2020年6月24日 投稿者: SATOXITS

GoでWebAssembly

開発：extension を HTML、CSS、JavaScript だけで書くのはやはり厳しいので、やはり複雑な処理は現地の言語で書きたいと思います。

社長：適材適所で行きましょう。

基盤：WebAssemblyというやつですね。

WebAssemblyって

開発：WikiでWebAssemblyを検索…

社長：Wikipediaの定期寄付って最低が月300円みたいですね。うちには敷居が高い…

基盤：コーヒー一杯を軽んじてる感じです。私なんて学生時代はコーヒーとか高嶺の花というか天上界の人の飲み物でしたよ。今だってそれでクラウドの仮想マシンが1ヶ月動くんですが…　AWSで$3.5/monthのやつをポコポコ立ててみようと思ってるのですが。

> https://ja.wikipedia.org/wiki/WebAssembly
>
> WebAssemblyは、ウェブブラウザのクライアントサイドスクリプトとして動作するプログラミング言語（低水準言語）である。wasmとも称されており、ブラウザ上でバイナリフォーマットの形で実行可能であることを特徴とする[2]。2017年現在開発が進められており、最初の目標としてCとC++からのコンパイルをサポートすることを目指している[3]他、Rust

がバージョン1.14以降で[4]、Goがバージョン1.11以降で[5]、Kotlin/Nativeがバージョン0.4以降で[6]で対応するなど、他のプログラミング言語のサポートも進められている。

開発： Go でも使えるというのがとても魅力だと思います。面白いのがWikiの中のこの部分。

> WebAssemblyはポータブルなスタックマシン[7]であり、既存のウェブブラウザで広く用いられているJavaScriptと比べ、構文解析と実行が高速になるよう設計されている[3]。
> …
> 内部的には、wasmコンパイラシステムは中間コードを扱うためにS式を使用している

基盤：S式！40年以上タイムスリップしました。

開発：Goでのサポートは Go 1.11、 2018-08-24 のブログでアナウンスされてます。約2年前ですね。

> https://blog.golang.org/go1.11
>
> Go 1.11 also adds an experimental port to WebAssembly (`js/wasm`). This allows programmers to compile Go programs to a binary format compatible with four major web browsers. You can read more about WebAssembly (abbreviated “Wasm”) at webassembly.org and see this wiki page on how to get started with using Wasm with Go.

基盤；「four major web browser」って、Safari が仲間外れなんでしょうかね（笑）

社長：私は最近、Safari には Safari なりの良い点、面白い点もあるとは思ってるんですけどね。でも唯我独尊のApple社製ですからねえ…

> https://golang.org/doc/go1.11#wasm
>
> WebAssembly
>
> Go 1.11 adds an experimental port to WebAssembly (`js/wasm`).
>
> Go programs currently compile to one WebAssembly module that includes the Go runtime for goroutine scheduling, garbage collection, maps, etc. As a result, the

> resulting size is at minimum around 2 MB, or 500 KB compressed. Go programs can call into JavaScript using the new experimental `syscall/js` package. Binary size and interop with other languages has not yet been a priority but may be addressed in future releases.

開発：Goのバイナリがでかいというのは定番ですが、別に2M程度ならちょっとした画像程度だから、転送量としては気にならないですね。

> As a result of the addition of the new `GOOS` value "`js`" and `GOARCH` value "`wasm`", Go files named `*_js.go` or `*_wasm.go` will now be ignored by Go tools except when those GOOS/GOARCH values are being used. If you have existing filenames matching those patterns, you will need to rename them.
>
> More information can be found on the WebAssembly wiki page.

誰でも最初は Hello world

開発：でその wikipage にお待ちかねの Hello world があるので、今日はそれをやってみます。まず、使用するのはおなじみのこれ。

```
% cat > hello.go
package main
import "fmt"
func main() {
  fmt.Println("Hello, WebAssembly!")
}
```

開発：こいつを WebAssembly にコンパイルします。

> Set `GOOS=js` and `GOARCH=wasm` environment variables to compile for WebAssembly:
>
> $ GOOS=js GOARCH=wasm go build -o main.wasm

基盤：これ、ちょっと意味がわからないんですが、環境変数を設定しているって意味ですかね。

開発：そうらしいです。bash からの仕様なんですかね？

```
% XX=a printenv XX
a
```

開発：あと、go ファイルを指定しないと、カレントディレクトリの .go を勝手に探してやってくれるようです。

基盤：徹底して手間を省く気満々ですね。

開発：で、build で main.wasm が出来ました。

```
MacMini% ls -l
total 4432
-rw-r–r– 1 ysato staff 78 Jun 24 09:04 hello.go
-rwxr-xr-x 1 ysato staff 2262220 Jun 24 09:17 main.wasm
```

基盤：噂通り2MB超 🙂

開発：あと、パッケージは main パッケージ一つにしてね、とあります。

> Note that you can only compile main packages. Otherwise, you will get an object file that cannot be run in WebAssembly. If you have a package that you want to be able to use with WebAssembly, convert it to a main package and build a binary.

