Go语言 标准库 image包

go中处理图片的标准库image支持常见的PNG、JPEG、GIF等格式的图片处理（可读取、裁剪、绘制、生成等）。

image接口

type Image

image是一个从颜色模型中采取color.Color的矩形网格

```
type Image interface {
    ColorModel() color.Model  //ColorModel  返回图片的 color.Model
    Bounds() Rectangle       //图片中非 0 color的区域
    At(x, y int) color.Color    //返回指定点的像素color.Color
}
```

任何一个struct只要实现了image中的三个方法，便实现了image接口。

Decode

Decode对一个根据指定格式进行编码的图片进行解码操作，string返回的是在注册过程中使用的格式化名字，如"gif"或者"jpeg"等.

```
func Decode(r io.Reader) (Image, string, error)
```

RegisterFormat

RegisterFormat注册一个image格式给解码使用，name是格式化名字，例如 " jpeg"或者 " png " ,magic标明格式化编码的前缀，magic字符串中能够包含一个?字符，用来匹配任何一个字符，decode是用来解码 " 编码图像"的函数，DecodeConfig是一个仅仅解码它的配置的函数

```
func RegisterFormat(name, magic string, decode func(io.Reader) (Image, error),
decodeConfig func(io.Reader) (Config, error))
```

type Alpha

用来设置图片的透明度

```
type Alpha struct {
    Pix    []uint8   // Pix 存储图片的像素 ，像 alpha 值. 在（X,Y）的像素 starts at
Pix[(y-Rect.Min.Y)*Stride + (x-Rect.Min.X)*1].
    Stride int       // Stride 表示垂直两个像素之间的步幅（距离）
    Rect   Rectangle // Rect 表示 图片边界.
}
```

NewAlpha

利用给定矩形边界产生一个alpha，Alpha类型代表一个8位的alpha通道（alpha通道表示透明度）。

```
func NewAlpha(r Rectangle) *Alpha        //利用给定矩形边界产生一个alpha
```

方法	说明
func (p *Alpha) AlphaAt(x, y int) color.Alpha	获取指定点的透明度
func (p *Alpha) At(x, y int) color.Color	获取指定点的color(指定点的红绿蓝的透明度)
func (p *Alpha) Bounds() Rectangle	获取alpha的边界
func (p *Alpha) ColorModel() color.Model	获取alpha的颜色模型
func (p *Alpha) Opaque() bool	检查alpha是否完全不透明
func (p *Alpha) PixOffset(x, y int) int	获取指定像素相对于第一个像素的相对偏移量
func (p *Alpha) Set(x, y int, c color.Color)	设定指定位置的color
func (p *Alpha) SetAlpha(x, y int, c color.Alpha)	设定指定位置的alpha
func (p *Alpha) SubImage(r Rectangle) Image	获取p图像中被r覆盖的子图像，父图像和子图像公用像素

示例：

```
package main

import (
    "fmt"
    "image"
    "image/color"
    "image/jpeg"
    "log"
    "os"
)

const (
    dx = 500
    dy = 200
)

func main() {

    file, err := os.Create("test.jpeg")
    if err != nil {
        log.Fatal(err)
    }
    defer file.Close()
    alpha := image.NewAlpha(image.Rect(0, 0, dx, dy))
```

```
    for x := 0; x < dx; x++ {
        for y := 0; y < dy; y++ {
            alpha.Set(x, y, color.Alpha{uint8(x % 256)}) //设定alpha图片的透明度
        }
    }

    fmt.Println(alpha.At(400, 100))    //144 在指定位置的像素
    fmt.Println(alpha.Bounds())        //(0,0)-(500,200)，图片边界
    fmt.Println(alpha.Opaque())        //false，是否图片完全透明
    fmt.Println(alpha.PixOffset(1, 1)) //501，指定点相对于第一个点的距离
    fmt.Println(alpha.Stride)          //500，两个垂直像素之间的距离
    jpeg.Encode(file, alpha, nil)      //将image信息写入文件中
}
```

type Alpha16

Alpha16类型代表一个16位的alpha通道。

```
type Alpha16 struct {
    // Pix holds the image's pixels, as alpha values in big-endian format. The
pixel at
    // (x, y) starts at Pix[(y-Rect.Min.Y)*Stride + (x-Rect.Min.X)*2]. 大端模式，
所以像素计算和alpha不同
    Pix []uint8
    // Stride is the Pix stride (in bytes) between vertically adjacent pixels.
    Stride int
    // Rect is the image's bounds.
    Rect Rectangle
}
```

