Serving Files

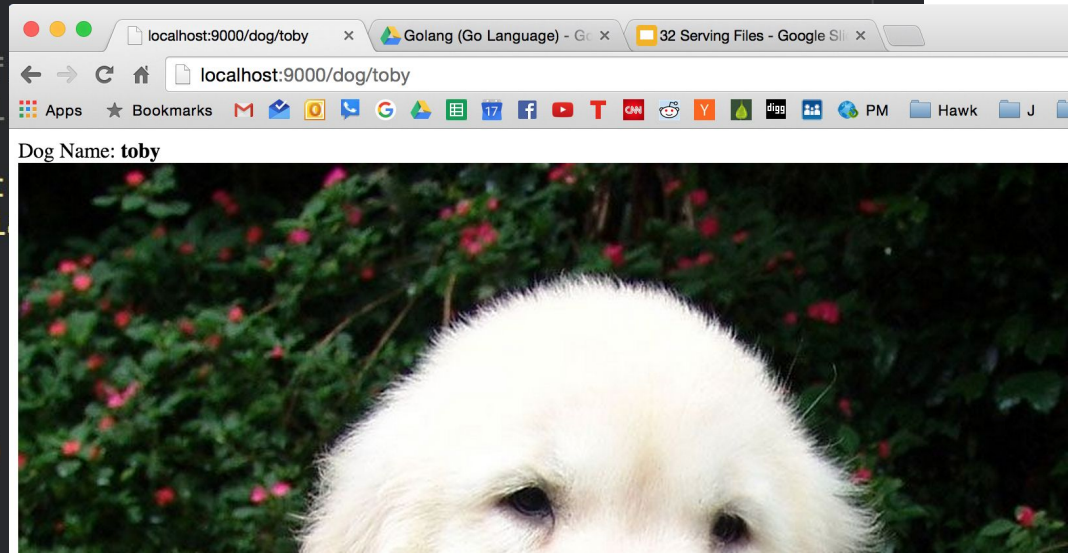
Serving Files

- io.Copy
- http.ServeContent
- http.ServeFile
- http.FileServer

io.Copy

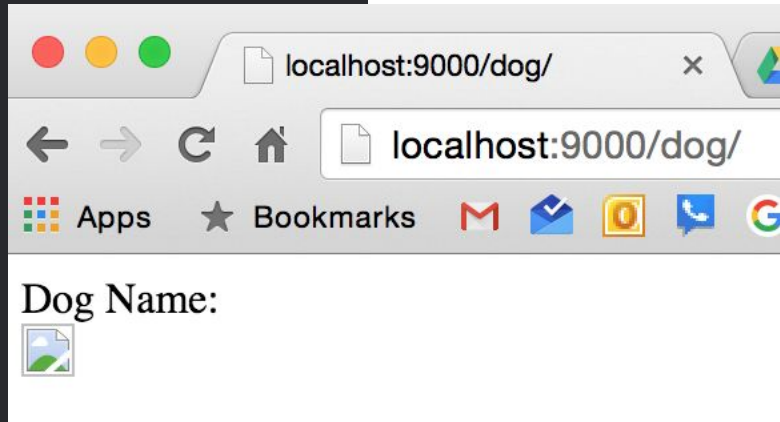
```go
package main

import (
    "io"
    "net/http"
    "strings"
)

func upTown(res http.ResponseWriter, req *http.Request) {
    res.Header().Set("Content-Type", "text/html; charset=utf-8")
    var dogName string
    fs := strings.Split(req.URL.Path, "/")
    if len(fs) >= 3 {
        dogName = fs[2]
    }
    // the image file is not coming f
    // in the next code samples, we'l
    io.WriteString(res, `
    Dog Name: <strong>`+dogName+`</st
    <img src="https://upload.wikimedi
    `)
}

func main() {
    http.HandleFunc("/dog/", upTown)
    http.ListenAndServe(":9000", nil)
}
```

image served from wikimedia …
how do I serve it from my own server?