基盤：それって go build で自動的には出来ないんですかね。ただ面倒だったのかな。

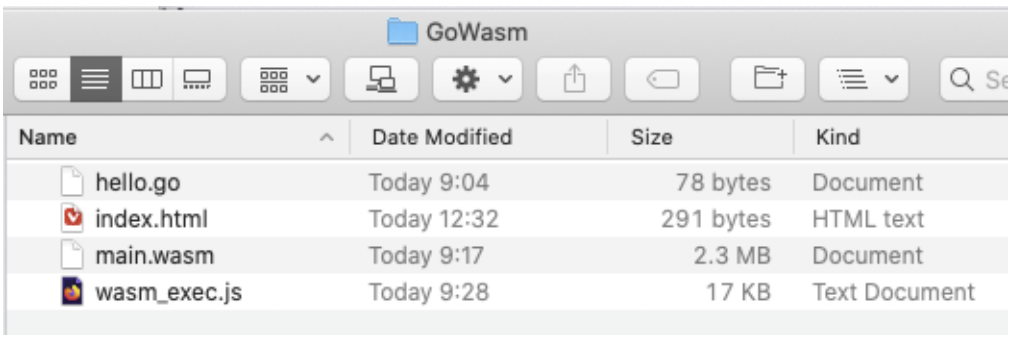
開発：さらにチュートリアルに言われるがままに。

```
% cp "$(go env GOROOT)/misc/wasm/wasm_exec.js" .
```

```
% cat index.html
<html>
 <head>
  <meta charset="utf-8"/>
  <script src="wasm_exec.js"></script>
  <script>
   const go = new Go();
   WebAssembly.instantiateStreaming(
    fetch("main.wasm"),go.importObject).
     then( (result) => { go.run(result.instance); } );
  </script>
 </head>
 <body></body>
</html>
```