其中alpha16中方法用法与alpha中方法用法类似，这里不再赘述.

NewAlpha16

利用给定矩形边界产生一个alpha，Alpha类型代表一个8位的alpha通道（alpha通道表示透明度）。

```
func NewAlpha16(r Rectangle) *Alpha16        //利用给定矩形边界产生一个alpha16
```

方法	说明
func (p *Alpha16) AlphaAt(x, y int) color.Alpha16	获取指定点的透明度
func (p *Alpha16) At(x, y int) color.Color	获取指定点的color(指定点的红绿蓝的透明度)
func (p *Alpha16) Bounds() Rectangle	获取alpha的边界
func (p *Alpha16) ColorModel() color.Model	获取alpha的颜色模型
func (p *Alpha16) Opaque() bool	检查alpha是否完全不透明
func (p *Alpha16) PixOffset(x, y int) int	获取指定像素相对于第一个像素的相对偏移量
func (p *Alpha16) Set(x, y int, c color.Color)	设定指定位置的color

方法 (p *Alpha16) SetAlpha(x, y int, c color.Alpha16)	说明 设定指定位置的alpha16
func (p *Alpha16) SubImage(r Rectangle) Image	获取p图像中被r覆盖的子图像，父图像和子图像公用像素

type Config

包括图像的颜色模型和宽高尺寸

```
type Config struct {
    ColorModel    color.Model
    Width, Height int
}
```

DecodeConfig

获取宽高及颜色模型

```
func DecodeConfig(r io.Reader) (Config, string, error)
```

type Gray

Gray类型代表一个8位的灰度色彩。用来设置图片的灰度。

NewGray

```
func NewGray(r Rectangle) *Gray
```

其中Gray方法与Alpha完全相同，不在赘述。

```
func (p *Gray) At(x, y int) color.Color
func (p *Gray) Bounds() Rectangle
func (p *Gray) ColorModel() color.Model
func (p *Gray) GrayAt(x, y int) color.Gray
func (p *Gray) Opaque() bool
func (p *Gray) PixOffset(x, y int) int
func (p *Gray) Set(x, y int, c color.Color)
func (p *Gray) SetGray(x, y int, c color.Gray)
func (p *Gray) SubImage(r Rectangle) Image
```

type Gray16

```
type Gray16 struct {
    // Pix holds the image's pixels, as gray values in big-endian format. The
pixel at
    // (x, y) starts at Pix[(y-Rect.Min.Y)*Stride + (x-Rect.Min.X)*2].
    Pix []uint8
    // Stride is the Pix stride (in bytes) between vertically adjacent pixels.
    Stride int
    // Rect is the image's bounds.
    Rect Rectangle
}
```

NewGray16

```
func NewGray16(r Rectangle) *Gray16
```

Gray16类型代表一个16位的灰度色彩。用来设置图片的灰度。

Gray16方法与Alpha16方法完全相同，不再赘述。

```
func (p *Gray16) At(x, y int) color.Color
func (p *Gray16) Bounds() Rectangle
func (p *Gray16) ColorModel() color.Model
func (p *Gray16) Gray16At(x, y int) color.Gray16
func (p *Gray16) Opaque() bool
func (p *Gray16) PixOffset(x, y int) int
func (p *Gray16) Set(x, y int, c color.Color)
func (p *Gray16) SetGray16(x, y int, c color.Gray16)
func (p *Gray16) SubImage(r Rectangle) Image
```

type NRGBA

NRGBA类型代表没有预乘alpha通道的32位RGB色彩，Red、Green、Blue、Alpha各8位。

```
type NRGBA struct {
    // Pix holds the image's pixels, in R, G, B, A order. The pixel at
    // (x, y) starts at <span style="color:#FF0000;">Pix[(y-Rect.Min.Y)*Stride +
(x-Rect.Min.X)*4]</span>.
    Pix []uint8
    // Stride is the Pix stride (in bytes) between vertically adjacent pixels.
    Stride int
    // Rect is the image's bounds.
    Rect Rectangle
}
```

