**Golang refactoring tools**

Go language provides many useful tools as part of its development eco system.

**Gofmt**

In average programming languages developers can adapt to different formatting styles. Common problem is how to approach unknown code base without a long prescriptive style guide.

Go takes an unusual approach and keep this responsibility to format the source code for you. The gofmt program (available as go fmt, which examines on the package level rather than source file level) reads a Go syntax and reformat your program in a standard coding style. In addition, it provides some additional refactoring capabilities, which will explore in detail.

*// The -w flag overwrites the files instead of prints out the result on the screen*

c> gofmt -w message.go

It formats the following code snippet:

*// filename: message.go*

package message

import "fmt"

func **FormatMessage**(name string) string{

**if** **len**(name) == 0 { **return** "Welcome" } **else** { **return** fmt.Sprintf("Hi, %s", name) }

}

Output:

*// filename: message.go*

package message

import "fmt"

func **FormatMessage**(name string) string {

**if** **len**(name) == 0 {

**return** "Welcome"

} **else** {

**return** fmt.Sprintf("Hi, %s", name)

}

}

*Note that gofmt uses tabs for indentation and blanks for alignment.*

The code is reformatted to obey all Go coding style standards. It does not rename any variables and functions. There is a tool that do static analyses on your code.

These are the flags supported by gofmt:

* -d prints diffs to standard out when file formatting is changed
* -e print all errors
* -l prints the filename to standard out when file formatting is changed
* -r applies the rewrite rule to the source before reformatting.
* -s simplifies code
* -w overwrites file with its formatted version

In the next two paragraphs we will explore how to simplify and apply rewrites rules to a source code.

Simplifing source code is applied when -s flag is presented. It improves the code readability by replacing blocks of code with their sipliefied syntax version.

Executing **go fmt -s -w transport.go:**

// filename: transport.go

**package** transport

import "fmt"

type Endpoint struct {

Protocol string

Host string

Port **int**

}

var endpoints []Endpoint = []Endpoint{

Endpoint{

Protocol: "HTTP",

Host: "localhost",

Port: 80},

Endpoint{

Protocol: "SSH",

Host: "10.10.5.9.xip.io",

Port: 22}}

func ListEndpoints(startIndex **int**) {

**for** **index**, \_ := range endpoints[startIndex:len(endpoints)] {

endpoint := endpoints[**index**]

fmt.Printf("Priority: %d Procotol: %s Address: %s:%d\n",

**index**, endpoint.Protocol, endpoint.Host, endpoint.Port)

}

}

The package will be simplified to:

// filename: transport.go

**package** transport

import "fmt"

type Endpoint struct {

Protocol string

Host string

Port **int**

}

var endpoints []Endpoint = []Endpoint{

{Protocol: "HTTP",

Host: "localhost",

Port: 80},

{Protocol: "SSH",

Host: "10.10.5.9.xip.io",

Port: 22}}

func ListEndpoints(startIndex **int**) {

**for** **index** := range endpoints[startIndex:] {

endpoint := endpoints[**index**]

fmt.Printf("Priority: %d Procotol: %s Address: %s:%d\n",

**index**, endpoint.Protocol, endpoint.Host, endpoint.Port)

}

}

These are the applied rules:

* An array, slice, or map composite literal of the form []T{T{}, T{}} will be simplified to []T{{}, {}}.
* A slice expression of the form s[a:len(s)] will be simplified to s[a:].
* A range of the form for x, \_ = range v {...} will be simplified to for x = range v {...}.
* A range of the form for \_ = range v {...} will be simplified to for range v {...}.

To define specified rewrite rule the -r flag must be used. It should be in the following format:

**pattern** -> replacement

Both pattern and replacement must be valid Go expressions. The pattern serves as wildcards matching arbitrary sub-expressions. They will be substituted for the same identifiers in the replacement.

Lets rename Endpoint struct to Server in transport package:

$ gofmt -r 'Endpoint -> Server' -w transport.go

$ gofmt -r 'endpoints -> servers' -w transport.go

$ gofmt -r 'ListEndpoints -> ListServers' -w transport.go

The result of this operation:

// filename: transport.go

**package** transport

import "fmt"

type Server struct {

Protocol string

Host string

Port **int**

}

var servers []Server = []Server{

{Protocol: "HTTP",

Host: "localhost",

Port: 80},

{Protocol: "SSH",

Host: "10.10.5.9.xip.io",

Port: 22}}

func ListServers(startIndex **int**) {

**for** **index** := range servers[startIndex:] {

endpoint := servers[**index**]

fmt.Printf("Priority: %d Procotol: %s Address: %s:%d\n",

**index**, endpoint.Protocol, endpoint.Host, endpoint.Port)

}

}