基盤：よくわからないけど、2つ目のスクリプトはmain.wasm をフェッチして実行する、ということのようですね。

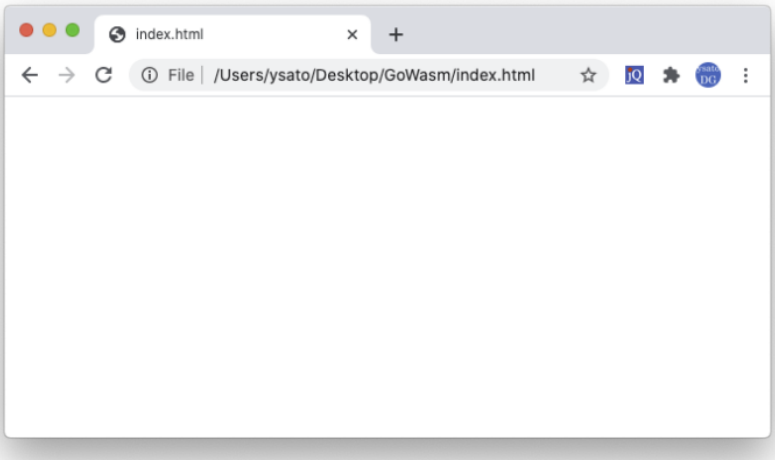
開発：準備OK。こういう事になりました。



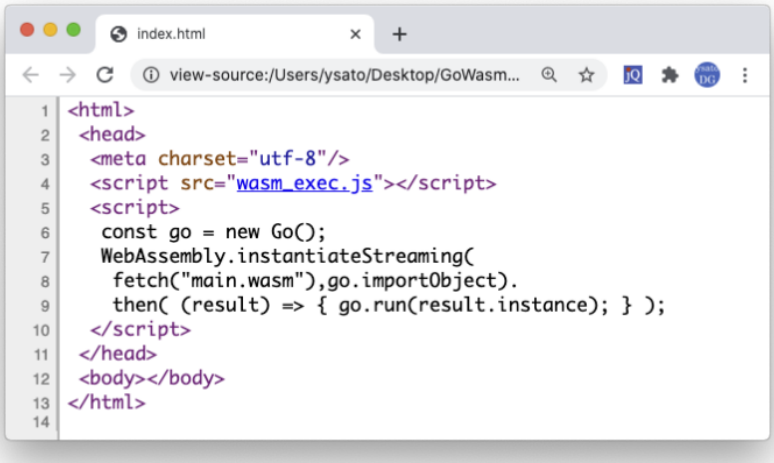
FILE URL は通らなかった

開発：で、この index.html をクリックすると…

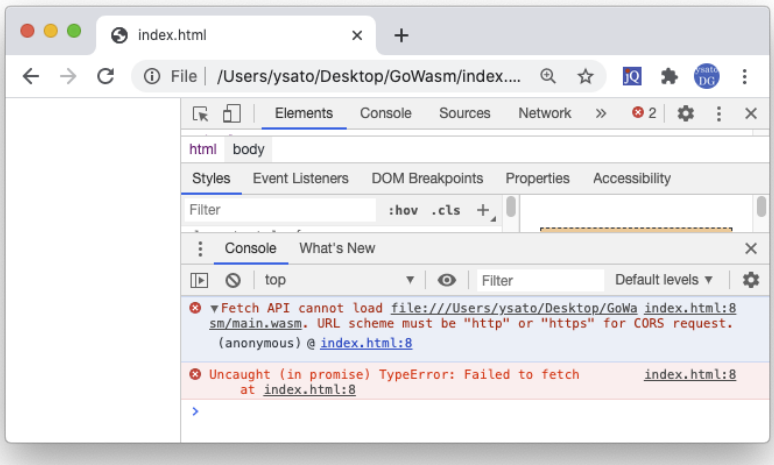
基盤：真っ白ですね。



開発：HTMLをコピペ間違えたかな？



基盤：問題なさそうな。インスペクタでは？



開発：え？

> Fetch API cannot load file:///Users/ysato/Desktop/GoWasm/main.wasm. URL scheme must be “http” or “https” for CORS request.

基盤：あれま。

開発：なんで file scheme じゃいかんのですかね？HTTPサーバって言ってもただファイルを提供してるだけだろうし、HTTPよりは直接見えるFILEのほうが素性が知れてるし高速だと思うのですが…　自分自身より遠くの他人を信用するってどういう事だろう。ちょっとがっかり。

基盤：サーバ名の部分を何かに使っているとか？

開発：ブラウザ自身にHTTPサーバのコードを含んでいてforkするだけでいいんじゃないかなぁ。

DeleGateは通らなかった

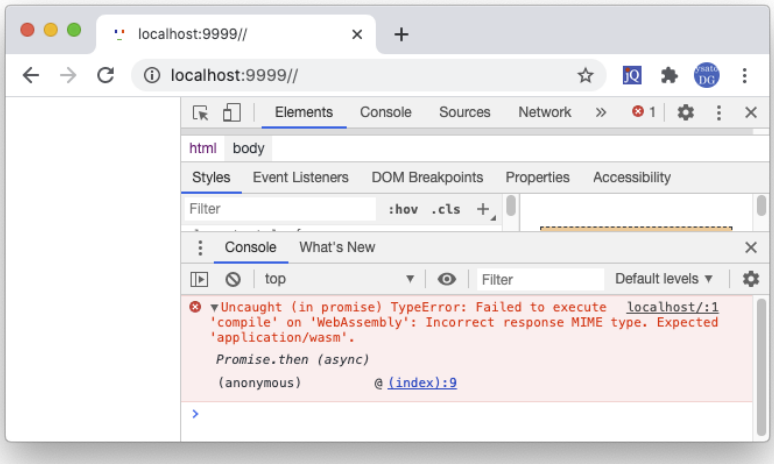
開発：気をとりなおしてサーバ経由にします。とりあえず勝手知ったる DeleGate でちゃちゃっと。

```
% delegated -fv -P9999 SERVER=http MOUNT="/* $PWD/*"
```

開発：でどうかな？

基盤：ふたたび真っ白ですね。

開発：何がご不満なのでしょうか？

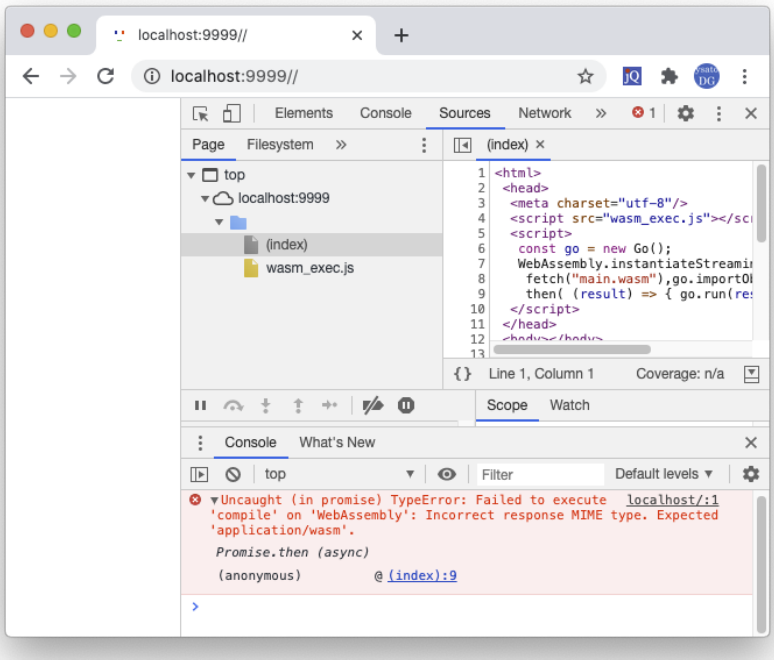


基盤：ああ、MIME タイプ をapplication/wasm にしろと。

開発：それじゃDeleGateにファイル型を教えましょう。真ん中あたりは Gopher 用の定義だったりする年代物。

```
FILETYPE=".wasm:9:BIN:binary:application/wasm"
```

