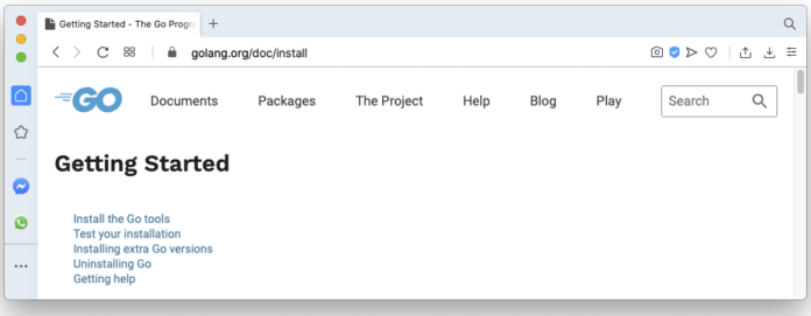
2020年5月17日 投稿者: YSATO@DELEGATE.ORG

Go 言語で Go

これからはプログラムはGoで書こう、そう思ってはいたのですが、環境整備に手間をとられてHello Worldやっただけで止まっていました。DeleGateサーバの経費節減計画も一段落、delegate.orgも無事復活できて和んでいたところ、Goの事を思い出したので、本日決行です。



インストール

まずはインストールですが、Goのサイトからパッケージをダウンロードしてインストール。クリックするだけ超絶簡単。ただ、コマンドラインからgoと打っても、無いといわれます。

```
% which go
go not found
% echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
```

それでTerminalを起動し直してみると、自動的にパスが追加されて通りました。ドキュメントを見たらそうしろと書いてありました。

```
% which go
/usr/local/go/bin/go
% echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/go/bin
```

うーん、このパスの追加はどういう仕掛けでできてるんでしょうか？すごく興味がありますが、それは置いておき。

性能測定

それでさっそくHello Worldですが（笑）、実行がかなりもっさりしています。

```
% time go run hello.go
Hello World!!
go run hello.go  0.16s user 0.10s system 113% cpu 0.230 total
```

0.2秒？インタープリタが重いんでしょうか？コンパイルしてみます。

```
% time go build hello.go
go build hello.go  0.06s user 0.07s system 141% cpu 0.095 total
% time ./hello
Hello World!!
./hello  0.00s user 0.00s system 77% cpu 0.004 total
```