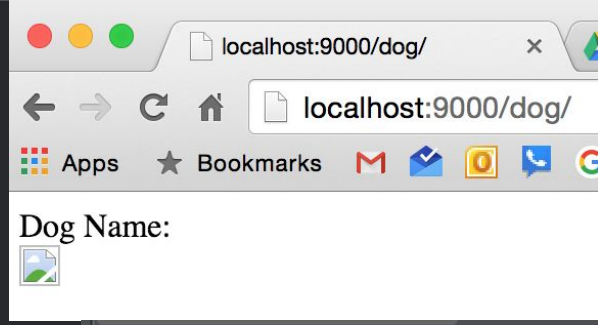


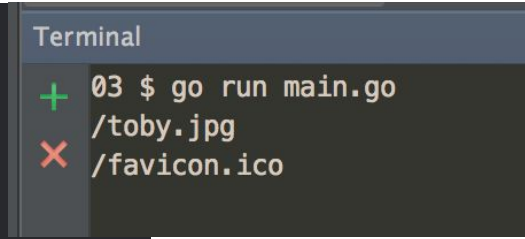
```go
package main

import (
    "io"
    "net/http"
    "strings"
)

func upTown(res http.ResponseWriter, req *http.Request) {
    res.Header().Set("Content-Type", "text/html; charset=utf-8")
    var dogName string
    fs := strings.Split(req.URL.Path, "/")
    if len(fs) >= 3 {
        dogName = fs[2]
    }
    // the image doesn't serve
    io.WriteString(res, `
    Dog Name: <strong>`+dogName+`</strong><br>
    <img src="/toby.jpg">
    `)
}

func main() {
    http.HandleFunc("/dog/", upTown)
    http.ListenAndServe(":9000", nil)
}
```

attempt to serve image ourselves
unsuccessful



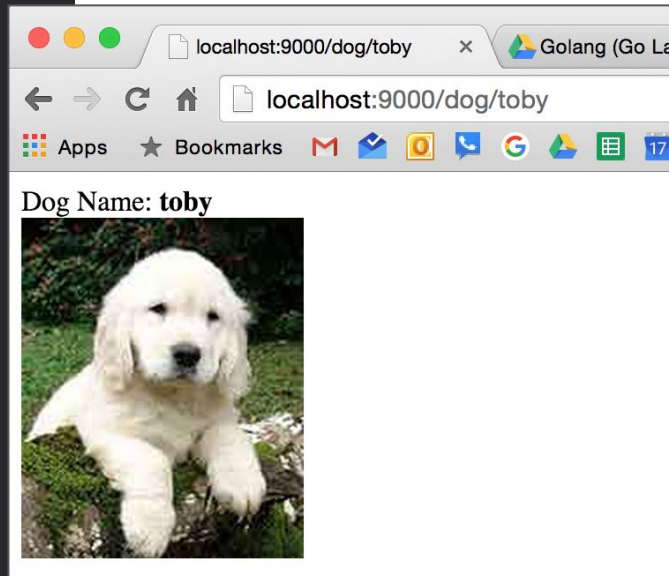
```go
package main

import (
    "io"
    "net/http"
    "strings"
    "fmt"
)

func upTown(res http.ResponseWriter, req *http.Request) {
    res.Header().Set("Content-Type", "text/html; charset=utf-8")
    var dogName string
    fs := strings.Split(req.URL.Path, "/")
    if len(fs) >= 3 {
        dogName = fs[2]
    }
    // the image doesn't serve
    io.WriteString(res, `
    Dog Name: <strong>`+dogName+`</strong><br>
    <img src="/toby.jpg">
    `)
}

func main() {
    http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request){
        fmt.Println(req.URL)
    })
    http.HandleFunc("/dog/", upTown)
    http.ListenAndServe(":9000", nil)
}
```





Add route to see
all requests to our server

```go
10 func upTown(res http.ResponseWriter, req *http.Request) {
11     res.Header().Set("Content-Type", "text/html; charset=utf-8")
12     var dogName string
13     fs := strings.Split(req.URL.Path, "/")
14     if len(fs) >= 3 {
15         dogName = fs[2]
16     }
17     // the image doesn't serve
18     io.WriteString(res, `
19     Dog Name: <strong>`+dogName+`</strong><br>
20     <img src="/toby.jpg">
21     `)
22 }
23
24 func dogPic(res http.ResponseWriter, req *http.Request) {
25     f, err := os.Open("toby.jpg")
26     if err != nil {
27         http.Error(res, "file not found", 404)
28         return
29     }
30     defer f.Close()
31
32     io.Copy(res, f)
33 }
34
35 func main() {
36     http.HandleFunc("/toby.jpg", dogPic)
37     http.HandleFunc("/dog/", upTown)
38     http.ListenAndServe(":9000", nil)
39 }
```