開発：これでどうかな？



開発：え？手でアクセスしてもこう返ってきてるのに…

```
HTTP/1.0 200 OK
Date: Wed, 24 Jun 2020 04:24:23 GMT
Server: DeleGate/11.0.2
DeleGate-Ver: 11.0.2 (delay=0)
MIME-Version: 1.0
Content-Type: application/wasm
Content-Length: 2262220
Last-Modified: Wed, 24 Jun 2020 00:17:07 GMT
```

基盤：ここはとりあえずチュートリアルに従いましょう。

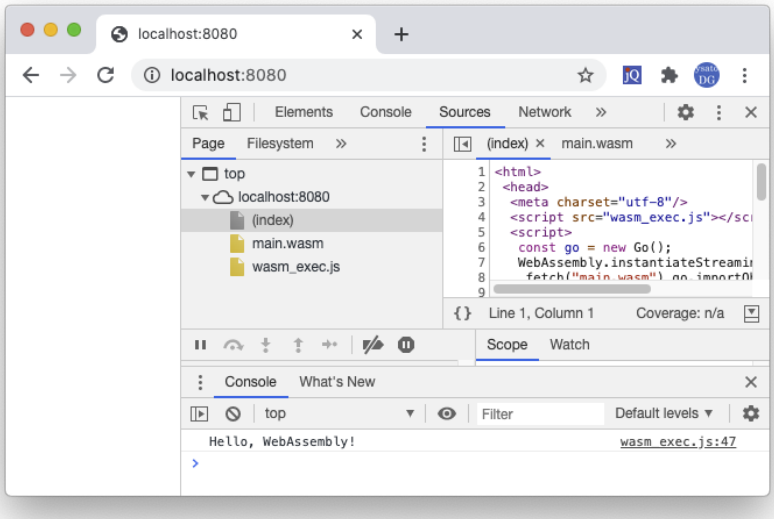
祝Hello World!

開発：仰せのとおり goexec でと…

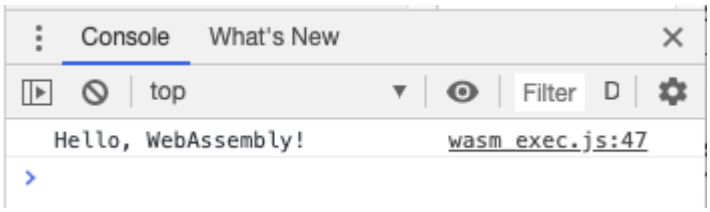
```
# install goexec: go get -u github.com/shurcooL/goexec
$ goexec 'http.ListenAndServe(`:8080`, http.FileServer(http.Dir(`
```

基盤：go get 帰ってきませんね。なんかプログレス報告がほしいところです。あ、終わった。数分かかったような。

開発：goexec っと。あれ？PATHが通ってないですね。export PATH=$PATH:$HOME/go/bin。これでいかがでしょうか？



基盤：通りましたね。



基盤：GoのHTTPは何を返してるんでしょうね？

```
HTTP/1.0 200 OK
Accept-Ranges: bytes
Content-Length: 2262220
Content-Type: application/wasm
Last-Modified: Wed, 24 Jun 2020 00:17:07 GMT
Date: Wed, 24 Jun 2020 04:48:41 GMT
```

開発：私は違いがわからない男。

基盤：Accept-Ranges の有無で引っかかってるとか。

開発：まあ、あとで調べます。ちょっとシラケちゃいました。ブレークしましょう。

＊　＊　＊

HomeBrewで道草

基盤：引き続き Node.js のススメがついてますが。

開発：いやはやこれだけ簡潔なチュートリアルで、おまけまでついちゃう。Goのセンスは素晴らしいです。どれどれ…

```
MacMini% GOOS=js GOARCH=wasm go run -exec="$(go env
GOROOT)/misc/wasm/go_js_wasm_exec" .
/usr/local/go/misc/wasm/go_js_wasm_exec: line 14: exec: node: not found
exit status 127
```

基盤：Node がインストールされてないってことですかね。macos node install で検索… Homebrew というのでインストールするようです。apt みたいなやつですかね。インストール方法…

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Home
```

開発：curl は入ってるからこのままでイケるしょう。コピペしてGo！

```
MacMini% /bin/sh -c "$(curl -fsSL https://raw.githubusercontent.com/Homebr
ew/install/master/install.sh)"
Password:
Sorry, try again.
Password:
Sorry, try again.
Password:
==> This script will install:
/usr/local/bin/brew
/usr/local/share/doc/homebrew
/usr/local/share/man/man1/brew.1
/usr/local/share/zsh/site-functions/_brew
/usr/local/etc/bash_completion.d/brew
/usr/local/Homebrew
==> The following existing directories will be made group writable:
/usr/local/bin
/usr/local/include
/usr/local/lib
/usr/local/share
/usr/local/lib/pkgconfig
/usr/local/share/man
/usr/local/share/man/man1
==> The following existing directories will have their owner set to ysato:
/usr/local/bin
/usr/local/include
/usr/local/lib
/usr/local/share
/usr/local/lib/pkgconfig
/usr/local/share/man
/usr/local/share/man/man1
==> The following existing directories will have their group set to admin:
/usr/local/bin
/usr/local/include
/usr/local/lib
/usr/local/lib/pkgconfig
==> The following new directories will be created:
/usr/local/etc
/usr/local/sbin
/usr/local/var
/usr/local/opt
/usr/local/share/zsh
/usr/local/share/zsh/site-functions
/usr/local/var/homebrew
/usr/local/var/homebrew/linked
/usr/local/Cellar
/usr/local/Caskroom
/usr/local/Homebrew
/usr/local/Frameworks

