`

画多边形。这个多边形边框包含给定坐标点之间的直线，并添加一个从第一个坐标点到最后一个的直线。

xy：2元组序列，[(x,y),(x,y),...]或者[x,y,x,y,x,y....]

fill：填充的颜色

outline：边框的颜色

`PIL.ImageDraw.Draw.rectangle(xy,fill=None,outline=None)`

画矩形。

xy：定义绘图方框的四个点，[(x0,y0),(x1,y1)]或者[x0,y0,x1,y1]形式的序列

outline：轮廓的颜色

fill：填充的颜色

`PIL.ImageDraw.Draw.shape(shape,fill=None,outline=None)`

画一个形状。这个方法是实验性的。

`PIL.ImageDraw.Draw.text(xy,text,fill=None,font=None,anchor=None,spacing=0,align=left)`

在给定的位置画字符。

xy：文字的左上角

text：要绘制的文字。如果它包含任何换行字符，这个文字将被传递到mulitline_text()

fill：文字的颜色

font：一个ImageFont实例

anchor：超链接?

spacing：如果文字被传递到mulitline_text()，这个参数表示两行之间的像素数

align：对齐方式，如果被传递到mulitline_text()，这个参数可以是left、center、right

`PIL.ImageDraw.Draw.mulitline_text(xy,text,fill=None,anchor=None,spacing=0,align='left')`

在给定的位置画字符。

xy：文字的左上角

text：要绘制的文字

fill：文字的颜色

font：一个ImageFont实例

anchor：超链接?

spacing：表示两行之间的像素数

align：对齐方式，可以是left、center、right

`PIL.ImageDraw.Draw.textsize(text,font=None,spacing=0)`

返回给定文字的尺寸，以像素为单位。

text：要计算的文字，如果它包含任何换行字符，这个文字将被传递到mulitline_textsize()

font：一个ImageFont实例

spacing：如果文字被传递到mulitline_textsize()，这个参数表示两行之间的像素数

`PIL.ImageDraw.Draw.mulitline_textsize(text,font=None,spacing=0)`

返回给定文字的尺寸，以像素为单位。

text：要计算的文字

font：一个ImageFont实例

spacing：表示两行之间的像素数

6.ImageEnhance Module

ImageEnhance模块中包含来一系列等用于图像增强等类，包括Color类、Brightness类、Contrast类、Shapness类。

6.1简单实例

改变图像的锐度

```
from PIL import ImageEnhance
enhancer=ImageEnchance.Sharpness(image)
for i in range(8):
    factor=i/40
    enchancer.enchance(factor).show("Sharpness %f" % factor)
```

6.2类

所有的增强类都使用相同的通用接口，包含一个单一方法。

`PIL.ImageEnhance._Enhance`

`enhance(factor)`

返回增强过的图像。

factor：一个控制增强的浮点数值。factor为1.0时返回原始图像的副本，更小的数代表更少的颜色（亮度值、对比度值等），更大的数代表更多的颜色。这个参数没有限制。

return：Image对象

`PIL.ImageEnhance.Color(image)`

调整图像的颜色平衡。

这个类可以用类似彩色电视机上控件的方式来调整图像的颜色平衡。黑白图像的factor将被给定为0.0，原始图像的factor给定为1.0。

`PIL.ImageEnhance.Contrast(image)`

调整图像的对比度。

这个类用于控制图像的图像的对比度，类似于电视机上的对比度控制。灰度图像的增强因子被给定为0.0，原始图像的增强因子是1.0。

`PIL.ImageEnhance.Brightness(image)`

调整图像的亮度值。

这个类用于控制图像的亮度值。黑色图像的增强因子是0.0，原始图像的影响因子是1.0。

`PIL.ImageEnhance.Sharpness(image)`

调整图像的锐度。

这个类用于调整图像的锐度值。模糊图像的增强因子是0.0，原图的是1.0，锐化后的图像是2.0。

7.ImageFile Module

ImageFile模块提供了图像打开和保存等功能。另外它还提供了一个Parser类，这个类可以逐片解码一张图像（例如当从网络中接收一张图像）。这个类等截扣与标准的sgmllib和xmllib模块一样。

7.1简单实例

解析一张图像

```
from PIL import ImageFile
fp=open("hopper.pgm","rb")
p=ImageFile.parser()
while 1:
    s=fp.read(1024)
    if not s:
        break
    p.feed(s)
im=p.close()
im.save("copy.jpg")
```