godoc.org/net/http

Bookmarks PM Hawk J

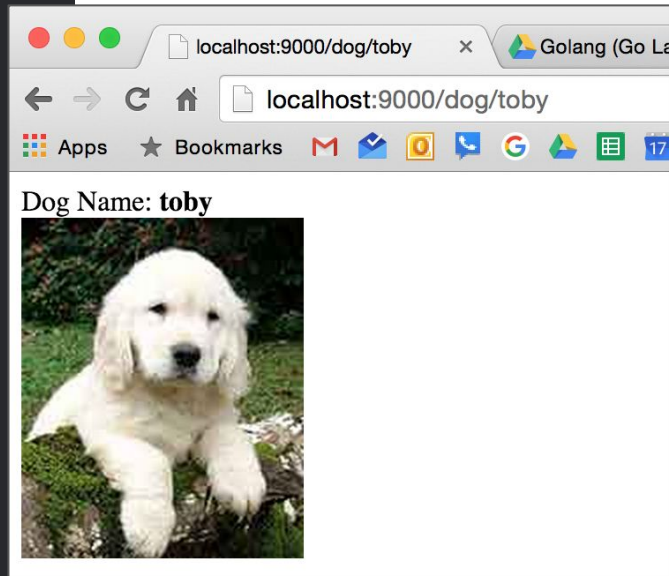
func Error

```
func Error(w ResponseWriter, error string, code int)
```

Error replies to the request with the specified error message and HTTP code. The error message should be plain text.

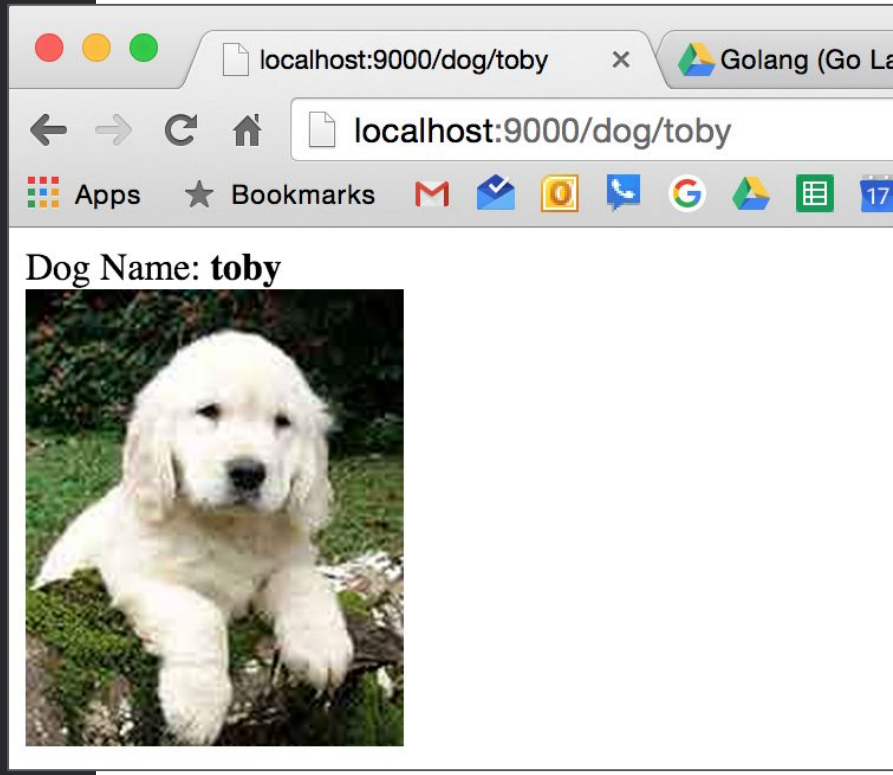
```go
10 func upTown(res http.ResponseWriter, req *http.Request) {
11     res.Header().Set("Content-Type", "text/html; charset=utf-8")
12     var dogName string
13     fs := strings.Split(req.URL.Path, "/")
14     if len(fs) >= 3 {
15         dogName = fs[2]
16     }
17     // the image doesn't serve
18     io.WriteString(res, `
19     Dog Name: <strong>`+dogName+`</strong><br>
20     <img src="/toby.jpg">
21     `)
22 }
23
24 func dogPic(res http.ResponseWriter, req *http.Request) {
25     f, err := os.Open("toby.jpg")
26     if err != nil {
27         http.Error(res, "file not found", 404)
28         return
29     }
30     defer f.Close()
31
32     io.Copy(res, f)
33 }
34
35 func main() {
36     http.HandleFunc("/toby.jpg", dogPic)
37     http.HandleFunc("/dog/", upTown)
38     http.ListenAndServe(":9000", nil)
39 }
```