Press RETURN to continue or any other key to abort
```

開発：うーん、なんで$HOMEの下に作ってくれないのかな。

基盤：/usr/local の下だし、いいんじゃないですか？

開発：うーん、おそらくcurlの出力スクリプトを変えればイケるのでは… grep local。

```
# On macOS, this script installs to /usr/local only.
  HOMEBREW_PREFIX="/usr/local"
  HOMEBREW_REPOSITORY="/usr/local/Homebrew"
```

開発：ほらね。

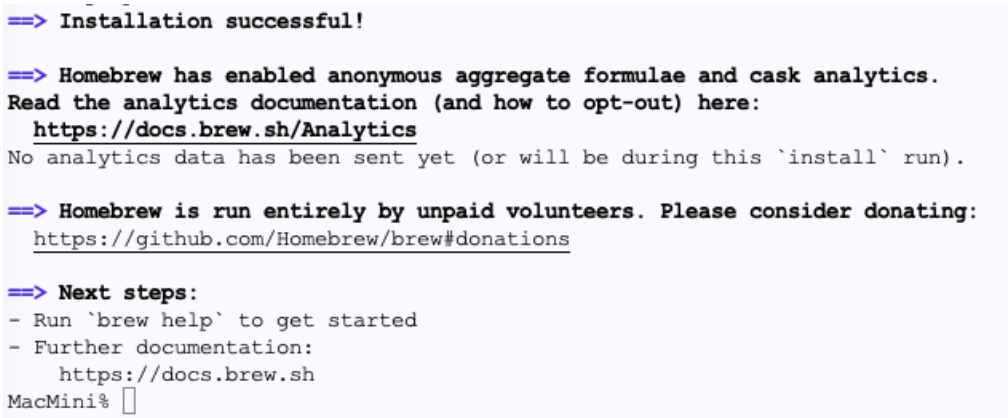
基盤：どういうスクリプトなんでしょう。

```
# On macOS, this script installs to /usr/local only.
# On Linux, it installs to /home/linuxbrew/.linuxbrew if you have sudo access
# and ~/.linuxbrew otherwise.
# To install elsewhere (which is unsupported)
# you can untar https://github.com/Homebrew/brew/tarball/master
# anywhere you like.
```

基盤：Macでは選択不可、Linuxでもunsuppotedですって。

開発：そうですか。/usr/local を飲みます。毒まんじゅうでもあるまい。RETURN をぽち。

基盤：なんかすごいやってますが…　あ、終わった。数分がかりでしたね。



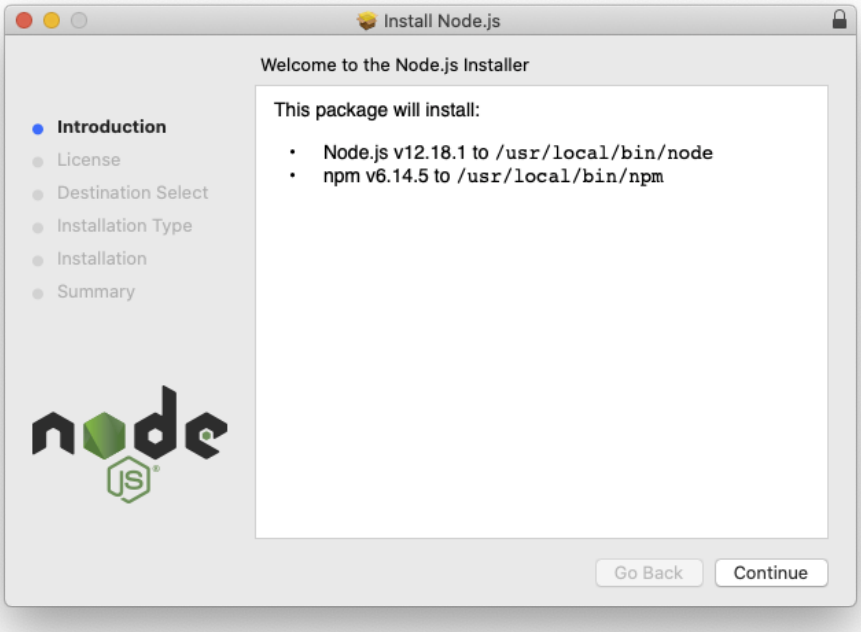
開発：では再チャレンジ！あれ？ああ、brew をインストールしただけで、node はまだだった（笑）。では、brew install nodebrew…