NewNRGBA

```
func NewNRGBA(r Rectangle) *NRGBA
```

type NRGBA64

NRGBA64类型代表没有预乘alpha通道的64位RGB色彩，Red、Green、Blue、Alpha各16位。

```
type NRGBA64 struct {
    // Pix holds the image's pixels, in R, G, B, A order and big-endian format.
The pixel at
    // (x, y) starts at Pix[(y-Rect.Min.Y)*Stride + (x-Rect.Min.X)*8].
    Pix []uint8
    // Stride is the Pix stride (in bytes) between vertically adjacent pixels.
    Stride int
    // Rect is the image's bounds.
    Rect Rectangle
}
```

NewNRGBA64

```
func NewNRGBA64(r Rectangle) *NRGBA64
```

type Paletted

Palette类型代表一个色彩的调色板。

```
type Paletted struct {
    // Pix holds the image's pixels, as palette indices. The pixel at
    // (x, y) starts at Pix[(y-Rect.Min.Y)*Stride + (x-Rect.Min.X)*1].
    Pix []uint8
    // Stride is the Pix stride (in bytes) between vertically adjacent pixels.
    Stride int
    // Rect is the image's bounds.
    Rect Rectangle
    // Palette is the image's palette.
    Palette color.Palette
}
```

NewPaletted

根据指定的宽高和颜色调色板生成一个新的调色板

```
func NewPaletted(r Rectangle, p color.Palette) *Paletted
```

type PalettedImage

调色板图像接口

```
type PalettedImage interface {
    ColorIndexAt(x, y int) uint8  //返回在位置(x,y)处像素的索引
    Image    //image接口
}
```

type Point

一个点的(x,y)坐标对

```
type Point struct {
    X, Y int
}
```

Pt

Pt是Point{X, Y}的简写

```
func Pt(X, Y int) Point      //Pt是Point{X, Y}的简写
```

方法	说明
func (p Point) Add(q Point) Point	两个向量点求和
func (p Point) Div(k int) Point	Div returns the vector p/k，求Point / k的值
func (p Point) Eq(q Point) bool	判定两个向量点是否相等
func (p Point) In(r Rectangle) bool	判断某个向量点是否在矩阵中
func (p Point) Mod(r Rectangle) Point	在矩阵r中求一个点q，是的p.x-q.x是矩阵宽的倍数，p.y-q.y是矩阵高的倍数
func (p Point) Mul(k int) Point	返回向量点和指定值的乘积组成的向量点
func (p Point) String() string	返回用string表示的向量点，其样式如(1,2)
func (p Point) Sub(q Point) Point	两个向量点求差

示例：

```
func main() {
    pt := image.Point{X: 5, Y: 5}
    fmt.Println(pt)            //(5,5) ，输出一个点位置（X,Y）
    fmt.Println(image.Pt(1, 2))   //(1,2)  ，Pt输出一个点位置的简写形式
    fmt.Println(pt.Add(image.Pt(1, 1)))  //(6,6)，两个点求和
    fmt.Println(pt.String())      //(5,5) ，以字符串形式输出点
    fmt.Println(pt.Eq(image.Pt(5, 5)))        //true，判断两个点是否完全相等
    fmt.Println(pt.In(image.Rect(0, 0, 10, 10)))    //true，判断一个点是否在矩阵中
    fmt.Println(pt.Div(2))          //(2,2)，求点的商
    fmt.Println(pt.Mul(2))             // (10,10)，求点的乘积
    fmt.Println(pt.Sub(image.Pt(1, 1)))     // (4,4)，求两个点的差
fmt.Println(pt.Mod(image.Rect(9, 8, 10, 10)))  // (9,9)，dx=10-9=1,dy=10-8=2,9-5=4,4是1和2的倍数并且（9,9）在矩阵中
}
```

