# Golang 图片处理 — image 库

image 库支持常见的 PNG、JPEG、GIF 等格式的图片处理， 可以对图片进行读取、裁剪、绘制、生成等操作。

## 读取、新建图片

### 读取

图片的读取，和文件的读取类似，只需要使用 os.Open()函数，获取一个输入流，然后将数据流进行解码操作。

需要注意的是，在解码阶段，要将不同类型的图片的解码器先进行注册，这样才不会报unknown format 的错误。

package main  
  
import (  
 "fmt"  
 "image"  
 \_ "image/png"  
 "os"  
)  
  
func main() {  
 f, err := os.Open("./gopher.png")  
 if err != nil {  
 panic(err)  
 }  
 img, formatName, err := image.Decode(f)  
 if err != nil {  
 panic(err)  
 }  
 fmt.Println(formatName)  
 fmt.Println(img.Bounds())  
 fmt.Println(img.ColorModel())  
}

Decode 方法返回的第一个值是一个 image.Image类型接口。不同的颜色模型的图片返回不同类型的值。该接口有三个方法：

// Image is a finite rectangular grid of color.Color values taken from a color  
// model.  
type Image interface {  
 // ColorModel returns the Image's color model.  
 ColorModel() color.Model  
 // Bounds returns the domain for which At can return non-zero color.  
 // The bounds do not necessarily contain the point (0, 0).  
 Bounds() Rectangle  
 // At returns the color of the pixel at (x, y).  
 // At(Bounds().Min.X, Bounds().Min.Y) returns the upper-left pixel of the grid.  
 // At(Bounds().Max.X-1, Bounds().Max.Y-1) returns the lower-right one.  
 At(x, y int) color.Color  
}

image 库中很多结构都实现了该接口，对于一些标准库中没有实现的功能，我们也可以自己实现该接口去满足。

### 新建

如果是需要新建一个图片，可以使用image.NewRGBA()方法。

img := image.NewRGBA(image.Rect(0, 0, 300, 300))

这里的 NewRGBA方法返回的是一个实现了 image.Image接口的 image.RGBA类型数据。 这里是一个300\*300的透明背景的图片。

## 保存图片

保存图片和保存文件也类似，需要先将图片编码，然后以数据流的形式写入文件。

package main  
  
import (  
 "bufio"  
 "image"  
 "image/png"  
 \_ "image/png"  
 "os"  
)  
  
func main() {  
 img := image.NewRGBA(image.Rect(0, 0, 300, 300))  
 outFile, err := os.Create("gopher2.png")  
 defer outFile.Close()  
   
 if err != nil {  
 panic(err)  
 }  
   
 b := bufio.NewWriter(outFile)  
 err = png.Encode(b, img)  
 if err != nil {  
 panic(err)  
 }  
   
 err = b.Flush()  
 if err != nil {  
 panic(err)  
 }  
}

## 裁剪图片

图片的裁剪主要使用SubImage()方法，如下：

img := image.NewRGBA(image.Rect(0, 0, 300, 300))  
subImage := img.SubImage(image.Rect(0, 0, 20, 20))

该方法将从创建的300 \* 300的图片裁剪出20 \* 20 像素的子图片。

## 绘制图片

绘制图片主要使用到 draw.Draw和draw.DrawMask方法。

// Draw calls DrawMask with a nil mask.  
func Draw(dst Image, r image.Rectangle, src image.Image, sp image.Point, op Op) {  
 DrawMask(dst, r, src, sp, nil, image.Point{}, op)  
}  
  
// DrawMask aligns r.Min in dst with sp in src and mp in mask and then replaces the rectangle r  
// in dst with the result of a Porter-Duff composition. A nil mask is treated as opaque.  
func DrawMask(dst Image, r image.Rectangle, src image.Image, sp image.Point, mask image.Image, mp image.Point, op Op) {  
 clip(dst, &r, src, &sp, mask, &mp)  
 if r.Empty() {  
 return  
 }  
  