```
MacMini% brew install nodebrew
==> Downloading https://github.com/hokaccha/nodebrew/archive/v1.0.1.tar.gz
==> Downloading from https://codeload.github.com/hokaccha/nodebrew/tar.gz/v1.0.1
######################################################################## 100.0%
==> Caveats
You need to manually run setup_dirs to create directories required by nodebrew:
  /usr/local/opt/nodebrew/bin/nodebrew setup_dirs

Add path:
  export PATH=$HOME/.nodebrew/current/bin:$PATH

To use Homebrew's directories rather than ~/.nodebrew add to your profile:
  export NODEBREW_ROOT=/usr/local/var/nodebrew

Bash completion has been installed to:
  /usr/local/etc/bash_completion.d

zsh completions have been installed to:
  /usr/local/share/zsh/site-functions
==> Summary
🍺  /usr/local/Cellar/nodebrew/1.0.1: 8 files, 38.6KB, built in 5 seconds
```

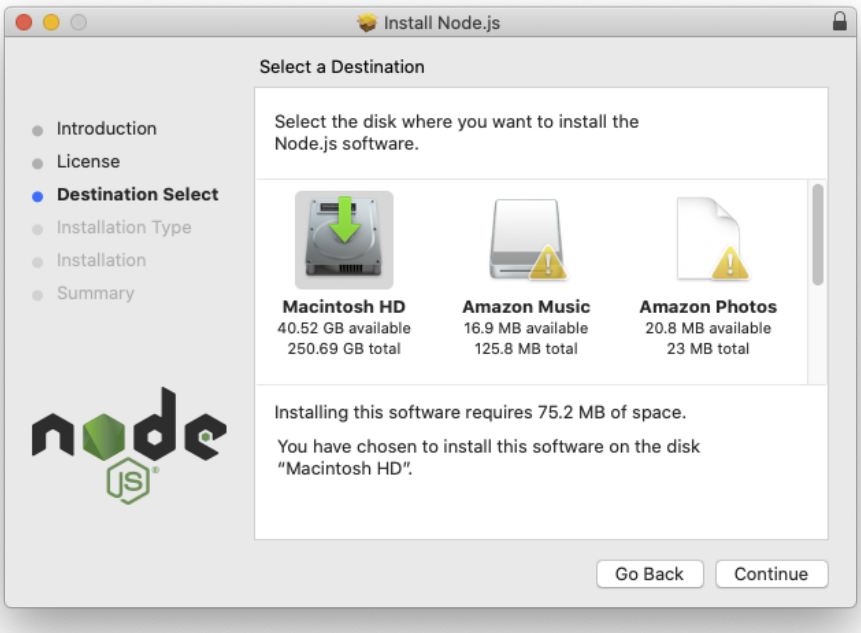
開発：何だコリア？

基盤：Node.jsのサイトに行ってみましょう。

開発：なんだ、パッケージあるんじゃない。じゃ、推奨のLTSをば。ぷち。ぷちっと。

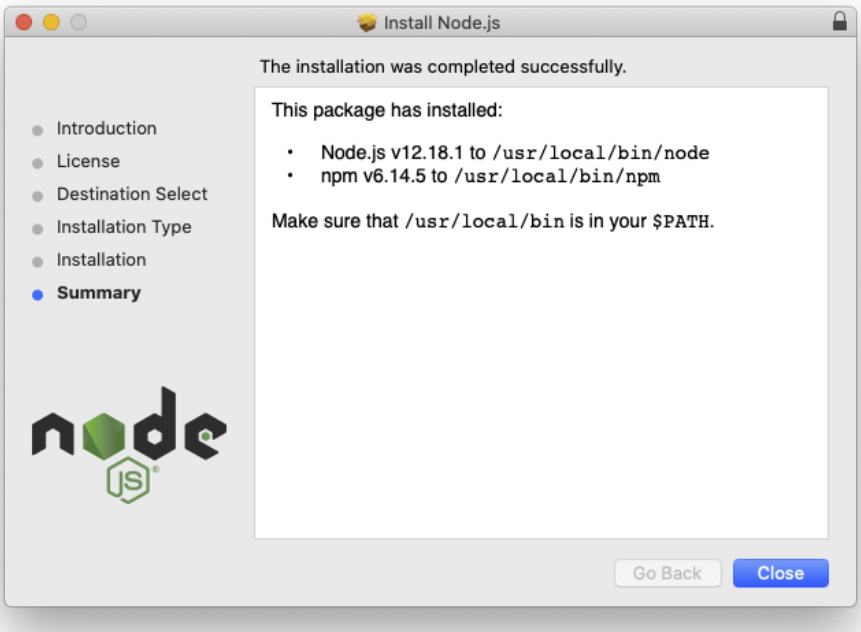


開発：おーいえーす、あいアグリー。



基盤：これ、アマゾンドライブにインストールしたら面白そうですね（笑）

開発：はい終了。



開発：で一応確認。

```
MacMini% which node
/usr/local/bin/node
```

基盤：OK。

Node でWebAssmblyを Go

開発：ではおもむろに再チャレンジ…

```
$ time GOOS=js GOARCH=wasm go run -exec="$(go env
GOROOT)/misc/wasm/go_js_wasm_exec" .
Hello, WebAssembly!