7.2Parser

`PIL.ImageFile.Parser`

递增图像解析。这个类实现了来标准的用户接口的进入和关闭。

`close()`

（用户）关闭流。

return：一个Image类

`feed(data)`

（用户）向解析器提供数据。

data：一个字符串型缓存

`reset()`

（用户）重置解析器。注意在创建解析器后，你只能立即调用这个方法，不能重复使用解析器实例。

7.3PyDecoder

`PIL.ImageFile.PyDcoder`

格式解码器的Python实现。重写这个类并且子啊解码方法中增加解码逻辑。

`cleanup()`

重写执行解码器的特定清除。

`decode(buffer)`

重写执行解码的过程。

buffer：包含要解码的数据的字节对象。如果handles_eof设置了，这个buffer参数将会被清空，并且self.fd将会被设置。

return：一个元组（消耗的字节，错误代码）。如果解码结束消耗的字节将返回小于0。错误代码来自ERRORS。

`init(args)`

重写执行解码器的特定初始化。

args：从tile入口得到的参数数组?

`set_as_raw(data,rawmode=None)`

从原始数据流中设置内部图像的简便方法。

data：要设置的字节

rawmode：解码器使用的原始数据模式。如果不被具体说明，将会被被设置成图像的模式。

`setfd(fd)`

被图像文件调用，用于设置Python文件对象。

fd：Python文件对象

`setimage(im,extents=None)`

被图像文件调用，用于设置解码器的核心输出图像。

im：核心图像对象

extents：一个4元组(x0,y0,x1,y1)，定义tile矩形。

8.ImageFilter Module

ImageFilter模块中包含了一组预定义好的滤波器的定义，它可以用在filter()方法中。

8.1简单实例

```
from PIL import ImageFilter
im1=im.filter(ImageFilter.BLUR)
im2=im.filter(ImageFilter.MinFilter(3))
```

```
im3=im.filter(ImageFilter.MinFilter)#与MinFilter(3)相同
```

8.2滤波器Filters

该模块包含了如下滤波器：

- BLUR 模糊
- CONTOUR 轮廓
- DETAL 细节
- EDGE_ENHANCE 边缘
- EDGE_ENHANCE_MORE 更多边缘
- EMBOSS 浮雕
- FIND_EDGE 寻找边缘
- SMOOTH 平滑
- SMOOTH_MORE 更多平滑
- SHARPEN 锐化

8.3类

`PIL.ImageFilter.GaussianBlur(radius=2)`

高斯模糊滤波器

radius：模糊半径

`PIL.ImageFilter.UnsharpMask(radius=2,percent=150,threshold=3)`

反锐化掩码滤波器

radius：模糊半径

percent：反锐化强度，用百分比的形式

threshold：控制将被锐化的最小亮度变化的阈值

`PIL.ImageFilter.Kernel(size,kernal,scale=None,offset=0)`

创建一个卷积核。当前版本只支持3x3、5x5整型或者浮点型的核。当前版本中，核只能应用于L和RGB图像。

size：核尺寸，以（宽，高）形式给定。在当前版本中，必须是(3,3)或者(5,5)

kernal：包含核重量的的序列

scale：比例因子。如果给定，结果的每个像素都将被除以这个值。默认值是核重量之和。

offset：偏移量。如果给定，这个值将会在除以比例因子后被加到结果上

`PIL.ImageFilter.RankFilter(size,rank)`

创建一个等级滤波器。这个等级滤波器在给定尺寸的窗口中排序所有像素，返回等级值。

size：核尺寸，以像素为单位

rank：要选取的像素值。使用0作为最小滤波，size*size/2作为中值滤波，size*size-1作为最大滤波

`PIL.ImageFilter.MedianFilter(size=3)`

创建一个中值滤波器。在给定尺寸的窗口中选取像素值的中值。

size：核尺寸，以像素为单位

`PIL.ImageFilter.MinFilter(size=3)`

创建一个最小值滤波器。在给定尺寸的窗口中选取像素值的最小值。

size：核尺寸，以像素为单位

`PIL.ImageFilter.MaxFilter(size=3)`

创建一个最大值滤波器。在给定尺寸的窗口中选取像素值的最大值。

size：核尺寸，以像素为单位

`PIL.ImageFilter.ModeFilter(size=3)`

创建一个模式滤波器。在给定尺寸的方框中选取出现最频繁的像素值。只出现一两次的像素值将会被忽略，如果没有像素值出现超高两次，将保持原始像素值。

size：核尺寸，以像素为单位

9.ImageFont Module

ImageFont模块定义类一个同名的ImageFont类，这个类的实例中存储这bitmap字体，需要与ImageDraw.Draw.text()方法一起使用。PIL使用自己的字体文件格式存储bitmap字体。你可以使用pilfont工具包将BDF和PCF字体描述器（X windows字体格式）转化为这种格式。