 // Fast paths for special cases. If none of them apply, then we fall back  
 // to general but slower implementations.  
 switch dst0 := dst.(type) {  
 case \*image.RGBA:  
 if op == *Over* {  
 if mask == nil {  
 switch src0 := src.(type) {  
 case \*image.Uniform:  
 sr, sg, sb, sa := src0.RGBA()  
 if sa == 0xffff {  
 drawFillSrc(dst0, r, sr, sg, sb, sa)  
 } else {  
 drawFillOver(dst0, r, sr, sg, sb, sa)  
 }  
 return  
 case \*image.RGBA:  
 drawCopyOver(dst0, r, src0, sp)  
 return  
 case \*image.NRGBA:  
 drawNRGBAOver(dst0, r, src0, sp)  
 return  
 case \*image.YCbCr:  
 // An image.YCbCr is always fully opaque, and so if the  
 // mask is nil (i.e. fully opaque) then the op is  
 // effectively always Src. Similarly for image.Gray and  
 // image.CMYK.  
 if imageutil.DrawYCbCr(dst0, r, src0, sp) {  
 return  
 }  
 case \*image.Gray:  
 drawGray(dst0, r, src0, sp)  
 return  
 case \*image.CMYK:  
 drawCMYK(dst0, r, src0, sp)  
 return  
 }  
 } else if mask0, ok := mask.(\*image.Alpha); ok {  
 switch src0 := src.(type) {  
 case \*image.Uniform:  
 drawGlyphOver(dst0, r, src0, mask0, mp)  
 return  
 case \*image.RGBA:  
 drawRGBAMaskOver(dst0, r, src0, sp, mask0, mp)  
 return  
 case \*image.Gray:  
 drawGrayMaskOver(dst0, r, src0, sp, mask0, mp)  
 return  
 // Case order matters. The next case (image.RGBA64Image) is an  
 // interface type that the concrete types above also implement.  
 case image.RGBA64Image:  
 drawRGBA64ImageMaskOver(dst0, r, src0, sp, mask0, mp)  
 return  
 }  
 }  
 } else {  
 if mask == nil {  
 switch src0 := src.(type) {  
 case \*image.Uniform:  
 sr, sg, sb, sa := src0.RGBA()  
 drawFillSrc(dst0, r, sr, sg, sb, sa)  
 return  
 case \*image.RGBA:  
 drawCopySrc(dst0, r, src0, sp)  
 return  
 case \*image.NRGBA:  
 drawNRGBASrc(dst0, r, src0, sp)  
 return  
 case \*image.YCbCr:  
 if imageutil.DrawYCbCr(dst0, r, src0, sp) {  
 return  
 }  
 case \*image.Gray:  
 drawGray(dst0, r, src0, sp)  
 return  
 case \*image.CMYK:  
 drawCMYK(dst0, r, src0, sp)  
 return  
 }  
 }  
 }  
 drawRGBA(dst0, r, src, sp, mask, mp, op)  
 return  
 case \*image.Paletted:  
 if op == *Src* && mask == nil {  
 if src0, ok := src.(\*image.Uniform); ok {  
 colorIndex := uint8(dst0.Palette.Index(src0.C))  
 i0 := dst0.PixOffset(r.Min.X, r.Min.Y)  
 i1 := i0 + r.Dx()  
 for i := i0; i < i1; i++ {  
 dst0.Pix[i] = colorIndex  
 }  
 firstRow := dst0.Pix[i0:i1]  
 for y := r.Min.Y + 1; y < r.Max.Y; y++ {  
 i0 += dst0.Stride  
 i1 += dst0.Stride  
 copy(dst0.Pix[i0:i1], firstRow)  
 }  
 return  
 } else if !processBackward(dst, r, src, sp) {  
 drawPaletted(dst0, r, src, sp, false)  
 return  
 }  
 }  
 }  
  
 x0, x1, dx := r.Min.X, r.Max.X, 1  
 y0, y1, dy := r.Min.Y, r.Max.Y, 1  
 if processBackward(dst, r, src, sp) {  
 x0, x1, dx = x1-1, x0-1, -1  
 y0, y1, dy = y1-1, y0-1, -1  
 }  
  
 // FALLBACK1.17  
 //  
 // Try the draw.RGBA64Image and image.RGBA64Image interfaces, part of the  
 // standard library since Go 1.17. These are like the draw.Image and  
 // image.Image interfaces but they can avoid allocations from converting  
 // concrete color types to the color.Color interface type.  
  
