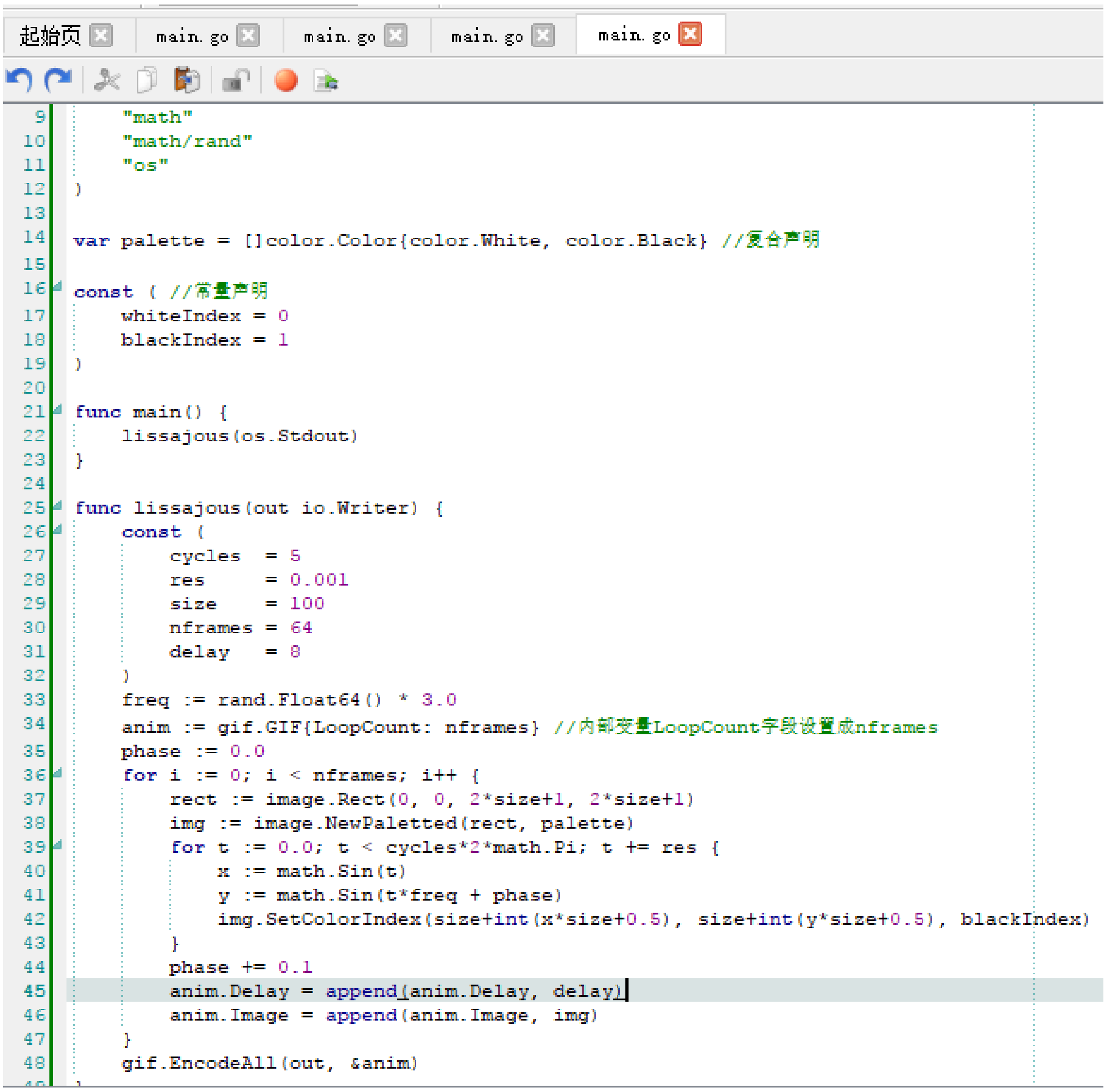
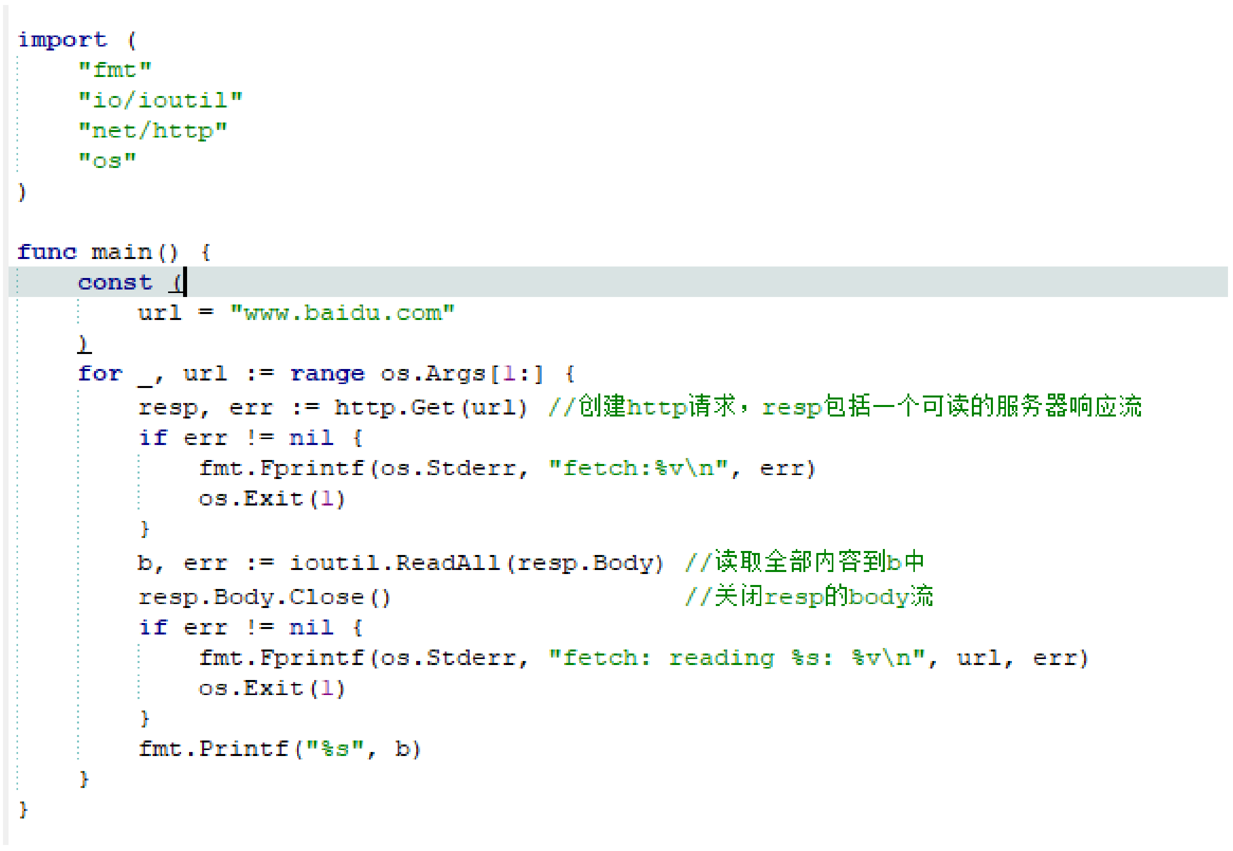
GIF动画



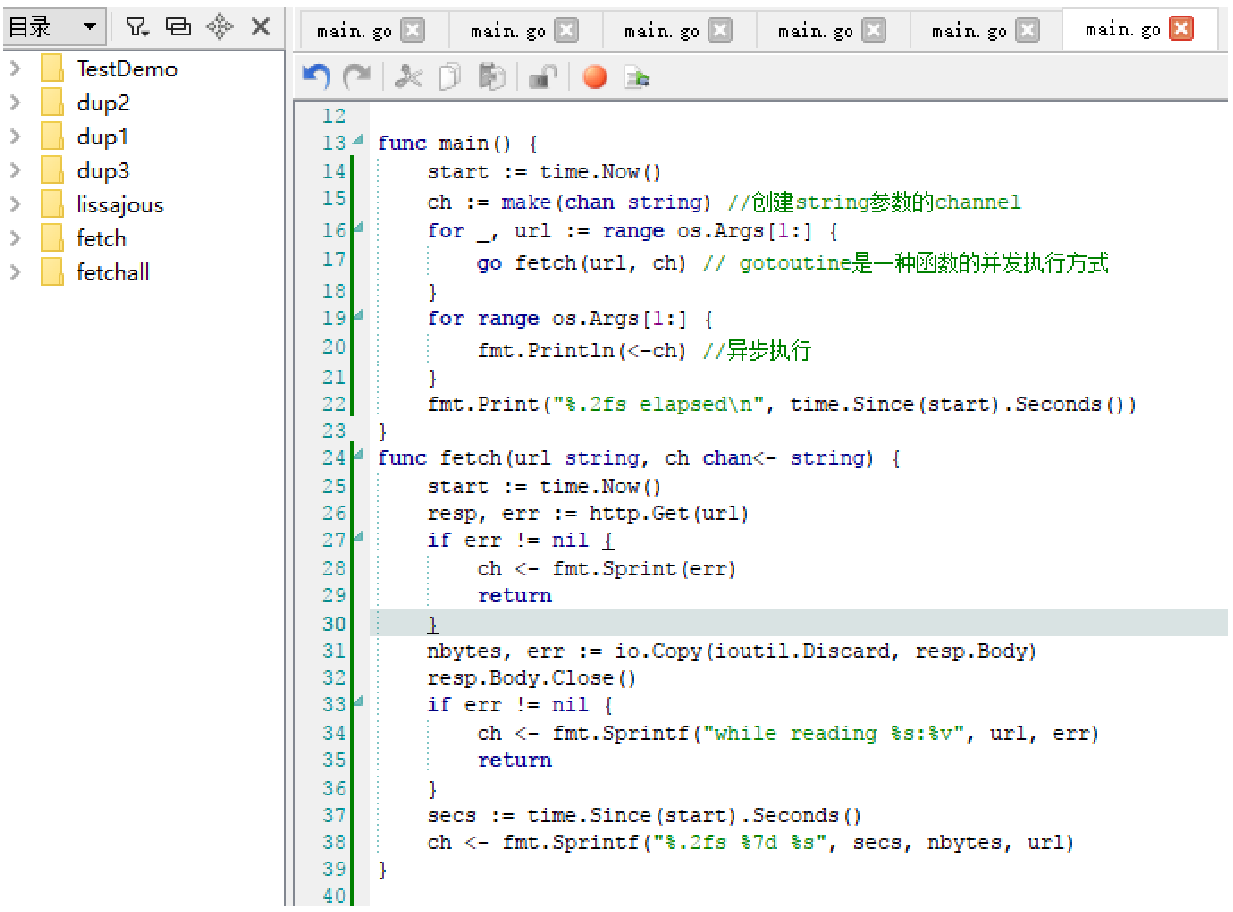
```go
    "math"
    "math/rand"
    "os"
)

var palette = []color.Color{color.White, color.Black} //复合声明

const ( //常量声明
    whiteIndex = 0
    blackIndex = 1
)

func main() {
    lissajous(os.Stdout)
}

func lissajous(out io.Writer) {
    const (
        cycles  = 5
        res     = 0.001
        size    = 100
        nframes = 64
        delay   = 8
    )
    freq := rand.Float64() * 3.0
    anim := gif.GIF{LoopCount: nframes} //内部变量LoopCount字段设置成nframes
    phase := 0.0
    for i := 0; i < nframes; i++ {
        rect := image.Rect(0, 0, 2*size+1, 2*size+1)