- We should be setting the content type
- We should be using an **ETag**
- We could do all of this, but there's a much easier way ...



http.ServeContent

```go
10 func upTown(res http.ResponseWriter, req *http.Request) {
11     res.Header().Set("Content-Type", "text/html; charset=utf-8")
12     var dogName string
13     fs := strings.Split(req.URL.Path, "/")
14     if len(fs) >= 3 {
15         dogName = fs[2]
16     }
17     // the image doesn't serve
18     io.WriteString(res, `
19     Dog Name: <strong>`+dogName+`</strong><br>
20     <img src="/toby.jpg">
21     `)
22 }
23
24 func dogPic(res http.ResponseWriter, req *http.Request) {
25     f, err := os.Open("toby.jpg")
26     if err != nil {
27         http.Error(res, "file not found", 404)
28         return
29     }
30     defer f.Close()
31
32     fi, err := f.Stat()
33     if err != nil {
34         http.Error(res, "file not found", 404)
35         return
36     }
37
38     http.ServeContent(res, req, f.Name(), fi.ModTime(), f)
39 }
40
41 func main() {
42     http.HandleFunc("/toby.jpg", dogPic)
43     http.HandleFunc("/dog/", upTown)
44     http.ListenAndServe(":9000", nil)
45 }
```



ServeContent
fills in the headers for you

godoc.org/net/http

Bookmarks PM Hawk J

func ServeContent

```
func ServeContent(w ResponseWriter, req *Request, name string, modtime time.Time, cont
ent io.ReadSeeker)
```

os.*File implements
io.ReadSeeker interface
We can pass a *File in here

ServeContent needs *Request
to look at the request that came in
to see if ETag, etc

The name of the file

The last time the file
was modified

godoc.org/net/http

Bookmarks PM Hawk J

func ServeContent

```
func ServeContent(w ResponseWriter, req *Request, name string, modtime time.Time, cont
ent io.ReadSeeker)
```

ServeContent replies to the request using the content in the provided ReadSeeker. The main benefit of ServeContent over io.Copy is that it handles Range requests properly, sets the MIME type, and handles If-Modified-Since requests.

If the response's Content-Type header is not set, ServeContent first tries to deduce the type from name's file extension and, if that fails, falls back to reading the first block of the content and passing it to DetectContentType. The name is otherwise unused; in particular it can be empty and is never sent in the response.

If modtime is not the zero time or Unix epoch, ServeContent includes it in a Last-Modified header in the response. If the request includes an If-Modified-Since header, ServeContent uses modtime to decide whether the content needs to be sent at all.

The content's Seek method must work: ServeContent uses a seek to the end of the content to determine its size.

If the caller has set w's ETag header, ServeContent uses it to handle requests using If-Range and If-None-Match.

Note that *os.File implements the io.ReadSeeker interface.

godoc.org/io

type ReadSeeker

```
type ReadSeeker interface {
    Reader
    Seeker
}
```

ReadSeeker is the interface that groups the basic Read and Seek methods.

godoc.org/io

type Reader

```
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

Reader is the interface that wraps the basic Read method.

godoc.org/io

type Seeker

```
type Seeker interface {
    Seek(offset int64, whence int) (int64, error)
}
```

Seeker is the interface that wraps the basic Seek method.

Seek sets the offset for the next Read or Write to offset, interpreted according to whence: 0 means relative to the origin of the file, 1 means relative to the current offset, and 2 means relative to the end. Seek returns the new offset and an error, if any.

Seeking to a negative offset is an error. Seeking to any positive offset is legal, but the behavior of subsequent I/O operations on the underlying object is implementation-dependent.

godoc.org/os

type File

- func Create(name string) (*File, error)
- func NewFile(fd uintptr, name string) *File
- func Open(name string) (*File, error)
- func OpenFile(name string, flag int, perm FileMode) (*File, error)
- func Pipe() (r *File, w *File, err error)
- func (f *File) Chdir() error
- func (f *File) Chmod(mode FileMode) error
- func (f *File) Chown(uid, gid int) error
- func (f *File) Close() error
- func (f *File) Fd() uintptr
- func (f *File) Name() string
- func (f *File) Read(b []byte) (n int, err error)
- func (f *File) ReadAt(b []byte, off int64) (n int, err error)
- func (f *File) Readdir(n int) (fi []FileInfo, err error)
- func (f *File) Readdirnames(n int) (names []string, err error)
- func (f *File) Seek(offset int64, whence int) (ret int64, err error)
- func (f *File) Stat() (FileInfo, error)
- func (f *File) Sync() error
- func (f *File) Truncate(size int64) error
- func (f *File) Write(b []byte) (n int, err error)
- func (f *File) WriteAt(b []byte, off int64) (n int, err error)
- func (f *File) WriteString(s string) (n int, err error)

func (*File) Stat