 if dst0, \_ := dst.(RGBA64Image); dst0 != nil {  
 if src0, \_ := src.(image.RGBA64Image); src0 != nil {  
 if mask == nil {  
 sy := sp.Y + y0 - r.Min.Y  
 my := mp.Y + y0 - r.Min.Y  
 for y := y0; y != y1; y, sy, my = y+dy, sy+dy, my+dy {  
 sx := sp.X + x0 - r.Min.X  
 mx := mp.X + x0 - r.Min.X  
 for x := x0; x != x1; x, sx, mx = x+dx, sx+dx, mx+dx {  
 if op == *Src* {  
 dst0.SetRGBA64(x, y, src0.RGBA64At(sx, sy))  
 } else {  
 srgba := src0.RGBA64At(sx, sy)  
 a := *m* - uint32(srgba.A)  
 drgba := dst0.RGBA64At(x, y)  
 dst0.SetRGBA64(x, y, color.RGBA64{  
 R: uint16((uint32(drgba.R)\*a)/*m*) + srgba.R,  
 G: uint16((uint32(drgba.G)\*a)/*m*) + srgba.G,  
 B: uint16((uint32(drgba.B)\*a)/*m*) + srgba.B,  
 A: uint16((uint32(drgba.A)\*a)/*m*) + srgba.A,  
 })  
 }  
 }  
 }  
 return  
  
 } else if mask0, \_ := mask.(image.RGBA64Image); mask0 != nil {  
 sy := sp.Y + y0 - r.Min.Y  
 my := mp.Y + y0 - r.Min.Y  
 for y := y0; y != y1; y, sy, my = y+dy, sy+dy, my+dy {  
 sx := sp.X + x0 - r.Min.X  
 mx := mp.X + x0 - r.Min.X  
 for x := x0; x != x1; x, sx, mx = x+dx, sx+dx, mx+dx {  
 ma := uint32(mask0.RGBA64At(mx, my).A)  
 switch {  
 case ma == 0:  
 if op == *Over* {  
 // No-op.  
 } else {  
 dst0.SetRGBA64(x, y, color.RGBA64{})  
 }  
 case ma == *m* && op == *Src*:  
 dst0.SetRGBA64(x, y, src0.RGBA64At(sx, sy))  
 default:  
 srgba := src0.RGBA64At(sx, sy)  
 if op == *Over* {  
 drgba := dst0.RGBA64At(x, y)  
 a := *m* - (uint32(srgba.A) \* ma / *m*)  
 dst0.SetRGBA64(x, y, color.RGBA64{  
 R: uint16((uint32(drgba.R)\*a + uint32(srgba.R)\*ma) / *m*),  
 G: uint16((uint32(drgba.G)\*a + uint32(srgba.G)\*ma) / *m*),  
 B: uint16((uint32(drgba.B)\*a + uint32(srgba.B)\*ma) / *m*),  
 A: uint16((uint32(drgba.A)\*a + uint32(srgba.A)\*ma) / *m*),  
 })  
 } else {  
 dst0.SetRGBA64(x, y, color.RGBA64{  
 R: uint16(uint32(srgba.R) \* ma / *m*),  
 G: uint16(uint32(srgba.G) \* ma / *m*),  
 B: uint16(uint32(srgba.B) \* ma / *m*),  
 A: uint16(uint32(srgba.A) \* ma / *m*),  
 })  
 }  
 }  
 }  
 }  
 return  
 }  
 }  
 }  
  
 // FALLBACK1.0  
 //  
 // If none of the faster code paths above apply, use the draw.Image and  
 // image.Image interfaces, part of the standard library since Go 1.0.  
  