type RGBA

RGBA类型代表传统的预乘了alpha通道的32位RGB色彩，Red、Green、Blue、Alpha各8位。

```
type RGBA struct {
    // Pix holds the image's pixels, in R, G, B, A order. The pixel at
    // (x, y) starts at Pix[(y-Rect.Min.Y)*Stride + (x-Rect.Min.X)*4]
    Pix []uint8
    // Stride is the Pix stride (in bytes) between vertically adjacent pixels.
    Stride int
    // Rect is the image's bounds.
    Rect Rectangle
}
```

NewRGBA

返回具有给定边界的新RGBA图像。

```
func NewRGBA(r Rectangle) *RGBA
```

```
func (p *RGBA) At(x, y int) color.Color
func (p *RGBA) Bounds() Rectangle
func (p *RGBA) ColorModel() color.Model
func (p *RGBA) Opaque() bool
func (p *RGBA) PixOffset(x, y int) int
func (p *RGBA) Set(x, y int, c color.Color)
func (p *RGBA) SetRGBA(x, y int, c color.RGBA)
func (p *RGBA) SubImage(r Rectangle) Image
```

示例：

```
package main

import (
    "fmt"
    "image"
    "image/color"
    "image/jpeg"
    "log"
    "os"
)

const (
    dx = 500
    dy = 200
)

func main() {

    file, err := os.Create("test.jpg")
    if err != nil {
        log.Fatal(err)
    }
```

```
    defer file.Close()
    rgba := image.NewRGBA(image.Rect(0, 0, dx, dy))
    for x := 0; x < dx; x++ {
        for y := 0; y < dy; y++ {
            rgba.Set(x, y, color.NRGBA{uint8(x % 256), uint8(y % 256), 0, 255})
        }
    }

    fmt.Println(rgba.At(400, 100))    //{144 100 0 255}
    fmt.Println(rgba.Bounds())        //(0,0)-(500,200)
    fmt.Println(rgba.Opaque())        //true，其完全透明
    fmt.Println(rgba.PixOffset(1, 1)) //2004
    fmt.Println(rgba.Stride)          //2000
    jpeg.Encode(file, rgba, nil)      //将image信息存入文件中
}
```

type RGBA64

RGBA64类型代表预乘了alpha通道的64位RGB色彩，Red、Green、Blue、Alpha各16位。

```
type RGBA64 struct {
    // Pix holds the image's pixels, in R, G, B, A order and big-endian format.
The pixel at
    // (x, y) starts at Pix[(y-Rect.Min.Y)*Stride + (x-Rect.Min.X)*8].
    Pix []uint8
    // Stride is the Pix stride (in bytes) between vertically adjacent pixels.
    Stride int
    // Rect is the image's bounds.
    Rect Rectangle
}
```

NewRGBA64

```
func NewRGBA64(r Rectangle) *RGBA64
```

```
func (p *RGBA64) At(x, y int) color.Color
func (p *RGBA64) Bounds() Rectangle
func (p *RGBA64) ColorModel() color.Model
func (p *RGBA64) Opaque() bool
func (p *RGBA64) PixOffset(x, y int) int
func (p *RGBA64) RGBA64At(x, y int) color.RGBA64
func (p *RGBA64) Set(x, y int, c color.Color)
func (p *RGBA64) SetRGBA64(x, y int, c color.RGBA64)
func (p *RGBA64) SubImage(r Rectangle) Image
```

type Rectangle

利用两个坐标点来生成矩阵

```
type Rectangle struct {
    Min, Max Point
}
```

Rect

Rect是Rectangle{Pt(x0, y0), Pt(x1, y1)}的一种简写形式

```
func Rect(x0, y0, x1, y1 int) Rectangle
```

