

付 録

GoのWebサーバでフォームを使う

本書のサンプルプログラムの結果は、GoのWebサーバ上で表示しましたが、常に同じフォント、「\n」による改行でした。

本付録では、GoのWebサーバ上でHTMLタグを使ってのページ表示、とりわけフォームによるデータ送信について手順中心に説明します。

A-1

HTMLファイルを表示

GoのWebサーバでHTMLファイルを表示させるには、読み込む内容が「HTML文書」であることを明確にする必要があります。

すなわち「ちゃんとしたHTMLファイルを書く」ことです。

■GoファイルとHTMLファイル

●新しいフォルダと新しいgoファイル

第3章からずっと使ってきたtrueserverフォルダの各モジュールとは別に、新しくwebappというフォルダを作り、そこにファイル「webapp.go」を作成します。

●HTMLファイル

HTMLファイルとして、「form.html」を「webapp」フォルダの中、「webapp.go」と同位置になるように配置します。

図A-1は、第3章から使ってきた「trueserver」フォルダの隣に「webapp」フォルダを作成した様子です。これで大丈夫です。

```
> trueserver
✓ webapp
<> form.html
GO webapp.go
```

図A-1 webapp/webapp.goとform.html

■フォームを表示する

●form.htmlの最初の内容

「form.html」の最初は、形だけでどこにも送信を指定しない内容にします。(リストA-1)

リストA-1のように、「DOCTYPE」や「html タグ」により、HTML 文書であることを明記してください。

リストA-1 「form.html」の最初の内容

```
<!DOCTYPE html>
<html>
  <h2>いろいろな入力をするページ</h2>
  <form>
    <p><input name="input" size="50"></p>
    <p><input type="submit" value="Go"></p>
  </form>
</html>
```

●「webapp.go」には構造体Pageを作成

「webapp.go」は、第3章から用いてきた「trueserver.go」と同じようなコードになりますが、重要なのは、「ページを表示させるための構造体」を作ることです。

形式は自由ですが、とにかく「フィールドをもつ構造体」です。

ここでは、フィールドNameとAnswerをもつ構造体「Page」を作成します。

HTML ファイルでフィールドを読み込むことがあるので、フィールドは「他のプログラムから探せる」大文字にしてください。

まず、「webapp.go」の中身をリストA-2のようにします。

リストA-2 「webapp.go」の最初の中身

```
package main

import (
    "net/http"
    "html/template" //新しい
)

type Page struct{ //形式は自由だが、とにかく構造体
    //フィールドは大文字
    Name string
    Answer string
}

func makepage(name string)Page{
```

[A-1] HTML ファイルを表示

```
return Page{name, "さあ始めましょう"}
}

//HTMLページを表示させる
func render(writer http.ResponseWriter, pg Page){
    t, _ := template.ParseFiles("form.html")
    t.Execute(writer, pg) // 構造体を引数に渡す
}

//URLget_stringで呼ばれる関数
func get_string(writer http.ResponseWriter, req *http.Request){
    pg := makepage("get_string") // 関数ごとに Page のインスタンスを作る
    render(writer, pg)
}

func main(){

    http.HandleFunc("/get_string", get_string)
    http.ListenAndServe(":8090", nil)
}
```

次ターミナル上でフォルダ「webapp」に移動し、おなじみのビルド・実行コマンドをリストA-3のように打ちます。

リストA-3 webapp フォルダに移動してコマンドを打つ

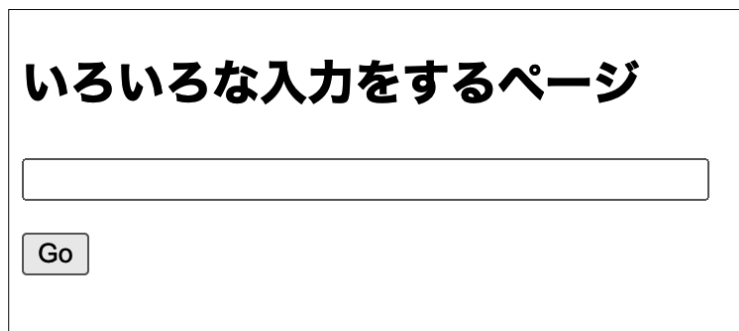
```
go build webapp.go
./webapp
```

Windowsでは、「webapp.exe」からネットワークへアクセスする許可を求めるウィンドウが出るので、プライベートネットワークで許可してください。

ブラウザで、以下のURL「get_string」にアクセスします。

http://127.0.0.1:8090/get_string

図A-2のようにブラウザ上にフォームが表示されたら大成功です。



図A-2 Goで書いたWebサーバでフォームの形が表示された

■フォームを送信できるようにする

●HTML ファイルに変数を渡す

「form.html」に記号を書いて、関数「get_string」から変数を渡すことができます。
ほかのWebフレームワークにもよくある「テンプレート」の「プレースホルダ」です。

このとき、構造体「Page」のインスタンスを作成した効果が効いてきます。
「form.html」をリスト A-4 のようにするのです。
「form.html」はこれで完成です。

