An Exploration of Go

Starting Webassembly

Admin

- Questions?
- exam coming soon
  - Open book, open note closed neighbor more in a few minutes

Web assembly

- So most of you told me you really hadn’t done much with web assembly – so lets start from the beginning
-

Web assembly

- So most of you told me you really hadn’t done much with web assembly – so lets start from the beginning
- Once upon a time long long ago….
  - The fine people at netscape (yes **long** ago when I was young and many of you were newborns) needed a scripting language to make the web more dynamic
  -

Web assembly

- So most of you told me you really hadn’t done much with web assembly – so lets start from the beginning
- Once upon a time long long ago….
  - The fine people at netscape (yes **long** ago when I was young and many of you were not even thought of) needed a scripting language to make the web more dynamic
  - And there was javascript
  -

Web assembly

- So most of you told me you really hadn’t done much with web assembly – so lets start from the beginning
- Once upon a time long long ago….
  - The fine people at netscape (yes **long** ago when I was young and many of you were not even thought of) needed a scripting language to make the web more dynamic
  - And there was javascript
  - And devs look on it and saw that it was bad

Web assembly

- So most of you told me you really hadn’t done much with web assembly – so lets start from the beginning
- Once upon a time long long ago….
  - The fine people at netscape (yes **long** ago when I was young and many of you were not even thought of) needed a scripting language to make the web more dynamic
  - And there was javascript
  - And devs look on it and saw that it was bad
  - But it was the only choice

Web assembly

- So most of you told me you really hadn’t done much with web assembly – so lets start from the beginning
- Once upon a time long long ago….
  - The fine people at netscape (yes **long** ago when I was young and many of you were not even thought of) needed a scripting language to make the web more dynamic
  - And there was javascript
  - And devs look on it and saw that it was bad
  - But it was the only choice
  - So javascript became a big deal (And Node was born etc)

Web Assembly history

- But but javascript
  - What are some of its perceived limitations?
  - Lucky volunteer(TM)
  -

Web Assembly history

- But but javascript
  - What are some of its perceived limitations?
    - Dynamically typed
      - So popular a decade or so ago, but turns out not to scale
      - Typescript is one of the most popular ecmascript standard languages
    - Slow
      - Lets discuss “slow” in terms of modern cloud based software
    - Lucky volunteer(TM)?

Web Assembly history

- But but javascript
  - What are some of its perceived limitations?
    - Dynamically typed
      - So popular a decade or so ago, but turns out not to scale
      - Typescript is one of the most popular ecmascript standard languages
    - Slow
      - Lets discuss “slow” in terms of modern cloud based software
      - For lots of stuff this doesn’t matter – latency for the Boston-Singapore database query is higher than the interpreted language slowdown.
    - when does it matter?

Web Assembly history

- But but javascript
  - What are some of its perceived limitations?
    - Dynamically typed
      - So popular a decade or so ago, but turns out not to scale
      - Typescript is one of the most popular ecmascript standard languages
    - Slow
      - Lets discuss “slow” in terms of modern cloud based software
      - For lots of stuff this doesn’t matter – latency for the Boston-Singapore database query is higher than the interpreted language slowdown.
    - If we haven’t covered it yet – when does it matter?
      - **Games**
      - Local computation-heavy work
      - Anything else?

Web Assembly History

- So in 2013 – asm.js
  - A strict subset of javascript
  - Designed to allow statically typed languages with manual memory management to be cross compiled to a subset of the javascript vm
    - So give me an example of the target language
    - Lucky volunteer(TM)?
  - Provided really good performance.

WebAssembly

- In 2015 the powers that be decided to evolve asm.js into webassemby.
- By 2017 the MVP was out and implemented in “all major browsers”
  - Firefox, chrome, safari, and IE (later also the chrome based edge)

WebAssembly

- Web assembly is a compile target
  - Cross compile – what do we mean?
    - Lucky volunteer(TM)?
- Runs on virtual machine in browser
  - Model similar to java
-

Webassembly

- So my web assembly runs in your browser on your machine
- What are you thinking?

Webassembly

- So my web assembly runs in your browser on your machine
- What are you thinking?
- What could ***possibly*** go wrong?

Webassembly

- So my web assembly runs in your browser on your machine
- What are you thinking?
- What could ***possibly*** go wrong?
  - So hopefully you have deep worries about security
  - Webassembly incorporates lots of lessons learned from earlier technologies and prevents webassembly programs from getting any resources on the local machine
    - Get resources from program itself
    - Or internet.

Go and web assembly

- Go was not the language Mozilla had in mind
  - Mozilla created rust and thought it was an ideal web assembly based language
  - Why?
    - Lucky volunteer(TM)?
  -

Go and web assembly

- Go was not the language Mozilla had in mind
  - Mozilla created rust and thought it was an ideal web assembly based language
  - Why?
    - Lucky volunteer(TM)?
  - Manual memory management
  - When go is compiled to webassembly so is its runtime
    - So application is minimum of 2mb
    - Hows that?
    - Lucky volunteer(TM)?