方法	说明
func (r Rectangle) Add(p Point) Rectangle	矩阵中两个点都与指定的点求和组成一个新的矩阵

方法	说明
func (r Rectangle) Canon() Rectangle	返回标准格式的矩形，如果有必要的话，会进行最小值坐标和最大坐标的交换
func (r Rectangle) Dx() int	返回矩阵宽度dx
func (r Rectangle) Dy() int	返回矩阵高度dy
func (r Rectangle) Empty() bool	判定是否该矩阵为空，即不包含任何point
func (r Rectangle) Eq(s Rectangle) bool	判断两个矩阵是否相等，指的是完全重合
func (r Rectangle) In(s Rectangle) bool	判断一个矩阵是否在另外一个矩阵之内
func (r Rectangle) Inset(n int) Rectangle	返回根据n算出的嵌入的矩阵，计算方法是矩阵的每个坐标都减去n，求得的矩阵必须在已知矩阵内嵌，如果没有的话则返回空矩阵
func (r Rectangle) Intersect(s Rectangle) Rectangle	求两个矩阵的相交矩阵，如果两个矩阵不相交，则返回 0 矩阵
func (r Rectangle) Overlaps(s Rectangle) bool	判断两个矩阵是否有交集，即判断两个矩阵是否有公共区域
func (r Rectangle) Size() Point	返回矩阵的宽和高，即dx和dy
func (r Rectangle) String() string	返回矩阵的字符串表示
func (r Rectangle) Sub(p Point) Rectangle	一个矩阵的两个坐标点同时减去一个指定的坐标点p，得到的一个新的矩阵
func (r Rectangle) Union(s Rectangle) Rectangle	两个矩阵的并集，这个是和Intersect（求两个矩阵的交集）相对的

示例：

```
package main

import (
    "fmt"
    "image"
)

func main() {
    rt := image.Rect(0, 0, 100, 50)
```

```
    rt1 := image.Rect(100, 100, 10, 10)

    fmt.Println(rt1.Canon())      //(10,10)-(100,100)，rt1大小坐标交换位置
    fmt.Println(rt, rt1)          //(0,0)-(100,50) (10,10)-(100,100)
    fmt.Println(rt.Dx(), rt.Dy()) //100 50，返回矩阵的宽度和高度
    fmt.Println(rt.Empty())       //false，矩阵是否为空
    fmt.Println(rt.Eq(rt1))       //false，两个矩阵是否相等
    fmt.Println(rt.In(rt1))       //false，矩阵rt是否在矩阵rt1中
    fmt.Println(rt.Inset(10))     //(10,10)-(90,40)，查找内嵌矩阵，用原矩阵坐标点减去给定的值10得到的矩阵，该矩阵必须是原矩阵的内嵌矩阵

    if rt.Overlaps(rt1) {
        fmt.Println(rt.Intersect(rt1)) //(10,10)-(100,50) //求两个矩阵的交集
    }
    fmt.Println(rt.Size())                //(100,50)，求矩阵大小，其等价与（dx，dy）
    fmt.Println(rt.String())              //  (0,0)-(100,50)
    fmt.Println(rt.Sub(image.Pt(10, 10))) // (-10,-10)-(90,40)，求矩阵和一个点的差，用于将矩阵进行移位操作
    fmt.Println(rt.Union(rt1))            //(0,0)-(100,100)，求两个矩阵的并集
}
```

type Uniform

Uniform是一个具有统一颜色无穷大小的图片，它实现了color.Color, color.Model, 以及 Image的接口

```
type Uniform struct {
    C color.Color
}
```

