errata

set http headers, request URLs, ServeFile

main.go

```go
package main

import (
    "fmt"
    "net/http"
)

func main() {
    http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
        res.Header().Set("Content-Type", "text/plain")
        res.Header()["Content-Type"] = []string{"text/plain"} // same as line 10
        fmt.Fprint(res, "Dog")
    })

    http.ListenAndServe(":9000", nil)
}

```

not a good way to do it, though

localhost:9000/helloeveryone

Golang (Go Language) - Go

localhost:9000/helloeveryone

Apps Bookmarks

Dog

Elements Network Sources Timeline Profiles Resources Aud

View: Preserve log Disable cache No throttl

Name

helloeveryone

favicon.ico

Headers Preview Response Timing

General

Remote Address: [::1]:9000

Request URL: http://localhost:9000/hello

Request Method: GET

Status Code: 200 OK

Response Headers view source

Content-Length: 3

Content-Type: text/plain

Date: Tue, 22 Sep 2015 08:35:46 GMT

type ResponseWriter

```
type ResponseWriter interface {
    // Header returns the header map that will be sent by
    // WriteHeader. Changing the header after a call to
    // WriteHeader (or Write) has no effect unless the modified
    // headers were declared as trailers by setting the
    // "Trailer" header before the call to WriteHeader (see example).
    // To suppress implicit response headers, set their value to nil.
    Header() Header

    // Write writes the data to the connection as part of an HTTP reply.
    // If WriteHeader has not yet been called, Write calls WriteHeader(http.StatusOK)
    // before writing the data.  If the Header does not contain a
    // Content-Type line, Write adds a Content-Type set to
    // the initial 512 bytes of written data to DetectConte
    Write([]byte) (int, error)

    // WriteHeader sends an HTTP response header with statu
    // If WriteHeader is not called explicitly, the first c
    // will trigger an implicit WriteHeader(http.StatusOK).
    // Thus explicit calls to WriteHeader are mainly used t
    // send error codes.
    WriteHeader(int)
}
```

A ResponseWriter interface is used by an HTTP handler to construct an HTT

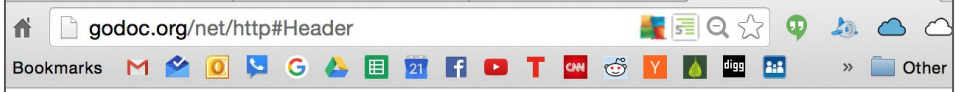
https://godoc.org/net/http#Header

okmarks

type Header

```
type Header map[string][]string
```

A Header represents the key-value pairs in an HTTP header.



type Header

```
type Header map[string][]string
```

A Header represents the key-value pairs in an HTTP header.

func (Header) Add

```
func (h Header) Add(key, value string)
```

Add adds the key, value pair to the header. It appends to any existing values associated with key.

func (Header) Del

```
func (h Header) Del(key string)
```

Del deletes the values associated with key.

func (Header) Get

```
func (h Header) Get(key string) string
```

Get gets the first value associated with the given key. If there are no values associated with the key, Get returns "". To access multiple values of a key, access the map directly with CanonicalHeaderKey.

func (Header) Set

```
func (h Header) Set(key, value string)
```

Set sets the header entries associated with key to the single element value. It replaces any existing values associated with key.

func (Header) Write

```
func (h Header) Write(w io.Writer) error
```

Write writes a header in wire format.

func (Header) WriteSubset

```
func (h Header) WriteSubset(w io.Writer, exclude map[string]bool) error
```

WriteSubset writes a header in wire format. If exclude is not nil, keys where exclude[key] == true are not written.

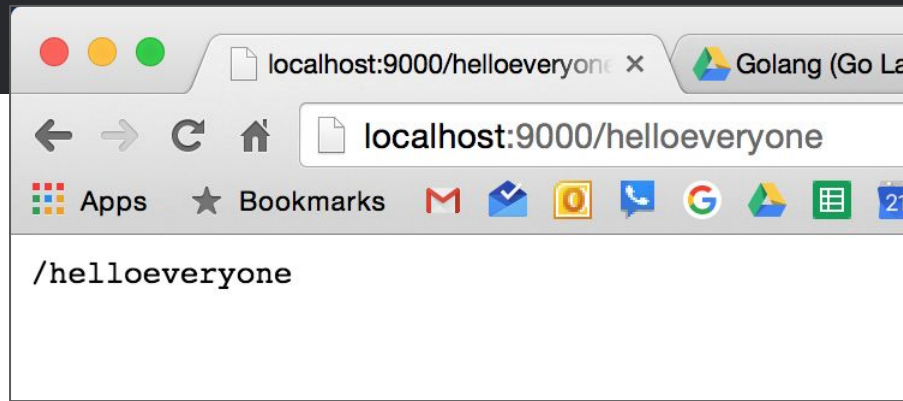
main.go

```go
package main

import (
    "fmt"
    "net/http"
)

func main() {
    http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
        res.Header().Set("Content-Type", "text/plain")
        fmt.Fprint(res, req.RequestURI)
    })

    http.ListenAndServe(":9000", nil)
}
```