Go and web assembly

- Go was not the language Mozilla had in mind
  - Mozilla created rust and thought it was an ideal web assembly based language
  - Why?
    - Lucky volunteer(TM)?
  - Manual memory management
  - When go is compiled to webassembly so is its runtime
    - So application is minimum of 2mb
    - Hows that?
    - Kinda crappy for a web commerce store – pretty ok for a 3d game

Webassembly and go

- Pure go translates to web assembly pretty easily.
  - Go with foreign dependencies is harder (qt gtk etc)
- In fact go provides us with tools to make go apps web assembly even easier
  - `In your go install, there is a misc folder, and in there is a wasm_exec.html, we'll use this as your index.html`
- Make a new folder for your running web project
  - Copy <path to go>/misc/wasm/wasm_exec.html to your project
  -

Now you need to web serve

- At this point you could just run a heavy duty webserver like ngix
  - And if we were deploying this to lots of people we would
  - But for development lets use a simple webserver that I pulled from a tutorial (reference link included)
- `See next slide`
  - `Put it into a single file, compile it to an executable`
  - `And then when we have our wasm ready we will run it`

```
package main
//https://itnext.io/webassemply-with-golang-by-scratch-e05ec5230558
import (
    "log"
    "net/http"
)

const(
    AddSrv = ":8080"
    TemplatesDir = "."
)

func main() {
    log.Printf("listening on %q...", AddSrv)
    fileSrv := http.FileServer(http.Dir(TemplatesDir))
    if err := http.ListenAndServe(AddSrv, fileSrv); err!=nil{
        log.Fatal(err)
    }
}
```

Lets have a look

- Lets try this now with the version of the project from the last class
  -

Lets have a look

- Lets try this now with the version of the project from the beginning of class
  - It doesn’t look beautiful, but it works
  - That same project that we just saw on desktop is now on the web browser with minimal extra work.

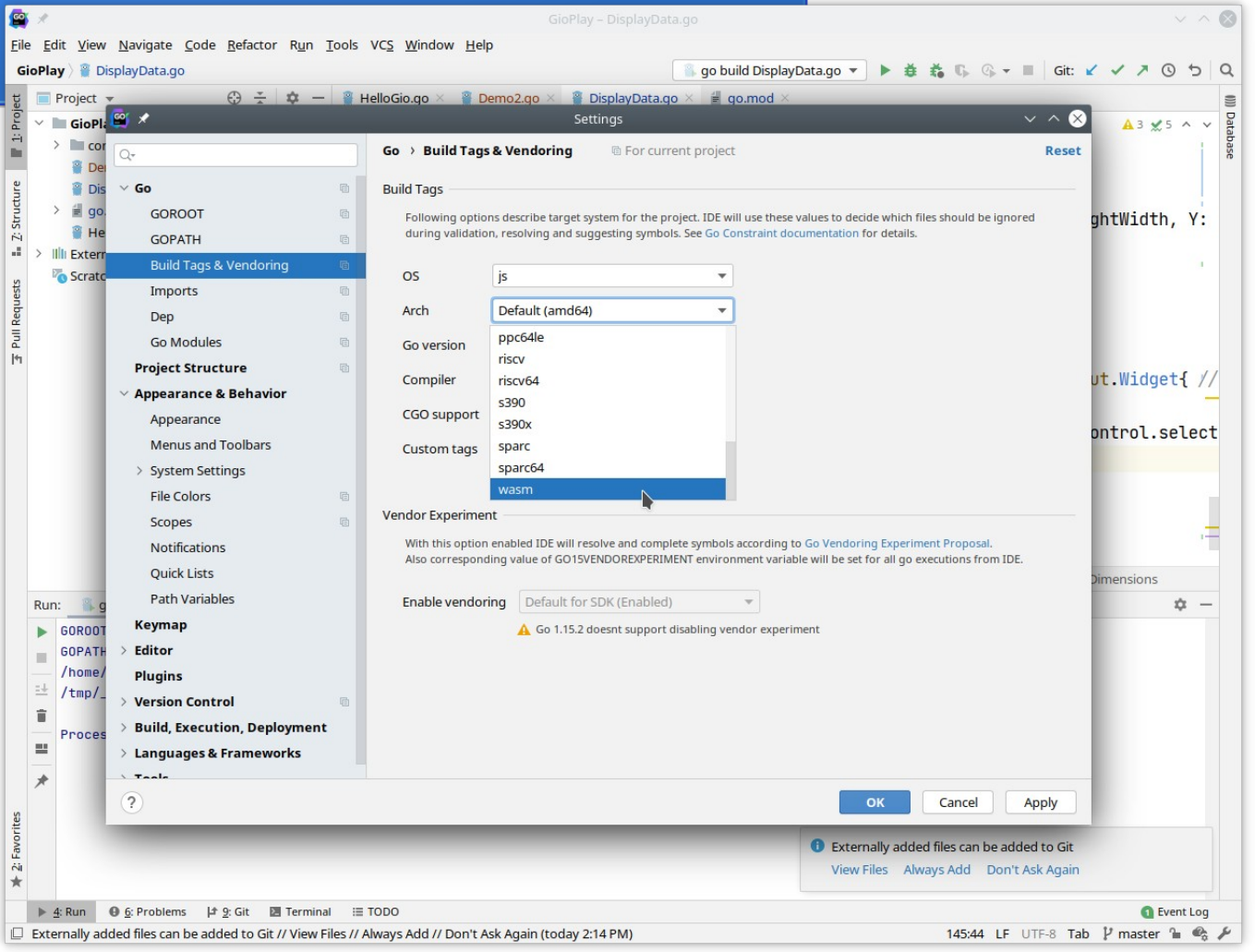
What about non-goland?