4ミリ秒と出ました。妥当な時間です。

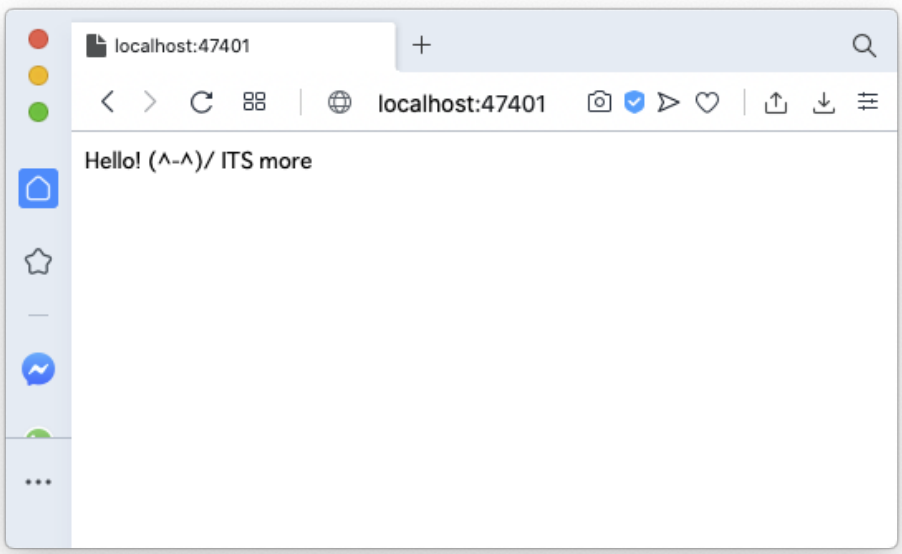
XcodeでGo

と思ったら、GoがXcodeのプロジェクトの選択肢にありません。はて？とりあえずスキップ。

GoでHTTPサーバ

とりあえずやってみたいのは HTTP サーバです。こんな感じでどうでしょう？

```
// 2020-0517-01 Orange Pekoe HTTP Server v0.0.1 by ITS more :-)
package main
import ( "net" )
func main() {
        port, _ := net.ResolveTCPAddr("tcp", ":47401" )
        sock, _ := net.ListenTCP("tcp", port)
        for {
                conn, _ := sock.Accept()
                go serv(conn)
        }
}
func serv(conn net.Conn) {
        req := make([]byte, 1024)
        conn.Read(req)
        res := "HTTP/1.0 200 Ok\r\n\r\nHello! (^-^)/ ITS more"
        conn.Write([]byte(res))
        conn.Close()
}
```

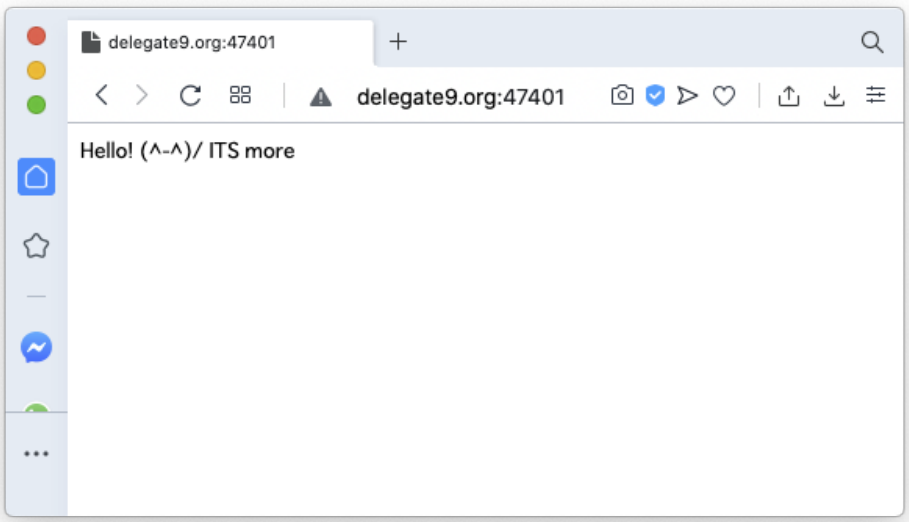


Orange Pekoe HTTP Server v0.0.1

動きました。簡単すぎてけしからんですね（笑）

サーバ公開

さっそく公開しましょう（^-^）。delegate9.org に持って行って実行です。Ubuntu で sudo apt install golang-go。しかるのちに go run pekoe.go。Azure の Networking で 47401 を開通させて、アクセスしてみます。