 var out color.RGBA64  
 sy := sp.Y + y0 - r.Min.Y  
 my := mp.Y + y0 - r.Min.Y  
 for y := y0; y != y1; y, sy, my = y+dy, sy+dy, my+dy {  
 sx := sp.X + x0 - r.Min.X  
 mx := mp.X + x0 - r.Min.X  
 for x := x0; x != x1; x, sx, mx = x+dx, sx+dx, mx+dx {  
 ma := uint32(*m*)  
 if mask != nil {  
 \_, \_, \_, ma = mask.At(mx, my).RGBA()  
 }  
 switch {  
 case ma == 0:  
 if op == *Over* {  
 // No-op.  
 } else {  
 dst.Set(x, y, color.Transparent)  
 }  
 case ma == *m* && op == *Src*:  
 dst.Set(x, y, src.At(sx, sy))  
 default:  
 sr, sg, sb, sa := src.At(sx, sy).RGBA()  
 if op == *Over* {  
 dr, dg, db, da := dst.At(x, y).RGBA()  
 a := *m* - (sa \* ma / *m*)  
 out.R = uint16((dr\*a + sr\*ma) / *m*)  
 out.G = uint16((dg\*a + sg\*ma) / *m*)  
 out.B = uint16((db\*a + sb\*ma) / *m*)  
 out.A = uint16((da\*a + sa\*ma) / *m*)  
 } else {  
 out.R = uint16(sr \* ma / *m*)  
 out.G = uint16(sg \* ma / *m*)  
 out.B = uint16(sb \* ma / *m*)  
 out.A = uint16(sa \* ma / *m*)  
 }  
 // The third argument is &out instead of out (and out is  
 // declared outside of the inner loop) to avoid the implicit  
 // conversion to color.Color here allocating memory in the  
 // inner loop if sizeof(color.RGBA64) > sizeof(uintptr).  
 dst.Set(x, y, &out)  
 }  
 }  
 }  
}

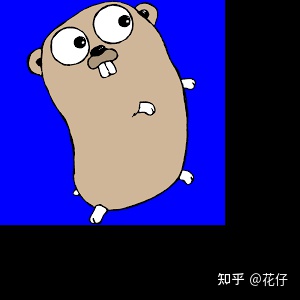
### Draw

Draw方法各个参数含义如下：

* dst 绘图的背景图
* r 背景图的绘图区域
* src 要绘制的图
* sp src 对应的绘图开始点
* op 组合方式

以下代码是将一个 Gopher 的图案绘制到了一张黑色背景空白图的左上角。

f, err := os.Open("./gopher.png")  
if err != nil {  
 panic(err)  
}  
gopherImg, \_, err := image.Decode(f) // 打开图片  
  
img := image.NewRGBA(image.Rect(0, 0, 300, 300))  
for x := 0; x < img.Bounds().Dx(); x++ { // 将背景图涂黑  
 for y := 0; y < img.Bounds().Dy(); y++ {  
 img.Set(x, y, color.Black)  
 }  
}  
draw.Draw(img, img.Bounds(), gopherImg, image.Pt(0, 0), draw.*Over*) // 将gopherImg绘制到背景图上

****

### DrawMask

DrawMask方法多了一个遮罩蒙层参数mask，以及蒙层的起始位置参数 mp。

Draw方法是 DrawMask的一种特殊形式，当 DrawMask 的 mask 参数为nil时，即为Draw方法。

DrawMask将背景图上的绘图区域起始点、要绘制图的起始点、遮罩蒙层的起始点进行对齐，然后对背景图上的绘图矩阵区域执行 [Porter-Duff](https://link.zhihu.com/?target=https://docs.microsoft.com/zh-tw/xamarin/xamarin-forms/user-interface/graphics/skiasharp/effects/blend-modes/porter-duff" \t "https://zhuanlan.zhihu.com/p/_blank)合并操作。

下面是给图片加一个圆形遮罩的示例：

func drawCirclePic() {  
 f, err := os.Open("./gopher.png")  
 if err != nil {  
 panic(err)  
 }  
 gopherImg, \_, err := image.Decode(f)  
  
 d := gopherImg.Bounds().Dx()  
  
 //将一个cicle作为蒙层遮罩，圆心为图案中点，半径为边长的一半  
 c := circle{p: image.Point{X: d / 2, Y: d / 2}, r: d / 2}  
 circleImg := image.NewRGBA(image.Rect(0, 0, d, d))  
 draw.DrawMask(circleImg, circleImg.Bounds(), gopherImg, image.Point{}, &c, image.Point{}, draw.*Over*)  
  