real 0m0.656s
user 0m1.650s
sys 0m0.159s
```

基盤：くっそ重いですね（笑）

開発：real より CPU が多いってどういうこと？ああ、マルチコアでのユーザ時間の和かな。

```
MacMini% time GOOS=js GOARCH=wasm go run -exec="$(go env
GOROOT)/misc/wasm/go_js_wasm_exec" .
Hello, WebAssembly!
GOOS=js GOARCH=wasm go run -exec="$(go env
GOROOT)/misc/wasm/go_js_wasm_exec"
1.66s user 0.31s system 189% cpu 1.036 total
MacMini%
```

基盤：そのようです。

開発：まあ、ブラウザ無し、サーバ無しでテストできるんで、多少遅くてもOKじゃないですかね。

基盤：それにしても、本題で無いところにばかり時間を取られますね。

開発：最初だからしかたが無いです。ブレークしましょう。

＊　＊　＊

Goでシステムコール

開発：されそれで、本題の SandBox で何ができるかです。

基盤：システムコールとかみんなトラップされてるんですかね？

開発：まずは uptime とか top 的な事したいです。

基盤：あれ？そういえば今日午前中にICカードのリーダライタが納入されたはずなのですが、携帯にぶぶっと来ないですね。

開発：ちょっと熱中してたし…

基盤：ちょっと郵便受け見てきます。・・・　来てました。JPだからメールでお知らせがなかったんですね。配送オプションで指定できたりするのかな？それはそうと、肝心の非接触リーダ・ライタが入ってません。

経理：アマゾンで確認します。・・・　ああ、それだけ6/26日に別便で来ます。

基盤：なーんだ。

開発：それで、Go自体はどんな system callもできる道有ですね。uptimeのライブラリとかないかな。どうも、探し方がわからない。あ、gopsutil というのの、package host というのに Uptime() という関数がある模様。

```
func Uptime() (uint64, error)
```

基盤：これ、Unixtime の64ビット版ですかね？

開発：じゃないですかね。ああ、Uptime() じゃないけどここに gopsutil の使い方の例が。

```
MacMini% cat mem.go
package main
import (
“fmt”
“github.com/shirou/gopsutil/mem”
)
func main() {
v, _ := mem.VirtualMemory()
// almost every return value is a struct
fmt.Printf(“Total: %v, Free:%v, UsedPercent:%f%%\n”, v.Total, v.Free, v.UsedPercent)
// convert to JSON. String() is also implemented
fmt.Println(v)
}
```

開発：それでは go run …

```
MacMini% go run mem.go
mem.go:6:8: cannot find package “github.com/shirou/gopsutil/mem” in any of:
```

```
/usr/local/go/src/github.com/shirou/gopsutil/mem (from $GOROOT)
/Users/ysato/go/src/github.com/shirou/gopsutil/mem (from $GOPATH)
```

開発：パッケージが無い言ってます。まあそりゃそうか。ダウンロードすれそこに置けばいいのかな。でも面倒くさい…　きっと、apt-get 的なものがあるに違いないと思うのですが。

基盤：それ、前やったことありますけど。忘れました。

＊　＊　＊

基盤：これでした。

go get *PackageName*

開発：やれやれですね。では go get github.com/shirou/gopsutil/mem … 出来ました。そして再び go run !

```
MacMini% time go run mem.go
Total: 8589934592, Free:1008758784, UsedPercent:71.788216%
go run mem.go 0.33s user 0.19s system 76% cpu 0.673 total
```

基盤：出ました、が、0.67秒、くっそ重い。

開発：コンパイルしましょう。

```
MacMini% time go build mem.go
go build mem.go 0.09s user 0.21s system 83% cpu 0.359 total
MacMini% time ./mem
Total: 8589934592, Free:1101942784, UsedPercent:64.464474%
./mem 0.00s user 0.00s system 7% cpu 0.053 total
```

基盤：0.05秒、合格です。

開発：そうか、8GiBって、8*1024^3 = 8589934592 なんですね。ちょっと得したような気分。

基盤：準備完了ですね。

開発：長い道のりでした。ちょっと一服しましょう。

＊　＊　＊

社長：我社もコーヒーメーカーという文明の利器を導入しましょうかね？

開発：いえ、お湯を沸かしてこのドリップパックを入れるのが、ちょうど良い休憩になるように思います。

基盤：ガス台まで徒歩3秒。

社長：しかし、1パック20円って。しかも十分美味しい。一体コーヒーの適正価格ていくら位なんですかね。

開発：そういえば最近はインスタントコーヒーというのを見なくなりました。レギューのパックが安くなったからでしょうか？

社長：インスタントに作れるというメリットはまだあると思いますけどね。

＊　＊　＊

禁じられた遊び

開発：さて、せっかくなので本番前に Node.js でリハーサルです。

基盤：引っ張りますね。

```
MacMini% cat mem.go
package main
import (
“fmt”
“github.com/shirou/gopsutil/mem”
```