Orange Pekoe HTTP Server working on delegate9.org

問題ないですね。昔の事を思うと、簡単過ぎて悲しくなります。

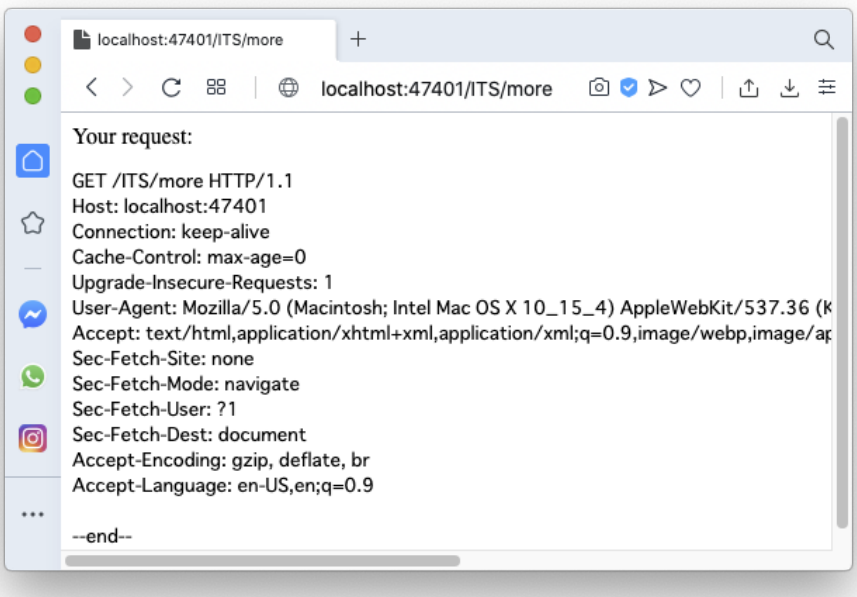
エコーサーバ

このままサクサク進むと思っていたところ、落とし穴に落ちました。最近の？ブラウザって、text/plain を自分では表示しないのがデフォなんですね。Go言語をまるで知らないので、何かの書き間違いなのだろうかと苦しみました。とりあえず text/html にして <plaintext> を突っ込みました。

```
// 2020-0517-02 Orange Pekoe HTTP Server v0.0.2 by ITS more :-)
package main
```

```
import ( "net" )
func serv(conn net.Conn) {
        qbuf := make([]byte, 1024)
        qlen, _ := conn.Read(qbuf)
        qstr := string(qbuf[:qlen])

        resh := "HTTP/1.0 200 Ok\r\n"
        resh += "Content-Type: text/html\r\n"
        resh += "\r\n"
        resh += "Your request:<plaintext>"
        conn.Write([]byte(resh))
        conn.Write([]byte(qstr))
        conn.Write([]byte("--end--\r\n"))
        conn.Close()
}
func main() {
        port, _ := net.ResolveTCPAddr("tcp", ":47401" )
        sock, _ := net.ListenTCP("tcp", port)
        for {
                conn, _ := sock.Accept()
                go serv(conn)
        }
}
```



リクエストのエコーできました。

やれやれ。

プログラムを送り込んで実行する

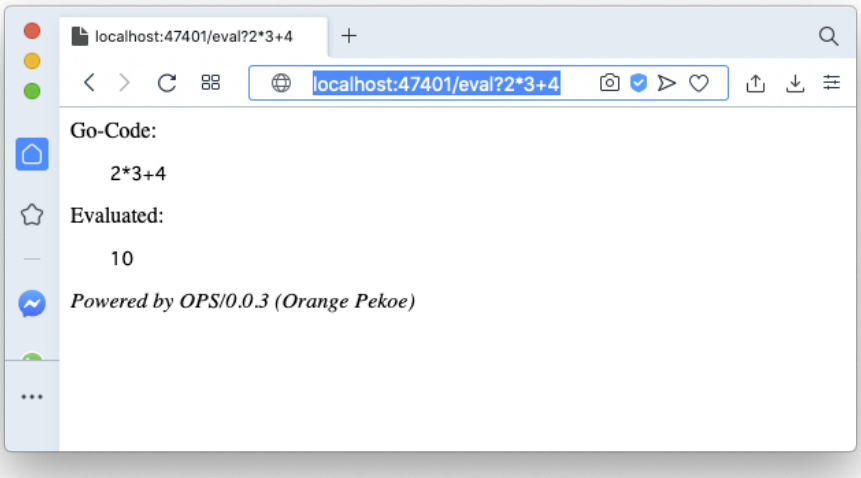
これ、この危険な遊びは、一番やってみたかった事です。Goのデモサイトでもやってますし。たぶん eval とか出来るはずです。…　…　やはり Eval() でした。

```
// 2020-0517-03 Orange Pekoe HTTP Server v0.0.3 by ITS more :-)
package main
import (
        "net"
        "fmt"
        "strings"
        "strconv"
        "go/token"
        "go/types"
)
func myid()(string){
        return "OPS/0.0.3 (Orange Pekoe)"
}
func rhdr(rcode int, ctype string, rmsg string)(string){
        return strings.Join([]string{
                "HTTP/1.0 " + strconv.Itoa(rcode) + " "+rmsg,
                "Content-Type: " + ctype,
                "Server: " + myid(),
        "",""},"\r\n")
}
func resp(conn net.Conn, str string){
        conn.Write([]byte(str))
}
func eval(conn net.Conn, xgocode string){
        gocode := udec(xgocode)
        rbuf := make([]byte, 1024)
        rstr := string(rbuf)
        fset := token.NewFileSet()
        rval, _ := types.Eval(fset,nil,token.NoPos,gocode)
        rstr = fmt.Sprintf("%d",rval.Value)
        rmsg := rhdr(200,"text/html","OK")
```

```
        rmsg += "Go-Code:<pre>\t" + henc(gocode) + "</pre>"
        rmsg += "Evaluated:<pre>\t"
        resp(conn,rmsg + henc(rstr) + "</pre>\r\n")
        resp(conn,"<i>Powered by "+myid()+"</i>\r\n")
}
func echo(conn net.Conn, qstr string){
        rmsg := rhdr(200,"text/html","OK")
        resp(conn,rmsg)
        resp(conn,"<HR>Your Request<HR><pre>")
        resp(conn,henc(qstr) + "</pre>\r\n")
        resp(conn,"<HR>My Response<HR><pre>")
        resp(conn,henc(rmsg) + "</pre>\r\n")
        resp(conn,"<HR>\r\n")
}
func henc(str string)(string){
        return strings.Replace(str,"<","<",-1)
}
func udec(str string)(string){
        return strings.Replace(str,"%3C","<",-1)
}
func serv(conn net.Conn) {
        qbuf := make([]byte, 1024)
        qlen, _ := conn.Read(qbuf)
        qstr := string(qbuf[:qlen])
        qarg := strings.Split(qstr," ")
        url := qarg[1]

        if( strings.HasPrefix(url,"/eval?") ){
                eval(conn,strings.Split(url,"?")[1])
        }else{
                echo(conn,qstr)
        }
        conn.Close()
}
func main() {
        port, _ := net.ResolveTCPAddr("tcp", ":47401" )
        sock, _ := net.ListenTCP("tcp", port)
        for {
                conn, _ := sock.Accept()
                go serv(conn)
```

```
        }
}
```