NewUniform

根据color.Color产生一个Uniform

```
func NewUniform(c color.Color) *Uniform
```

方法	说明
func (c *Uniform) At(x, y int) color.Color	获取指定点的像素信息
func (c *Uniform) Bounds() Rectangle	获取图像的边界矩阵信息
func (c *Uniform) ColorModel() color.Model	获取图像的颜色模型
func (c *Uniform) Convert(color.Color) color.Color	将图片的像素信息转换为另外一种指定的像素信息
func (c *Uniform) Opaque() bool	判定图片是否完全透明
func (c *Uniform) RGBA() (r, g, b, a uint32)	返回图片的r,g,b,a（红，绿，蓝，透明度）的值

type YCbCr

YcbCr代表完全不透明的24位Y'CbCr色彩；每个色彩都有1个亮度成分和2个色度成分，分别用8位字节表示。

JPEG、VP8、MPEG家族和其他编码方式使用本色彩模型。这些编码通常将Y'CbCr 和YUV两个色彩模型等同使用（Y=Y'=黄、U=Cb=青、V=Cr=品红）。但严格来说，YUV模只用于模拟视频信号，Y'是经过伽玛校正的Y。RGB和Y'CbCr色彩模型之间的转换会丢失色彩信息。两个色彩模型之间的转换有多个存在细微区别的算法。本包采用JFIF算法。

YCbCr是一个Y'CbCr颜色的图片，每个Y样本表示一个像素，但是每个Cb和Cr能够代表一个或者更多的像素，YStride是在相邻垂直像素的Y slice索引增量，CStride是Cb和 Cr　slice在相邻垂直像素（映射到独立色度采样）的索引增量．通常YStride和len(Y)是 8 的倍数，而CStride结果如下：

```
For 4:4:4, CStride == YStride/1 && len(Cb) == len(Cr) == len(Y)/1.
For 4:2:2, CStride == YStride/2 && len(Cb) == len(Cr) == len(Y)/2.
For 4:2:0, CStride == YStride/2 && len(Cb) == len(Cr) == len(Y)/4.
For 4:4:0, CStride == YStride/1 && len(Cb) == len(Cr) == len(Y)/2.
```

```
type YCbCr struct {
    Y, Cb, Cr      []uint8
    YStride        int
    CStride        int
    SubsampleRatio YCbCrSubsampleRatio
    Rect           Rectangle
}
```

NewYCbCr

通过给定边界和子样本比例创建新的YCbCr

```
func NewYCbCr(r Rectangle, subsampleRatio YCbCrSubsampleRatio) *YCbCr
```

方法	说明
func (p *YCbCr) At(x, y int) color.Color	获取指定点的像素
func (p *YCbCr) Bounds() Rectangle	获取图像边界
func (p *YCbCr) COffset(x, y int) int	获取指定点相对于第一个Cb元素的像素点的相对位置
func (p *YCbCr) ColorModel() color.Model	获取颜色Model
func (p *YCbCr) Opaque() bool	判定是否完全透明
func (p *YCbCr) SubImage(r Rectangle) Image	根据指定矩阵获取原图像的子图像
func (p *YCbCr) YCbCrAt(x, y int) color.YCbCr	
func (p *YCbCr) YOffset(x, y int) int	获取相对于第一个Y元素的像素点的相对位置

type YCbCrSubsampleRatio

YCbCr的色度子样本比例，常用于NewYCbCr(r Rectangle, subsampleRatio YCbCrSubsampleRatio)中用来创建YCbCr。

```
const (
    YCbCrSubsampleRatio444 YCbCrSubsampleRatio = iota
    YCbCrSubsampleRatio422
    YCbCrSubsampleRatio420
    YCbCrSubsampleRatio440
)
```

String

YCbCrSubsampleRatio结构的字符串表示

```
func (s YCbCrSubsampleRatio) String() string
```

基本操作

读取

图片读取和文件读取类似，需要先获取流：

- 注册图片的解码器（如：jpg则`import _ "image/jpeg"`, png则`import _ "image/png"`）
- 通过`os.open`打开文件获取流；
- 通过`image.Decode`解码流，获取图片；

```
package main

import (
    "fmt"
    "image"
    "image/color"
    _ "image/jpeg"
    "os"
)

func readPic() image.Image {
    f, err := os.Open("C:\\Users\\包子\\Pictures\\Default.jpg")
    if err != nil {
        panic(err)
    }
    defer f.Close()

    img, fmtName, err := image.Decode(f)
    if err != nil {
        panic(err)
    }
    fmt.Printf("Name: %v, Bounds: %+v, Color: %+v", fmtName, img.Bounds(),
img.ColorModel())

    return img
```

```
}
```