```
func (f *File) Stat() (FileInfo, error)
```

Stat returns the FileInfo structure describing file. If there is an error, it will be of type *PathError.

godoc.org/os

okmarks PM Hawk J »

type FileInfo

```
type FileInfo interface {
    Name() string       // base name of the file
    Size() int64        // length in bytes for regular files; system-dependent for other
    Mode() FileMode     // file mode bits
    ModTime() time.Time // modification time
    IsDir() bool        // abbreviation for Mode().IsDir()
    Sys() interface{}   // underlying data source (can return nil)
}
```

A FileInfo describes a file and is returned by Stat and Lstat.

an even better, easier way ...

http.ServeFile

godoc.org/net/http

okmarks PM Hawk J

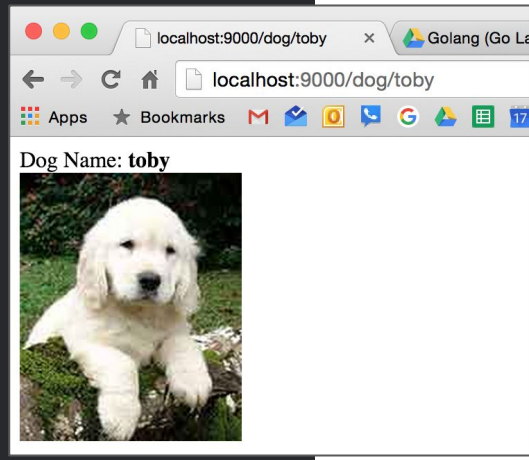
func ServeFile