9.1简单实例

```
from PIL import ImageFont,ImageDraw
draw=ImageDraw.Draw(image)
font=ImageFont.load("arial.pil")
draw.text((10,10),"hello",font=font)
font=ImageFont.truetype("arial.ttf",15)
draw.text((10,25),"world",font=font)
```

9.2函数

`PIL.ImageFont.load(filename)`

加载字体文件。这个函数从给定的bitmap字体文件加载字体对象，返回相关的字体对象

filename：字体文件名

return：字体对象

`PIL.ImageFont.load_path(filename)`

加载字体文件。与load()相同，但是是从python路径中搜索bitmap字体。

filename：字体文件名

return：字体对象

`PIL.ImageFont.truetype(font=None,size=10,index=0,encoding)`

加载Truetype或者OpenType字体文件，创建一个字体对象。这个函数从给定的文件中加载一个字体对象，根据给定的尺寸和字体创建一个字体对象。这个函数需要_imagingft服务。

font：一个truetype字体文件。在Windows中，如果这个没有找到这个文件名的文件，加载器将从Windows的fonts/文件夹中查找。

size：字体的尺寸，以像素点为单位

index：要加载的字体面（默认值是第一个可用面）

encoding：要使用的字体编码（默认值是Unicode）。通用的编码是"unic"(Unicode)，"symb"(Microsoft symbol)，"ADOB"(Adobe Standard)，"ADBE"(Adobe Expert)和"armn"(Apple Roman)。查阅FreeType文档以获得更多信息。

return：一个字体对象

`PIL.ImageFont.load_default()`

加载“聊胜于无”的默认字体

return：一个字体对象

9.3方法

`PIL.ImageFont.ImageFont.getsize(text)`

return：（宽，高）

`PIL.ImageFont.ImageFont.getmask(text,mode)`

为文本创建一个bitmap（位图）。如果字体使用抗锯齿，这个位图必须是L模式，并且最大值是255。其他情况，模式是1。

text：要渲染的文本

mode：通过一些图形驱动程序来指定那种模式更好。如果缺省，渲染器可能返回任何一个模式。注意模式始终是一个字符串，以简化C-level实现。

return：一个由PIL.Image.core接口模块定义的PIL内部存储内存实例

10.ImageGrab Module (macOS and Windows-only)

ImageGrab模块可用于复制屏幕或剪贴板内容到PIL图像内存中，目前只有macOS、Windows系统可用。

`PIL.ImageGrab.grab(bbox=None)`

屏幕快照。方框内的像素在Windows系统中以RGB模式返回，在macOS中以RGBA模式返回。如果方框范围缺省，将会将整个屏幕复制。

bbox：要复制的范围，默认是整个屏幕

return：一个图像

`PIL.ImageGrab.grabclipboard()`

如果剪贴板里有图像，对它进行一次快照。

return：在Windows系统中返回一个图像、文件名列表或者None。注意如果返回的是文件列表，文件名可能不代表图像文件。在macOS中返回一个图像或者None。

11.ImageMath Module

ImageMath模块可用于计算对图像数学操作的表达式的值。这个模块提供了一个求值函数，这个函数可以像数学操作一样计算图像，使用一个表达式字符串和一个或多个图像进行计算。描述字符串是符合Python语法的一个语句。

11.1简单实例

使用ImageMath模块

```
from PIL import Image,ImageMath
im1=Image.open("image1.jpg")
im2=Image.open("image2.jpg")
out=ImageMath.eval("convert(min(a,b),'L')",a=im1,b=im2)
out.save("result.png")
```