解码后返回的第一个参数为Image接口：

```
type Image interface {
  ColorModel() color.Model // 返回图片的颜色模型
  Bounds() Rectangle       // 返回图片外框
  At(x, y int) color.Color // 返回(x,y)像素点的颜色
}
```

示例：获取一张图片尺寸

```
package main

import (
    "fmt"
    "image"
    _ "image/jpeg"
    _ "image/png"
    "os"
)

func getImageDimension(imagePath string) (int, int) {
    file, err := os.Open(imagePath)
    defer file.Close()
    if err != nil {
        fmt.Fprintf(os.Stderr, "%v\n", err)
    }

    image, _, err := image.DecodeConfig(file)
    if err != nil {
        fmt.Fprintf(os.Stderr, "%s: %v\n", imagePath, err)
    }
    return image.Width, image.Height
}

func main() {
    width, height := getImageDimension("C:\\Users\\包子\\Pictures\\Default.jpg")
    fmt.Println("Width:", width, "Height:", height)
}
```

新建

新建一个图片非常简单，只需`image.NewRGBA`即可创建一个透明背景的图片了。

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

示例：

```
package main

import "image"
import "image/color"
```

```
import "image/png"
import "os"

func main() {
    // Create an 100 x 50 image
    img := image.NewRGBA(image.Rect(0, 0, 100, 50))

    // Draw a red dot at (2, 3)
    img.Set(2, 3, color.RGBA{255, 0, 0, 255})

    // Save to out.png
    f, _ := os.OpenFile("out.png", os.O_WRONLY|os.O_CREATE, 0600)
    defer f.Close()
    png.Encode(f, img)
}
```

示例：生成一张彩色图片

```
package main

import (
    "fmt"
    "image"
    "image/color"
    "image/png"
    "math"
    "os"
)

type Circle struct {
    X, Y, R float64
}

func (c *Circle) Brightness(x, y float64) uint8 {
    var dx, dy float64 = c.X - x, c.Y - y
    d := math.Sqrt(dx*dx+dy*dy) / c.R
    if d > 1 {
        return 0
    } else {
        return 255
    }
}

func main() {
    var w, h int = 280, 240
    var hw, hh float64 = float64(w / 2), float64(h / 2)
    r := 40.0
    θ := 2 * math.Pi / 3
    cr := &Circle{hw - r*math.Sin(0), hh - r*math.Cos(0), 60}
    cg := &Circle{hw - r*math.Sin(θ), hh - r*math.Cos(θ), 60}
    cb := &Circle{hw - r*math.Sin(-θ), hh - r*math.Cos(-θ), 60}

    m := image.NewRGBA(image.Rect(0, 0, w, h))
    for x := 0; x < w; x++ {
        for y := 0; y < h; y++ {
```

```
            c := color.RGBA{
                cr.Brightness(float64(x), float64(y)),
                cg.Brightness(float64(x), float64(y)),
                cb.Brightness(float64(x), float64(y)),
                255,
            }
            m.Set(x, y, c)
        }
    }

    f, err := os.OpenFile("rgb.png", os.O_WRONLY|os.O_CREATE, 0600)
    if err != nil {
        fmt.Println(err)
        return
    }
    defer f.Close()
    png.Encode(f, m)
}
```

保存

保存图片也很简单，需要编码后，写入文件流即可：

- 注册图片的解码器
- 通过`os.create`创建文件；
- 通过`png.Encode`编码图片并写入文件；

示例：

```
func savePic(img *image.RGBA) {
    f, err := os.Create("C:\\tmp.jpg")
    if err != nil {
        panic(err)
    }
    defer f.Close()
    b := bufio.NewWriter(f)
    err = jpeg.Encode(b, img, nil)
    if err != nil {
        panic(err)
    }
    b.Flush()
}
```