```
)
func main() {
fmt.Printf("—- mem.go —-\n")
v, _ := mem.VirtualMemory()
fmt.Printf("Total: %v, Free:%v\n",
v.Total, v.Free)
}
```

開発：これを Wasm にして。

```
MacMini% GOOS=js GOARCH=wasm go build -o main.wasm
```

開発：node で実行！

```
MacMini% GOOS=js GOARCH=wasm go run -exec="$(go env
GOROOT)/misc/wasm/go_js_wasm_exec" .
—- mem.go —-
panic: runtime error: invalid memory address or nil pointer dereference
(signal 0xb code=0x0 addr=0x0 pc=0x0]

goroutine 1 [running]:
main.main()
/Users/ysato/Desktop/GoWasm/mem.go:10 +0x6
exit status 2
```

開発：終わった（笑）

基盤：エラーコードにはなんと？

開発：プリントしましょう。

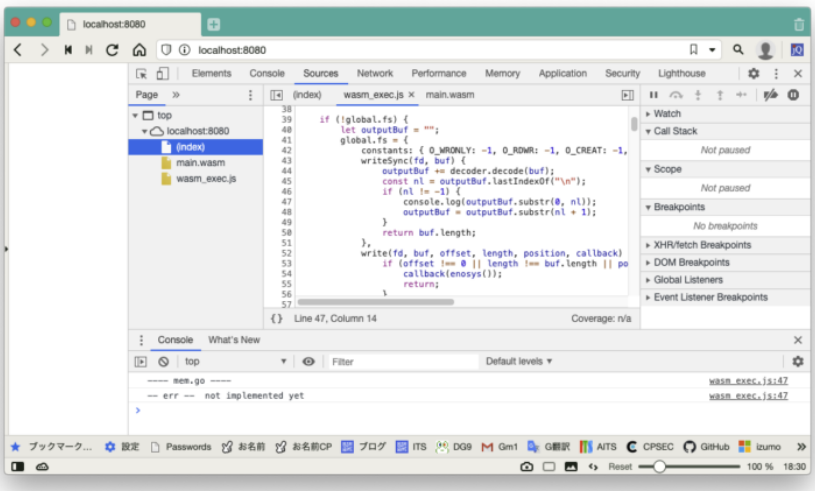
```
MacMini% cat mem.go
package main
import (
"fmt"
"github.com/shirou/gopsutil/mem"
)
func main() {
```

```
fmt.Printf("—- mem.go —-\n")
v, err := mem.VirtualMemory()
if( err != nil ){
fmt.Println("– err — ",err);
}else{
fmt.Printf("Total: %v, Free:%v\n",
v.Total, v.Free)
}
}
MacMini% GOOS=js GOARCH=wasm go build -o main.wasm
MacMini% GOOS=js GOARCH=wasm go run -exec="$(go env
GOROOT)/misc/wasm/go_js_wasm_exec" .
—- mem.go —-
— err — not implemented yet
```

開発：うーん「not implemented yet」がおざなりの回答なのか、どうかですね。コンパイルの時点ではエラーは出ないから、実行環境によっては許可されるアクセスなのかも知れない。Node.js で実装されていないという意味かも。

基盤：ほぼ絶望的な状況ですが、一縷の希望を抱きつつ本番のブラウザへ。

開発： 開け http://localhost:8080 ！



基盤：最終的かつ不可逆的に終了しました。

開発：まあ、最終的かどうかはわからないです。

開発：ただ、どの道、file URL で wasm にアクセスできない時点で、我々の進む道は絶たれていたのです。どの道ローカルにも HTTPサーバが必要なのなら、そっちでそういう処理はやれば良いとは言えます。

＊　＊　＊

携帯：（ぶるぶるぶる）

社長：はい。えっそうなの。うん、出られます。じゃ。

社長：コロナの影響で中断していたボウリングのリーグ戦ですが、先週から再開してたそうです。来週が最終戦になっちゃうとか。

開発：ええー！

基盤：われわれの唯一のフィジカル健康法が…

社長：てかもう投げ方忘れちゃったけど（笑）来週が楽しみですね。

＊　＊　＊

開発：さて楽しかったGo WebAssembly チュートリアル、一応最後まで目を通しますか。Get Going with WebAssembly … 面白そうだけど、同時通訳とか付いてるといいな。DOM、Canvas。うーん、JavaScirpt で書くのより何がうれしいんだろう？「You can use net/http library to make HTTP requests from Go」これってどこにでもつなげるってわけじゃないんだよねきっと。Further examples。やっぱグラフィックスは面白いね。Reducing the size of Wasm files…　えー、これはすごい。「At present, Go generates large Wasm files, with the smallest possible size being around ~2MB. If your Go code imports libraries, this file size can increase dramatically. 10MB+ is common.」。TinyGo「The “Hello world” example is 575 bytes. 」それはすばらしい。これはもし本気でやる時になったら要検討です。

基盤：それってwasmじゃないbuildでもそうなんですかね。だとするとすごいですけどお腹すきました。

社長：長丁場になりました。午前中に始める前に比べると、かなりこの周辺の景色がはっきりしたいと思います。飲みに行きましょう。

— 2020-0624 SatoxITS