`PIL.ImageMath.eval(expression,environment)`

在给定的环境中计算表达式。在当前的版本中，ImageMath模块只支持单通道图像。要处理多通道图像的画，请使用split()和merge()方法。

expression：一个使用标准Python表达式语法的字符串。除了标准操作符之外，还可以使用11.2.4中的函数。

environment：将图像名映射到实例的字典。如上例所示，你可以使用一个或者多个关键字参数来代替字典。注意这个名称必须是有效的Python标识符。

return：一个图像、整型值、浮点数值或者一个像素元组，与表达式有关。

11.2表达式语法

表达式必须是标准Python表达式，但是它们在非标准环境中运算。你可以依旧可以使用PIL方法，也包括下面的运算符和函数。

11.2.1标准运算符

你可以使用标准的算术运算符，例如加(+)、减(-)、乘(*)、除(/)。这个模块也支持一元减号(-)、模运算(%)、乘方(**)操作符。

注意，所有操作都是用32位整数或32位浮点值来完成的，如有必要。例如，如果你将两个8位图像相加，结果将是一个32位整数图像。如果用8位图像中与浮点常量相加，结果将是32位浮点图像。

你可以使用下面描述的convert()、float()和int()函数进行强制转化。

11.2.2位运算符

这个模块也提供了单个位的操作。这些操作包含与(&)、或(|)、非(~)和异或(^)。

注意在位运算之前所有的操作数将会被转化成32位有符号整型数。这意味着你如果对一个灰度图像进行～操作，将会得到负值。你可以使用&操作来屏蔽掉不想要的位。浮点数图像上位运算符不可用。

11.2.3逻辑运算符

逻辑运算符，例如and、or和not作用于整个图像而不是单独像素。一个空的图像（所有的像素值都是0）将会被当成false。所有的其他图像将会被当成true。

注意and和or返回最后一个操作数，而不总是返回布尔值。

11.2.4内置函数

这些函数将应用于单个像素

`abs(image)`

绝对值

`convert(image,mode)`

将图像转化成给定模式，模式要以字符常量形式给定

`float(image)`

将图像转化成32位浮点数。和convert(image,"F")相同

`int(image)`

将图像转化成32位整型数。和convert(image,"I")相同。注意1-bit和8-bit图像将会自动转化成32-bit整型图像以获得正确的结果。

`max(image1,image2)`

最大值

`min(image1,image2)`

最小值

12.ImageMorph Module

ImageMorph模块提供了对图像的形态学运算。

`PIL.ImageMorph.LutBuilder(patterns=None,op_name=None)`

Bases：对象

一个从描述性语言建立变形对照表的类。输入模式patterns是字符串序列的列表，例如：

```
4:(...
    .1.
    111)->1
```

（空白包括换行符被忽略）。选项4描述了一系列的对称操作（在这种情况下，4-rotation），模式描述为：

- . 或者X-忽略
- 1-像素打开
- 0-像素关闭

这个操作的结果在“->”之后描述。

操作：

- 4-4路旋转
- N-否定

- 1–如果没有其他操作时的虚拟操作（必须始终给定一个操作）
- M–镜像

例子：

```
lb=LutBuilder(pattern=["4:(... .1. 111)->1"])
lut=lb.build_lut()
add_patterns(patterns)
build_default_lut()
build_lut()
#将所有模式编译到形态学对照表中
get_lut()
```

`PIL.ImageMorph.MorphOp(lut=None,op_name=None,patterns=None)`

Bases：对象

一个用于二元形态操作的类。

```
apply(image)
#在一个图像上使用一个形态学操作
#返回一个包含改变的像素和变形的图像数量的元组
get_on_pixels(image)
#获得所有二进制图像中打开的像素的列表
#返回一个包含所有符合条件像素坐标元组(x,y)的列表。
load_lut(filname)
#从mrl文件加载一个操作
match(image)
#获得符合形态学操作的坐标的列表
save_lut()
#将一个操作保存到mrl文件
set_lut()
#从外部资源设置lut
```

13.ImageOps Module

ImageOps模块中包含一系列的“已完成的”图像处理操作。这个模块有一些实验性，大部分操作都是只能用于L和RGB模式。它可以完成直方图均衡、裁剪、量化、镜像等操作。