Remote calc. 🙂

動いた。

オブジェクトのサイズは

Go言語も少し覚えたし、めでたしめでたしです。というか、C言語ユーザとしては、全く違和感の無い言語ですね。それで、バイナリのサイズはどのくらいでしょうか？

```
% go build pekoe.go
% ls -l pekoe
-rwx------  1 sato  staff  4601840 May 17 17:17 pekoe
% size pekoe
__TEXT  __DATA  __OBJC  others          dec
3256320 283720  0       17012720        20552760
```

おっとびっくり！DeleGateより少しでかい…　ネットワーク系がでかいんでしょうか？ちなみに Hello World は…

```
% ls -l hello
-rwx------  1 sato  staff  2169960 May 17 17:24 hello
```

```
% size hello
__TEXT  __DATA  __OBJC  others          dec
1462272 266632  0       16895080        18623984
% nm -n pekoe | more
...
0000000001001000 t _go.buildid ## seems offset 16MB
```

それだけでもこれですか…　Evalを使ってたらデカくなるだろうとは思いますが。いったい何が入っているのでしょう？

```
% nm -n hello
X-D
```

組み込み系コンパイラなら、参照しない関数はざっくり切ってくれるのですが。セキュリティ上もそのほうが良いと思われます。きっと、ビルドのオプションにあるのでしょう。

実行時のメモリ消費は

とりあえず大人しく待ち受け状態でですが、コンパイル版（ネイティブコード）の場合、ps で見るとこんな様子です：

```
     SZ    RSS    TIME CMD
4976972   1808 0:00.01 ./pekoe
```

一方、インタープリタ？版の場合：

```
     SZ    RSS TIME    CMD
5016516  14232 0:00.24 go run pekoe.go
4977996   2580 0:00.00 /var/folders/bw/8dlt3xks44727rb0y662kw_m000
```

評価

これまで何十年、ほぼC言語一筋だったプログラマから見て、Goの記法はとても自然で、ほとんど違和感を感じません。言語、ツール、ライブラリのドキュメントは

ちゃんとしてます。とても素直でさっぱりとした言語と環境だと思われ、なんとなくこんな感じかなという勘が、だいたい当たります。とても気持ちが良い。嫌な目に遭わされたWin32とか組み込み系 🙁 の世界とはえらい違いです。

私が今必要としているHTTPサーバ機能は、実際のところ、今日作ってみたこれ程度のものなので、それが100行程度で書け、好きなようにイジりまわせるというのは素晴らしいです。性能的にも問題なさそうだし。どこへ行ってもネイティブで動くし　…

もう、Goって最高！

私の歴史が今日、動きました 🙂

2020-0517 SatoxITS

Go-言語で-Go-–-株式会社-ITS-more-5 [redacted]