**課題１**

1. golangをインストール

2. /usr/local/にgoというディレクトリ作成

3. /usr/local/go/にsrc と bin ディレクトリ作成

4. GOPATHを/usr/local/go/srcを設定

5. /usr/local/go/src/ に server.goを作成し Handlerを記述

6. sudo go run server.go を実行して動作を確認

**課題2**

1. 課題１のコードにbasic認証の行うauthHandler を実装

2. curl -v "http://localhost:8080/secret/"を実行して以下のレスポンスを確認

\* Trying ::1...

TCP\_NODELAY set

\* Connected to localhost (::1) port 8080 (#0)

> GET /secret/ HTTP/1.1

> Host: localhost:8080

> User-Agent: curl/7.54.0

> Accept: \*/\*

>

< HTTP/1.1 401 Unauthorized

< Date: Wed, 26 Jun 2019 12:16:55 GMT

< Content-Length: 0

<

\* Connection #0 to host localhost left intact

**課題3**

1. 課題２のコードに計算を行うcalcHandler を実装

2. curl "http://localhost:8080/calc?ldsafw"を実行して以下のレスポンスを確認ERROR

3. curl "http://localhost:8080/calc?1+4+9"を実行して以下のレスポンスを確認

14

ソースコード（課題１，課題２のコードも含まれます）

package main

import (

"fmt"

"go/token"

"go/types"

"log"

"net/http"

)

// 課題2

const (

basicAuthUser = "amazon"

basicAuthPassword = "candidate"

)

// 課題1

func amazonHandler(w http.ResponseWriter, r \*http.Request) {

\_, \_ = fmt.Fprintf(w, "AMAZON")

}

func authHandler(w http.ResponseWriter, r \*http.Request) {

if user, pass, ok := r.BasicAuth(); !ok || user != basicAuthUser || pass != basicAuthPassword {

//w.Header().Set("", "")

w.WriteHeader(http.StatusUnauthorized)

//http.Error(w, "ERROR", 401)

} else {

\_, \_ = fmt.Fprintf(w, "SUCCESS")

}

}

// 課題3

func calcHandler(w http.ResponseWriter, r \*http.Request) {

result, \_ := types.Eval(token.NewFileSet(), nil, token.NoPos, r.URL.RawQuery)

if result.Type == nil {

\_, \_ = fmt.Fprintf(w, "ERROR")

return

}

\_, \_ = fmt.Fprintf(w, result.Value.String())

}

func main() {

// 課題1

http.HandleFunc("/", amazonHandler)

// 課題2

http.HandleFunc("/secret/", authHandler)

// 課題3

http.HandleFunc("/calc", calcHandler)

if err := http.ListenAndServe(":8080", nil); err != nil {

log.Fatal(err)

}

}

**課題4**

1. 課題3のコードに在庫管理をするstockHandlerを実装

2. 与えられたfunctionが課題で用意されているfunctionかどうか確認

3. 与えられるswitch 文をfunction の値に応じて処理を変更

4. function = deleteallの場合，売上を入れる変数と在庫を管理するスライスを初期化

5. function = addstockの場合，nameがスライスに存在しなければ新たに追加する，存在する場合，在庫数を追加する．この時，追加する数が整数値かどうかも判定している

6. function = checkstockの場合，nameが与えられていればその在庫を表示，なければ，nameでソートを行い，在庫数が0以外の商品を表示する

7. function = sellの場合，まずnameが存在しているかどうか確認，存在していれば次にamountを確認し売上に合算して，amountの分だけ在庫を減らす．この時，price については，与えられていなければ0となるようにコードがなっているため，priceの値を判断する処理は行っていない．

8. function = checksalesの場合，変数salesを表示する

ソースコード（課題４のみのコードになります）

package main

import (

"fmt"

"log"

"net/http"

"net/url"

"sort"

"strconv"

)

// 課題4

var function = []string{

"addstock",

"checkstock",

"sell",

"checksales",

"deleteall",

}

// 課題4

func isFunc(key string, functions []string) bool {

for \_, function := range functions {

if key == function {

return true

}

}

return false

}

func stockHandler(w http.ResponseWriter, r \*http.Request) {

values, err := url.ParseQuery(r.URL.RawQuery)

if err != nil {

\_, \_ = fmt.Fprintf(w, "ERROR")

}

if !isFunc(values.Get("function"), function) {

\_, \_ = fmt.Fprintf(w, "ERROR")

}

switch values.Get("function") {

// addstock

case function[0]:

if \_, ok := stocks[values.Get("name")]; !ok {

n, err := strconv.Atoi(values.Get("amount"))

if err != nil && values.Get("amount") == "" {

fmt.Println(err)

stocks[values.Get("name")] = 1

} else if err != nil {

\_, \_ = fmt.Fprintf(w, "ERROR")

return

}

stocks[values.Get("name")] += n

} else {

if values.Get("amount") == "" {

stocks[values.Get("name")] += 1

} else {

tmpNum, \_ := strconv.Atoi(values.Get("amount"))

stocks[values.Get("name")] += tmpNum

}

}

// checkstock

case function[1]:

if name := values.Get("name"); name != "" {

\_, \_ = fmt.Fprintf(w, name+": "+strconv.Itoa(stocks[name]))

} else {

data := Data{}

for k, v := range stocks {

tmp := StockData{k, v}

data = append(data, tmp)

}

fmt.Println(data)

//sort.Sort(data)

sort.SliceStable(data, func(i, j int) bool {

return data[i].Name < data[j].Name

})

fmt.Println(data)

for \_, d := range data {

if d.Amount == 0 {

continue

}

\_, \_ = fmt.Fprint(w, d.Name+": "+strconv.Itoa(d.Amount)+"\n")

}

}

// sell

case function[2]:

if \_, ok := stocks[values.Get("name")]; !ok {

\_, \_ = fmt.Fprintf(w, "ERROR")

return

}

if values.Get("amount") == "" {

if stocks[values.Get("name")] >= 1 {

price, \_ := strconv.Atoi(values.Get("price"))

sales += price

stocks[values.Get("name")] -= 1

}

} else {

if tmpNum, \_ := strconv.Atoi(values.Get("amount")); stocks[values.Get("name")] >= tmpNum {

price, \_ := strconv.Atoi(values.Get("price"))

sales += price \* tmpNum

stocks[values.Get("name")] -= tmpNum

}

}

// checksales

case function[3]:

\_, \_ = fmt.Fprintf(w, "sales: "+strconv.Itoa(sales))

// delete

case function[4]:

stocks = make(map[string]int)

sales = 0

}

fmt.Println(stocks)

}

func main() {

// 課題4

http.HandleFunc("/stocker", stockHandler)

if err := http.ListenAndServe(":8080", nil); err != nil {

log.Fatal(err)

}

}