http://godoc.org/net/http#Request

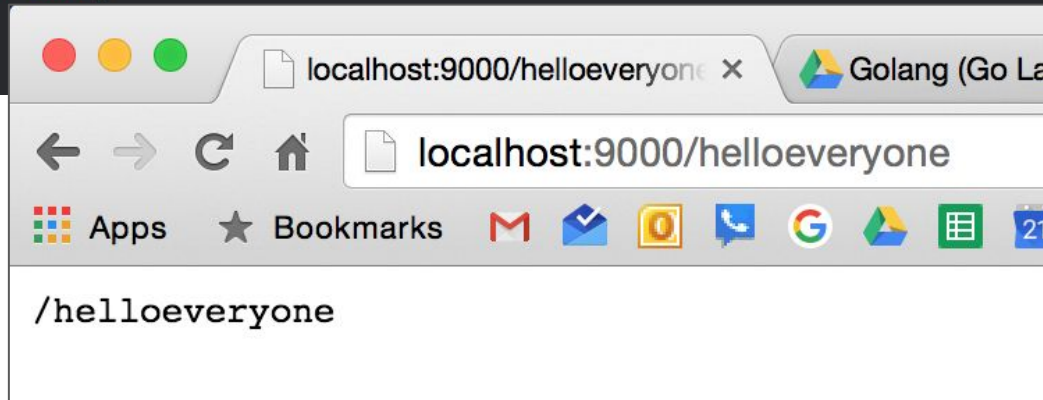
main.go

```go
package main

import (
    "fmt"
    "net/http"
)

func main() {
    http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
        res.Header().Set("Content-Type", "text/plain")
        fmt.Fprint(res, req.URL.Path)
    })

    http.ListenAndServe(":9000", nil)
}
```



type URL

```
type URL struct {
    Scheme   string
    Opaque   string    // encoded opaque data
    User     *Userinfo // username and password information
    Host     string    // host or host:port
    Path     string
    RawPath  string // encoded path hint (Go 1.5 and later only; see EscapedPath method)
    RawQuery string // encoded query values, without '?'
    Fragment string // fragment for references, without '#'
}
```

A URL represents a parsed URL (technically, a URI reference). The general form represented is:

```
scheme://[userinfo@]host/path[?query][#fragment]
```

URLs that do not start with a slash after the scheme are interpreted as:

```
scheme:opaque[?query][#fragment]
```

Note that the Path field is stored in decoded form: /%47%6f%2f becomes /Go/. A consequence is that it is impossible to tell which slashes in the Path were slashes in the raw URL and which were %2f. This distinction is rarely important, but when it is, code must not use Path directly.

Go 1.5 introduced the RawPath field to hold the encoded form of Path. The Parse function sets both Path and RawPath in the URL it returns, and URL's String method uses RawPath if it is a valid encoding of Path, by calling the EncodedPath method.

In earlier versions of Go, the more indirect workarounds were that an HTTP server could consult req.RequestURI and an HTTP client could construct a URL struct directly and set the Opaque field instead of Path. These still work as well.

Example

Example (Opaque)

Example (Roundtrip)

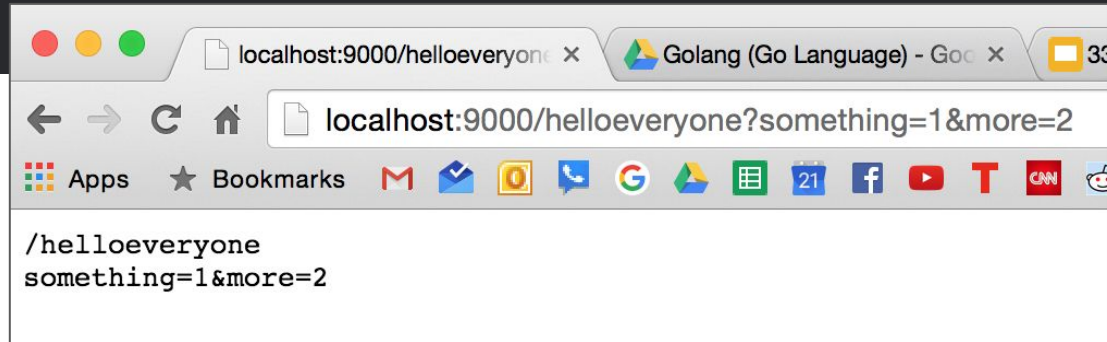
main.go

```go
package main

import (
    "fmt"
    "net/http"
)

func main() {
    http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
        res.Header().Set("Content-Type", "text/plain")
        fmt.Fprintln(res, req.URL.EscapedPath())
        fmt.Fprintln(res, req.URL.RawQuery)
    })

    http.ListenAndServe(":9000", nil)
}
```



main.go

```go
package main

import (
    "net/http"
)

func main() {
    http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
        res.Header().Set("Content-Type", "text/plain")
        http.ServeFile(res, req, "temp.txt")
    })

    http.ListenAndServe(":9000", nil)
}
```

When you compile your program, the files and folders to serve don't get compiled with it. You need to run your binary from a folder location relative to the files it is going to access. For instance, in a previous example we had an assets folder. We would need our binary to run from a folder that had that assets folder.

FYI, go-bindata will generate go code of files: creates a giant byte slice of a file and embeds it in the binary