        img := image.NewPaletted(rect, palette)
        for t := 0.0; t < cycles*2*math.Pi; t += res {
            x := math.Sin(t)
            y := math.Sin(t*freq + phase)
            img.SetColorIndex(size+int(x*size+0.5), size+int(y*size+0.5), blackIndex)
        }
        phase += 0.1
        anim.Delay = append(anim.Delay, delay)
        anim.Image = append(anim.Image, img)
    }
    gif.EncodeAll(out, &anim)
```

获取url



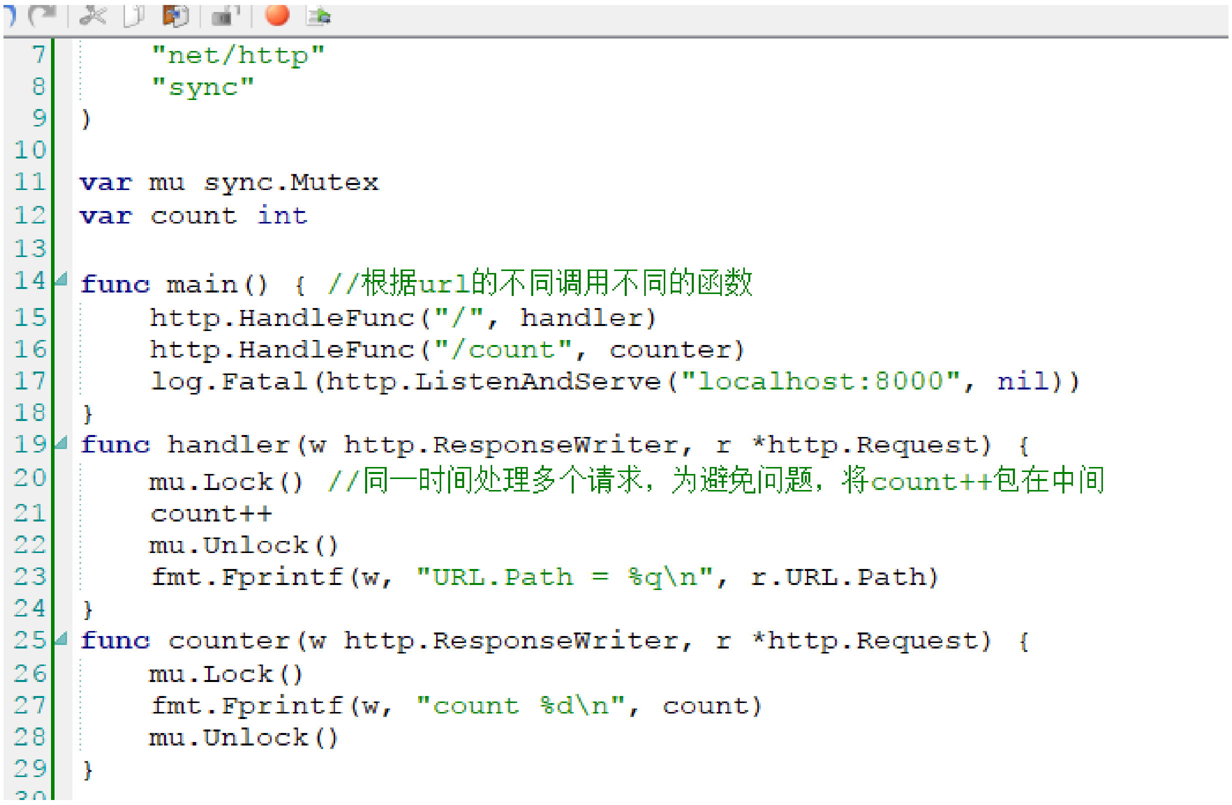
```go
import (
	"fmt"
	"io/ioutil"
	"net/http"
	"os"
)

func main() {
	const (
		url = "www.baidu.com"
	)
	for _, url := range os.Args[1:] {
		resp, err := http.Get(url) //创建http请求，resp包括一个可读的服务器响应流
		if err != nil {