リスト A-4 「form.html」を完成

```
<!DOCTYPE html>
<html>
  <h2>いろいろな入力をするページ</h2>
  <form action="{{.Name}}" method="POST">
    <p><input name="input" size="50"></p>
    <p><input type="submit" value="Go"></p>
  </form>
  <p><b>{{.Answer}}</b></p>
</html>
```

リスト A-4 の {{.Name}} および {{.Answer}} の「ドット」は、リスト A-3 で関数「get_string」の中で作成した構造体「Page」のインスタンス「pg」のフィールド Name および Answer の値です。

この状態であれば、「form.html」の保存だけで変更を反映できます。
先ほどの URL「get_string」にアクセスしてみましょう。

図 A-3 のように「さあ始めましょう」と表示されたら、関数「get_string」から値を渡せたことになります。しかし、まだ送信はできません。

いろいろな入力をするページ

さあ始めましょう

図 A-3 関数 get_string から値を渡せた

●関数「get_string」でPOST 命令を受けとる

そのために、関数「get_string」をリスト A-5 のように完成させます。

```
func get_string(writer http.ResponseWriter, req *http.Request){
    pg := makepage("get_string")

    if req.Method == "POST" {
        gs := req.FormValue("input")
        pg.Answer = gs // 送信された値をそのまま返す
    }

    render(writer, pg)
}
```

テキストフィールドに何か文字列を入力して、「Go」ボタンを押します。
図A-4のように、テキストフィールドに入力した値が下に表示されます。
 それで、この関数およびURLの名前を「get_string」にしたわけです。

A screenshot of a web browser window. The address bar shows a URL starting with 'http://'. Below the address bar, the text 'いろいろな入力をするページ' is displayed in a large, bold, black font. Underneath this text is a search bar containing the text 'GoでWebフォーム'. Below the search bar is a button labeled 'Go'.

図 A-4 テキストフィールドから送信した値を下に表示できた

■整数の計算をさせるには

●受け取った文字列を整数に変換する

Web フォームから送信する値は、常に「文字列」です。

そのため、フォームからたとえば「34」と入力した場合、文字列である「34」を「整数 34」に変換してから計算して、結果を再び文字列にして表示します。

文字列を整数に変換する方法は、第4章でプログラムを作って使っているの、ここでは同じ方法で、もっと簡単に書きます。

「webapp.go」に以下のように追記していきます。

まず、必要なパッケージを追加インポートします。(リスト A-6)

リスト A-6 「webapp.go」で必要なパッケージのインポート

```
import (  
    "net/http"  
    "html/template"  
    "math"  
    "fmt"  
)
```

文字列を整数に変換する関数「string2int」を、「webapp.go」中に書いてしまいます。
同じプログラムの中なので小文字の関数名を使います。(リスト A-7)

リスト A-7 webapp.go の中に関数「string2int」

```
// 文字列を受け取って数字に変更する  
func string2int(content string)int{  
  
    memory := []int{}  
  
    // 数字であることを確かめて、スライスmemoryの要素にする  
    digits:="0123456789"  
    for _, v := range content{  
        for i, s := range digits{  
            if v == s{  
                memory =append(memory, i)  
            }  
        }  
    }  
  
    //memoryの要素から整数を作成  
    sum:=0  
    lm :=len(memory)  
    for i := 0; i<lm; i++ {  
        mag := math.Pow10(lm-i)  
        sum +=memory[i]*int(mag)  
    }  
    return sum/10  
}
```

[A-1] HTML ファイルを表示

最後に、「form.html」をテンプレートに用いつつ、送信されてきた文字列を数字に変換し、2を足して返す関数「add2」を作成します。

この関数はURL「add2」で呼ばれるようにします。(リストA-8)

リストA-8 関数「add2」

```
//URLadd2 で呼ばれる関数
func add2(writer http.ResponseWriter, req *http.Request){
    pg := makepage("add2")
    if req.Method == "POST" {
        gs := req.FormValue("input")
        a2 := string2int(gs)+2 // 整数に変換して2を足す
        pg.Answer=fmt.Sprintf("%s+2=%d", gs, a2)

    }
    render(writer, pg)
}
func main(){

    http.HandleFunc("/get_string", get_string)
    http.HandleFunc("/add2", add2) // 追加
    http.ListenAndServe(":8090", nil)
}
```

ファイル「webapp.go」を保存してから再ビルドし、「webapp.exe」を実行します。
ブラウザで、以下のURL「add2」にアクセスしてみましょう。

http://127.0.0.1:8090/add2

図A-3とまったく同じフォームが表示されますが、数字を入れて送信すると計算結果が表示されます。(図A-5)

いろいろな入力をするページ

さあ始めましょう

いろいろな入力をするページ

34+2=36

図A-5 関数「get_string」と同じフォームを使って、数値計算もできた

以上、もっとも簡単な方法ではありましたが、GoでWebフォームを送信できました。
本文で紹介しているいろいろなテクニックも取り入れて、華麗なWebアプリケーション形式で楽しく勉強してはいかがでしょうか。

ただし、あくまで自分のPCの中で学習用として使用してください。