`PIL.ImageOps.autocontrast(image,cutoff=0,ignore=None)`

最大化（标准化）图像对比。此函数计算输入图像的直方图，从直方图中移去截断百分比的最亮和最暗像素，变换图像使最暗像素变成黑色（0），最亮变成白色（255）。

image：要处理的图像

cutoff：直方图中要截断的像素的百分比

ignore：背景像素值（使用None表示没有背景）

return：image

`PIL.ImageOps.colorize(image,black,white)`

将灰度图像上色。黑色和白色参数必须是RGB元组；这个函数计算一个颜色楔来将原图中所有黑色映射到第一个颜色，将所有白色映射到第二个颜色。

image：要上色的图像

black：用于黑色输入像素的颜色值

white：用于白色输入像素的颜色值

return：image

`PIL.ImageOps.crop(image,border=0)`

移除图像的边框。图像的4个边将会移除相同数量的像素。这个函数可以应用于所有模式的图像。

image：要裁剪的图像

boder：要移除的像素数

return：一个image

`PIL.ImageOps.deform(image,deformer,resample=2)`

畸变变形一个图像。

image：要进行畸变的图像

deformer：一个deformer对象，任何能使用getmesh方法的对象都可用使用

resample：一个可选的重采样滤波器。和PIL.Image.transform函数中的可用值一样

return：image

`PIL.ImageOps.equalize(image,mask=None)`

均衡化图像的直方图。这个函数应用一个非线性映射于输入图像，为了在输出图像中均衡分布灰度值。

image：要进行均衡化的图像

mask：一个可选的掩码。如果给定，只有被掩码选中的像素将会被包含在分析中

return：image

`PIL.ImageOps.expand(image,border=0,fill=0)`

为图像添加边框。

image：要扩展的图像

boder：边框宽度，以像素为单位

fill：像素填充值。默认是0（黑色）

return：image

`PIL.ImageOps.fit(image,size,method=0,bleed=0.0,centering=(0.5,0.5))`

返回一个给定尺寸的裁剪过的图像，裁剪到要求的长宽比和尺寸。这个方法有Kevin Cazabon提供。

size：要输出的尺寸，以像素为单位，以（宽，高）元组形式给定

method：要使用的重采样方法，默认是PIL.Image.NEAREST

bleed：移除图像外面的边框（应用于全部四个边。值是小数形式的百分比（使用0.01表示百分之一），默认值是0，无边框）

centering：控制裁剪位置。使用（0.5，0.5）来进行中心裁剪（例如如果裁剪宽度，取左侧50%，右侧50%）。使用（0.0，0.0）将会从左上角开始裁剪（即如果裁剪宽度，取全部右侧值，如果裁剪高度，取全部底部值）。使用（1.0，1.0）将会从右下角开始裁剪（即如果裁剪宽度，将取全部左侧值，如果裁剪高度将会取全部的上方值）

return：image

`PIL.ImageOps.flip(image)`

纵向翻转图像（从顶部到底部）

image：要翻转的图像

retunr：image

`PIL.ImageOps.grayscale(image)`

将图像转化成灰度图。

image：要转化的图像

return：image

`PIL.ImageOps.invert(image)`

倒置（取反）图像。

image：要进行倒置的图像

return：image

`PIL.ImageOps.mirror(image)`

水平翻转图像（从左到右）。

image：要进行镜像翻转的图像

return：image

`PIL.ImageOps.posterize(imge,bits)`

从每个颜色通道移除一定数量的字节。

image：要进行色调分离的图像

bits：每个通道要保持的字节数

return：image

`PIL.ImageOps.solarize(image,threshold=128)`

倒置所有高于阈值的像素值

image：要进行过曝处理的图像

threhold：所有高于这个灰度等级的像素值将会被取反

return：image

14.ImagePalette Module

ImagePalette模块包含一个同名等类来表示调色板图像的调色板。这个模块并没有确定的记录，但是他从2001年开始没有变化，所以如果需要的话，你可以使用它来安全地阅读源代码或者推测图像的内部结构。这个ImagePalette类包含几个方法，但是它们都被标记为“实验性的”。如果需要请阅读它们。

`PIL.ImagePalette.ImagePalette(mode='RGB',palette=None,size=0)`

调色板映射图像的调色板。

mode：调色板使用的模式。默认是RGB

palette：一个可选的调色板。如果给定，它必须是字节数组，即一个在0-255之间的整型数组或列表，长度必须是mode中的颜色数和size的积。这个列表必须安装通道对齐（所有R的值必须连接在G和B值的前面）。默认值是每个通道0到255

size：一个可选的调色板尺寸。如果给定，不能大于等于256。默认是0

```
#一些方法都是实验性的
getcolor(color)
#个顶一个RGB元组，分配调色板入口
getdata()
#以合适的格式获得调色板内容，im.putpalette的低级原型
save(fp)
#将调色盘保存到文本文件
tobytes()
#将调色板转化成字节
tostring()
#将调色板转化成字节
```