			fmt.Fprintf(os.Stderr, "fetch:%v\n", err)
			os.Exit(1)
		}
		b, err := ioutil.ReadAll(resp.Body) //读取全部内容到b中
		resp.Body.Close()                   //关闭resp的body流
		if err != nil {
			fmt.Fprintf(os.Stderr, "fetch: reading %s: %v\n", url, err)
			os.Exit(1)
		}
		fmt.Printf("%s", b)
	}
}
```

并发获取多个url

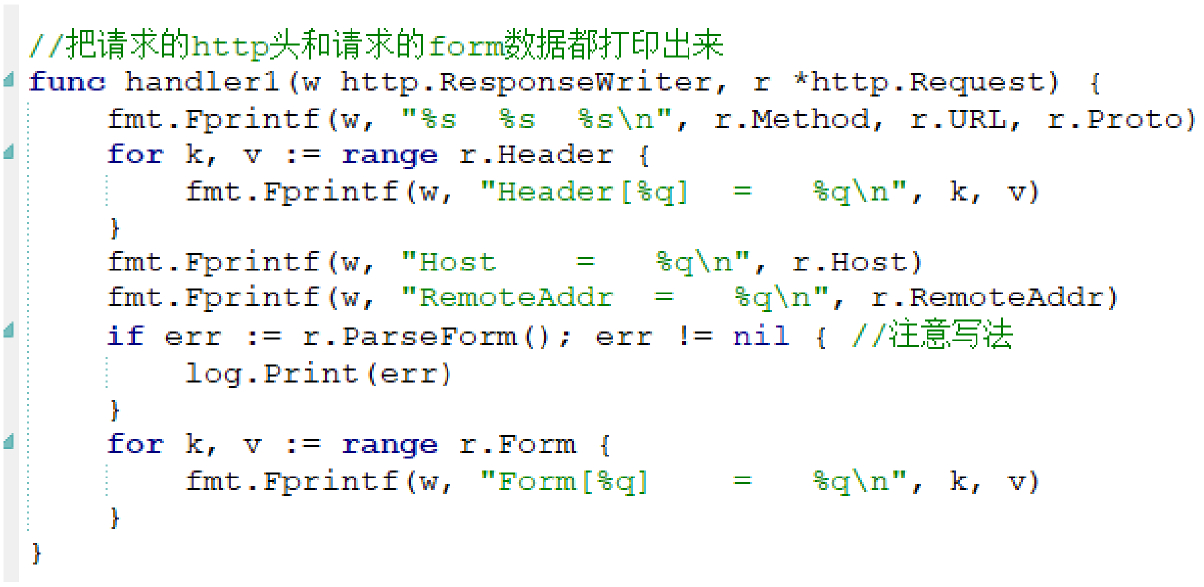


```go
func main() {
	start := time.Now()
	ch := make(chan string) //创建string参数的channel
	for _, url := range os.Args[1:] {
		go fetch(url, ch) // gotoutine是一种函数的并发执行方式
	}
	for range os.Args[1:] {
		fmt.Println(<-ch) //异步执行
	}
	fmt.Print("%.2fs elapsed\n", time.Since(start).Seconds())
}
func fetch(url string, ch chan<- string) {
	start := time.Now()