图片修改

很多操作都需要用到绘制图片：

Draw

绘制图片

```
func Draw(dst Image, r image.Rectangle, src image.Image, sp image.Point, op Op)
```

主要参数说明：

- dst：绘图的背景图
- r：背景图的绘图区域
- src：要绘制的图
- sp：要绘制图src的开始点
- op：组合方式

DrawMask

DrawMask多了一个遮罩蒙层参数，Draw为其一种特殊形式（遮罩相关参数为nil）。

```
func DrawMask(dst Image, r image.Rectangle, src image.Image, sp image.Point, mask
image.Image, mp image.Point, op Op)
```

转换

读取的jpg图像不是RGBA格式的（为YCbCr格式）；在操作前需要先转换格式：

- 创建一个大小相同的RGBA图像；
- 把jpg画到新建的图像上去；

```
func jpg2RGBA(img image.Image) *image.RGBA {
    tmp := image.NewRGBA(img.Bounds())

    draw.Draw(tmp, img.Bounds(), img, img.Bounds().Min, draw.Src)
    return tmp
}
```

裁剪

通过subImage方法可方便地裁剪图片（需要为RGBA格式的）。

```
func subImg() {
    pic := readPic()
    fmt.Printf("Type: %T\n", pic)
    img := jpg2RCBA(pic)

    sub := img.SubImage(image.Rect(0, 0, pic.Bounds().Dx(),
pic.Bounds().Dy()/2))
    savePic(sub.(*image.RGBA))
}
```

缩放

图片缩放分为保持比例与不保持比例的缩放；保持比例时，要确定新图片的位置（是否居中），以及如何填充空白处。
为了缩放，需要引入新的库`golang.org/x/image/draw`。

在保持比例缩放时，需要先计算缩放后的图片大小：

- 分别计算宽、高的缩放比例，以小者为准；
- 若是居中（否则靠左上）需要计算填充大小，然后据此计算位置；

```

```

```
func calcResizedRect(width int, src image.Rectangle, height int, centerAlign
bool) image.Rectangle {
    var dst image.Rectangle
    if width*src.Dy() < height*src.Dx() { // width/src.width < height/src.height
        ratio := float64(width) / float64(src.Dx())

        tH := int(float64(src.Dy()) * ratio)
        pad := 0
        if centerAlign {
            pad = (height - tH) / 2
        }
        dst = image.Rect(0, pad, width, pad+tH)
    } else {
        ratio := float64(height) / float64(src.Dy())
        tW := int(float64(src.Dx()) * ratio)
        pad := 0
        if centerAlign {
            pad = (width - tW) / 2
        }
        dst = image.Rect(pad, 0, pad+tW, height)
    }

    return dst
}
```

有了缩放后的大小后，即可通过双线性插值bilinear的方式进行图片的缩放

- img为要缩放的图片
- width、height为缩放后的大小
- keepRatio为是否保持比例缩放
- fill为填充的颜色（R、G、B都为fill）
- centerAlign：保持比例缩放时，图片是否居中存放

```
import (
    "image"
    "image/color"

    "golang.org/x/image/draw"
)

func resizePic(img image.Image, width int, height int, keepRatio bool, fill int,
centerAlign bool) image.Image {
    outImg := image.NewRGBA(image.Rect(0, 0, width, height))
    if !keepRatio {
        draw.BiLinear.Scale(outImg, outImg.Bounds(), img, img.Bounds(),
draw.Over, nil)
        return outImg
    }

    if fill != 0 {
        fillColor := color.RGBA{R: uint8(fill), G: uint8(fill), B: uint8(fill),
A: 255}
        draw.Draw(outImg, outImg.Bounds(), &image.Uniform{C: fillColor},
image.Point{}, draw.Src)
    }
```

```
    dst := calcResizedRect(width, img.Bounds(), height, centerAlign)
    draw.ApproxBiLinear.Scale(outImg, dst.Bounds(), img, img.Bounds(), draw.Over,
nil)
    return outImg
}
```