```
func ServeFile(w ResponseWriter, r *Request, name string)
```

ServeFile replies to the request with the contents of the named file or directory.

As a special case, ServeFile redirects any request where r.URL.Path ends in "/index.html" to the same path, without the final "index.html". To avoid such redirects either modify the path or use ServeContent.

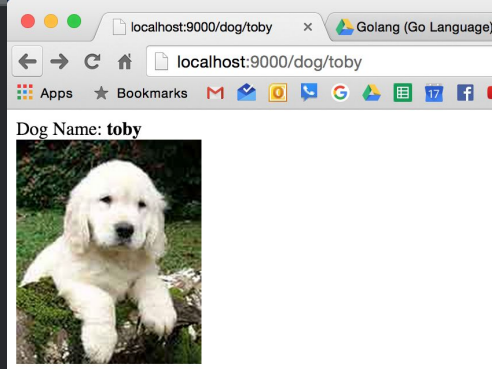
```go
9   func upTown(res http.ResponseWriter, req *http.Request) {
10      res.Header().Set("Content-Type", "text/html; charset=utf-8")
11      var dogName string
12      fs := strings.Split(req.URL.Path, "/")
13      if len(fs) >= 3 {
14          dogName = fs[2]
15      }
16      // the image doesn't serve
17      io.WriteString(res, `
18      Dog Name: <strong>`+dogName+`</strong><br>
19      <img src="/toby.jpg">
20      `)
21  }
22
23  func dogPic(res http.ResponseWriter, req *http.Request) {
24      http.ServeFile(res, req, "toby.jpg")
25  }
26
27  func main() {
28      http.HandleFunc("/toby.jpg", dogPic)
29      http.HandleFunc("/dog/", upTown)
30      http.ListenAndServe(":9000", nil)
31  }
```

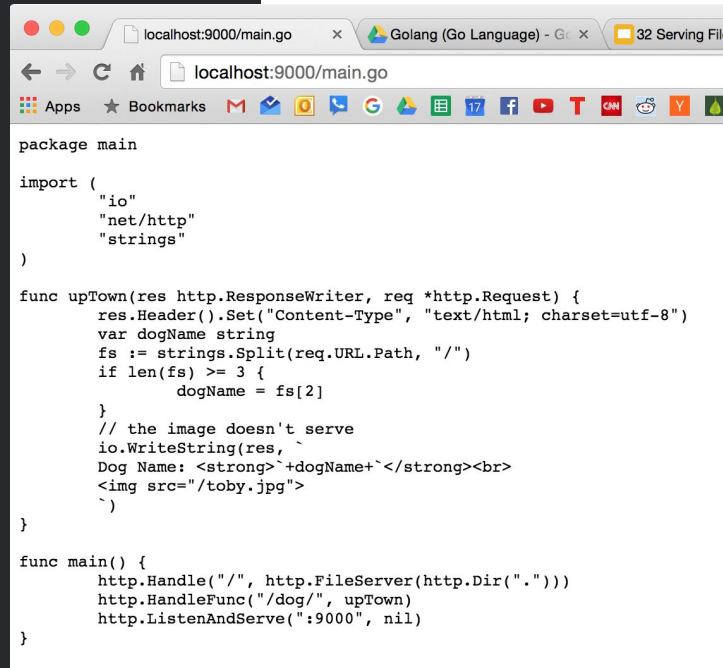


yet still, an even better, easier way ...

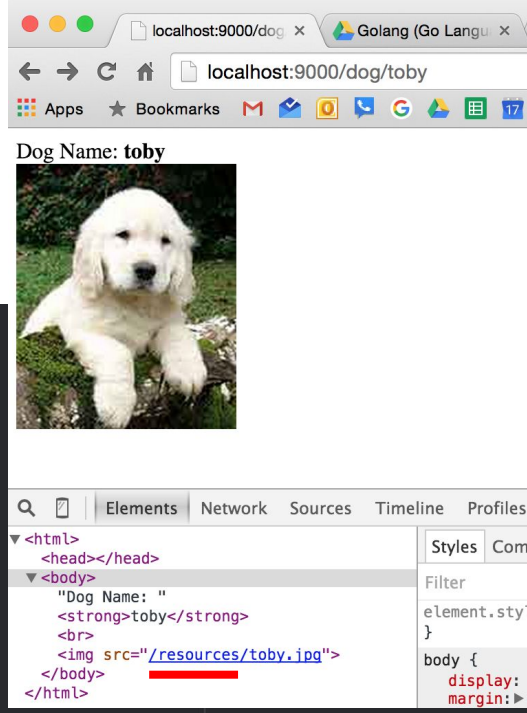
http.FileServer

```go
package main

import (
    "io"
    "net/http"
    "strings"
)

func upTown(res http.ResponseWriter, req *http.Request) {
    res.Header().Set("Content-Type", "text/html; charset=utf-8")
    var dogName string
    fs := strings.Split(req.URL.Path, "/")
    if len(fs) >= 3 {
        dogName = fs[2]
    }
    // the image doesn't serve
    io.WriteString(res, `
    Dog Name: <strong>`+dogName+`</strong><br>
    <img src="/toby.jpg">
    `)
}

func main() {
    http.Handle("/", http.FileServer(http.Dir(".")))
    http.HandleFunc("/dog/", upTown)
    http.ListenAndServe(":9000", nil)
}
```





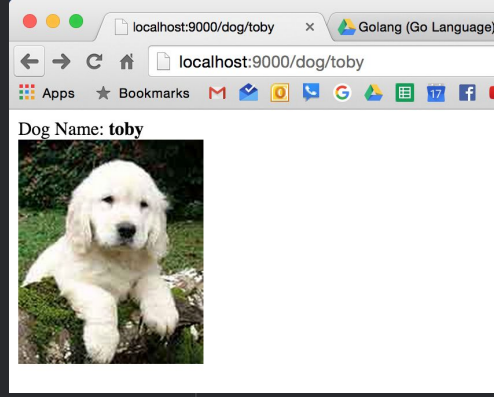
we ask for “/resources/toby.jpg”
this matches the routing on line 24
the file that we want is called “toby.jpg” not “/resources/toby.jpg”
we therefore need to strip “/resources” from what we’ve requested, the URI “/resources/toby.jpg”
This leaves us with just “/toby.jpg”
the FileServer then looks in “./assets” for “/toby.jpg”, eg, it looks for “./assets/toby.jpg”
even though we asked for “/resources/toby.jpg” in our HTML
on our server we served “./assets/toby.jpg”



Another way to say this:
I’m asking for “/resources/toby.jpg”
Where is file server going to look for this?
It is going to look in “./assets”
That means what I’m attempting to serve back to the client is
“./assets/resources/toby.jpg”

FileServer will serve files from
the specified FileSystem directory