	resp, err := http.Get(url)
	if err != nil {
		ch <- fmt.Sprint(err)
		return
	}
	nbytes, err := io.Copy(ioutil.Discard, resp.Body)
	resp.Body.Close()
	if err != nil {
		ch <- fmt.Sprintf("while reading %s:%v", url, err)
		return
	}
	secs := time.Since(start).Seconds()
	ch <- fmt.Sprintf("%.2fs %7d %s", secs, nbytes, url)
}
```

web服务



```go
    "net/http"
    "sync"
)

var mu sync.Mutex
var count int

func main() { //根据url的不同调用不同的函数
    http.HandleFunc("/", handler)
    http.HandleFunc("/count", counter)
    log.Fatal(http.ListenAndServe("localhost:8000", nil))
}
func handler(w http.ResponseWriter, r *http.Request) {
    mu.Lock() //同一时间处理多个请求，为避免问题，将count++包在中间
    count++
    mu.Unlock()
    fmt.Fprintf(w, "URL.Path = %q\n", r.URL.Path)
}
func counter(w http.ResponseWriter, r *http.Request) {
    mu.Lock()
    fmt.Fprintf(w, "count %d\n", count)
    mu.Unlock()
}
```



```go
//把请求的http头和请求的form数据都打印出来
func handler1(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "%s  %s  %s\n", r.Method, r.URL, r.Proto)
    for k, v := range r.Header {
        fmt.Fprintf(w, "Header[%q]  =   %q\n", k, v)
    }
    fmt.Fprintf(w, "Host     =   %q\n", r.Host)
    fmt.Fprintf(w, "RemoteAddr  =   %q\n", r.RemoteAddr)
    if err := r.ParseForm(); err != nil { //注意写法
        log.Print(err)
    }
    for k, v := range r.Form {
        fmt.Fprintf(w, "Form[%q]    =   %q\n", k, v)
    }
}
```