15.ImagePath Module

ImagePath模块用于存储和操作二维向量数据，Path类可以被传递到ImageDraw模块的方法中。

16.ImageQt Module

ImageQt模块包含了对从PIL图像创建PyQt4和PyQt5的Qimage类的支持。

`ImageQt.ImageQt(image)`

从PIL的Image对象创建一个ImageQt对象，这个类是QtGui.QImage的子类，这意味着你可以直接将结果对象传递到PyQt的API函数或方法中。

这个操作目前支持1、L、P、RGB和RGBA模式的图像。要操作其他模式的话，你必须先进行转化。

```
from PIL import Image,ImageQt
im=Image.open('a.jpg')
qim=ImageQt(im)
```

17.ImageSequence Module

ImageSequence模块包含一个打包器类，用于允许你迭代图像序列的帧（gif图像）。

```
#从动画中取出帧
from PIL import Image,ImageSequence
im=Image.open("animation.fli")
index=1
for frame in ImageSequence.Iterator(im):
    frame.save("frame%d.png" % index)
    index+=1
```

`PIL.ImageSequence.Iterator(im)`

这个类实现了一个迭代器，用于在图像序列上进行循环。你可以使用[]操作符访问索引。如果你试图访问一个不存在的帧，这个操作符将会引发一个IndexError。

18.ImageStat Module

ImageStat模块用于计算一个图像或者图像一个区域的全局统计值。

`PIL.ImageStat.Stat(image_or_list,mask=None)`

计算给定图像的统计值，返回一个stat实例。如果使用mask参数，只有mask包含的区域将会被包含在统计值中。你也可以传递一个之前计算出的直方图给这个方法。

image：一个PIL的image对象，或者是一个预先计算出的直方图

mask：一个可选的掩码

```
#stat实例可用的方法
stat=ImageStat.Stat(im)
stat.extrema
#计算图像中每个通道的最大和最值
stat.count
#计算图像中每个通道像素数
stat.sum
#计算图像中每个通道的所有的像素和
stat.sum2
#计算图像中每个通道中所有像素的平方和
stat.mean
#计算图像中每个通道的像素值的算数平均值
stat.median
#计算图像中每个通道像素值的中值
stat.rms
#计算图像中每个通道的均方根
stat.var
#计算图像每个通道的方差
stat.stddev
#计算图像中每个通道的标准差
```

19.ImageTk Module

ImageTk模块包含了对从PIL图像创建和修改Tkinter的BitmapImage和PhotoImage对象的支持。

20.ImageWin Module (Windows-only)

ImageWin模块包含了对在Windows系统上创建和显示图像的支持。ImageWin可以和PythonWin或者其他用户接口工具包一起使用，来提供获得Windows设备上下文或者Windows句柄的入口。

21.ExifTags Module

ExifTgas模块揭示类两个字典，这两个字典提供了各种众所周知的EXIF标签常数和明文名称。

22.TiffTags Module

TiffTags模块揭示了许多标准TIFF元数据标签数字、名称和类型信息。

23.PSDraw Module

PSDraw模块提供了简单的对Postscript打印机的支持，你可以通过这个模块打印文字、图形和图像。

24.PixelAccess Class

PixelAccess类提供了在像素层面读写PIL.Image数据的接口。

25.PyAccess Module

PyAccess模块提供了一个PixelAccess类的CFFI/Python实现。这个实现远快于PixelAccess的PyPy版本。

注意，提取单个像素相当慢，如果你想循环所有的像素，建议你使用Pillow API中的其他部分。

```
#下面的脚本加载了一个图像，从中提取类一个像素并改变它
from PIL import Image
im=Image.open('hopper.jpg')
px=im.load()
print(px[4,4])
px[4,4]=(0,0,0)
print(px[4,4])
```

结果是：

```
(23,24,68)
(0,0,0)
```

26.PIL Package (autodoc of remaining modules)”

文档尚未被移植或编写的模块可以在这里找到。

网络地址PIL Package

27.Plugin reference

用于编写插件的参考。

网络地址Plugin reference