 SavePng(circleImg)  
}  
  
type circle struct { // 这里需要自己实现一个圆形遮罩，实现接口里的三个方法  
 p image.Point // 圆心位置  
 r int  
}  
  
func (c \*circle) ColorModel() color.Model {  
 return color.AlphaModel  
}  
  
func (c \*circle) Bounds() image.Rectangle {  
 return image.Rect(c.p.X-c.r, c.p.Y-c.r, c.p.X+c.r, c.p.Y+c.r)  
}  
  
// 对每个像素点进行色值设置，在半径以内的图案设成完全不透明  
func (c \*circle) At(x, y int) color.Color {  
 xx, yy, rr := float64(x-c.p.X)+0.5, float64(y-c.p.Y)+0.5, float64(c.r)  
 if xx\*xx+yy\*yy < rr\*rr {  
 return color.Alpha{A: 255}  
 }  
 return color.Alpha{}  
}

****

给图片加一个圆角遮罩的示例：

func drawRadiusPic() {  
 f, err := os.Open("./gopher.png")  
 if err != nil {  
 panic(err)  
 }  
 gopherImg, \_, err := image.Decode(f)  
   
 w := gopherImg.Bounds().Dx()  
 h := gopherImg.Bounds().Dy()  
   
 c := radius{p: image.Point{X: w, Y: h}, r: int(40)}  
 radiusImg := image.NewRGBA(image.Rect(0, 0, w, h))  
 draw.DrawMask(radiusImg, radiusImg.Bounds(), gopherImg, image.Point{}, &c, image.Point{}, draw.*Over*)  
   
 SavePng(radiusImg)  
}  
  
type radius struct {  
 p image.Point // 矩形右下角位置  
 r int  
}  
  
func (c \*radius) ColorModel() color.Model {  
 return color.AlphaModel  
}  
  
func (c \*radius) Bounds() image.Rectangle {  
 return image.Rect(0, 0, c.p.X, c.p.Y)  
}  
  
// 对每个像素点进行色值设置，分别处理矩形的四个角，在四个角的内切圆的外侧，色值设置为全透明，其他区域不透明  
func (c \*radius) At(x, y int) color.Color {  
 var xx, yy, rr float64  
 var inArea bool  
 // left up  
 if x <= c.r && y <= c.r {  
 xx, yy, rr = float64(c.r-x)+0.5, float64(y-c.r)+0.5, float64(c.r)  
 inArea = true  
 }  
 // right up  
 if x >= (c.p.X-c.r) && y <= c.r {  
 xx, yy, rr = float64(x-(c.p.X-c.r))+0.5, float64(y-c.r)+0.5, float64(c.r)  
 inArea = true  
 }  
 // left bottom  
 if x <= c.r && y >= (c.p.Y-c.r) {  
 xx, yy, rr = float64(c.r-x)+0.5, float64(y-(c.p.Y-c.r))+0.5, float64(c.r)  
 inArea = true  
 }  
 // right bottom  
 if x >= (c.p.X-c.r) && y >= (c.p.Y-c.r) {  
 xx, yy, rr = float64(x-(c.p.X-c.r))+0.5, float64(y-(c.p.Y-c.r))+0.5, float64(c.r)  
 inArea = true  
 }  
   
 if inArea && xx\*xx+yy\*yy >= rr\*rr {  
 return color.Alpha{}  
 }  
 return color.Alpha{A: 255}  
}

****

**在图案进行圆形、圆角绘制的过程中，因为最小单位是1px，所以可能会有锯齿边缘的问题，解决这个问题可以通过先将原图放大，遮罩后再缩小来解决。**

# Reference

* **[The Go image/draw package - The Go Blog (golang.org)](https://link.zhihu.com/?target=https://blog.golang.org/image-draw" \t "https://zhuanlan.zhihu.com/p/_blank)**
* **[Porter-Duff blend 模式 - Xamarin | Microsoft Docs](https://link.zhihu.com/?target=https://docs.microsoft.com/zh-tw/xamarin/xamarin-forms/user-interface/graphics/skiasharp/effects/blend-modes/porter-duff" \t "https://zhuanlan.zhihu.com/p/_blank)**