```go
9  func upTown(res http.ResponseWriter, req *http.Request) {
10     res.Header().Set("Content-Type", "text/html; charset=utf-8")
11     var dogName string
12     fs := strings.Split(req.URL.Path, "/")
13     if len(fs) >= 3 {
14         dogName = fs[2]
15     }
16     // the image doesn't serve
17     io.WriteString(res, `
18     Dog Name: <strong>`+dogName+`</strong><br>
19     <img src="/assets/toby.jpg">
20     `)
21 }
22
23 func main() {
24     http.Handle("/assets/", http.StripPrefix("/assets", http.FileServer(http.Dir("./assets"))))
25     http.HandleFunc("/dog/", upTown)
26     http.ListenAndServe(":9000", nil)
27 }
```



we ask for “/assets/toby.jpg”
this matches the routing on line 24
the file that we want is called “toby.jpg” not “/assets/toby.jpg”
we therefore need to strip “/assets” from what we’ve requested, the URI “/assets/toby.jpg”
This leaves us with just “/toby.jpg”
the FileServer then looks in “./assets” for “/toby.jpg”, eg, it looks for “./assets/toby.jpg”

func Handle

```
func Handle(pattern string, handler Handler)
```

type Handler

```
type Handler interface {
    ServeHTTP(ResponseWriter, *Request)
}
```

type Handler

- func FileServer(root FileSystem) Handler
- func NotFoundHandler() Handler
- func RedirectHandler(url string, code int) Handler
- func StripPrefix(prefix string, h Handler) Handler
- func TimeoutHandler(h Handler, dt time.Duration, msg string) Handler

func StripPrefix

```
func StripPrefix(prefix string, h Handler) Handler
```

StripPrefix returns a handler that serves HTTP requests by removing the given prefix from the request URL's Path and invoking the handler h. StripPrefix handles a request for a path that doesn't begin with prefix by replying with an HTTP 404 not found error.

func FileServer

```
func FileServer(root FileSystem) Handler
```

FileServer returns a handler that serves HTTP requests with the contents of the file system rooted at root.

type Dir

```
type Dir string
```

A Dir implements FileSystem using the native file system restricted to a specific directory tree.

While the FileSystem.Open method takes '/'-separated paths, a Dir's string value is a filename on the native file system, not a URL, so it is separated by filepath.Separator, which isn't necessarily '/'.

An empty Dir is treated as ".".

godoc.org/net/http

okmarks

type FileSystem

```
type FileSystem interface {
    Open(name string) (File, error)
}
```

type Dir

- func (d Dir) Open(name string) (File, error)

conversion

```
func main() {
    http.Handle("/assets/", http.StripPrefix("/assets", http.FileServer(http.Dir("./assets"))))
```

exercise

Create a static http server (perhaps named static-http) that serves the contents of the current working directory.

main.go

```go
package main

import (
    "log"
    "net/http"
)

func main() {
    log.Fatal(http.ListenAndServe(":9000", http.FileServer(http.Dir("."))))
}

```

```
Sun Sep 20 03:03:21 PDT 2015
GolangTraining $ cd 45_serving-files/10_static-file-server/
10_static-file-server $ go install
10_static-file-server $
```

```
Documents $ pwd
/Users/tm002/Documents
Documents $ 10_static-file-server
```

localhost:9000

Golang (Go Languag

localhost:9000

Apps Bookmarks

.DS_Store
.gitignore
.localized
appengine-guestbook-go/
Atom.app/
course_files/
CreatingWebsites/
favicon.ico
final_202.pdf
fonts/
go/
go books/
goblueprints/
golib/
graphics/
how to build web apps with go/
Microsoft User Data/
PS_v04.05.00_Mac.zip
sample.html
todd_mcleod.jpg
todd_mcleod_cu.jpg
todd_mcleod_cu_web.jpg

Don’t ever trust a file from a user

file inclusion vulnerability