- So if we are using goland, then web assembly is easy
  - But go and webassembly without goland is still not bad.
  - At the command line
    - Set the GOOS and GOARCH environment variables
    - For example from the doc I put in the news:
    - GOOS=js GOARCH=wasm go build -o game.wasm <your file name>
  - But the command line is scary??!?!?
    - No but….

From goland

- You can do this painlessly in modern goland
  - Open up the settings/preferences dialog
    - <file> <settings>Or <goland><preferences> in mac land
  - Open the <go> option and select the <build tags & vendoring> option
  - In the OS option choose ‘js’
  - In the Arch option choose wasm
    - Now you are building for web assembly
    - And your executable will no longer run on your local system as before

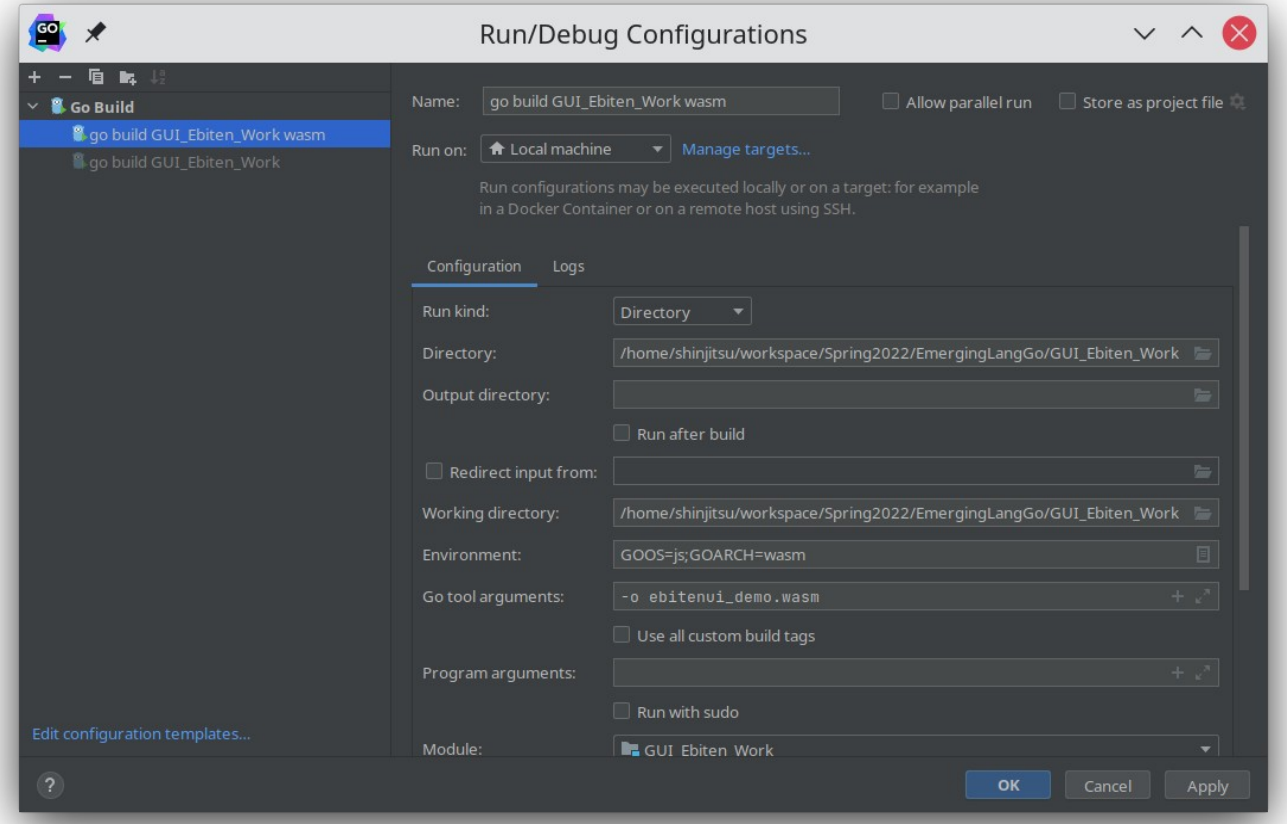
1000 words:


GioPlay – DisplayData.go
File Edit View Navigate Code Refactor Run Tools VCS Window Help
GioPlay DisplayData.go
go build DisplayData.go
Git:
Project
HelloGio.go
Demo2.go
DisplayData.go
go.mod
Settings
Go › Build Tags & Vendoring
For current project
Reset
Go
GOROOT
GOPATH
Build Tags & Vendoring
Imports
Dep
Go Modules
Project Structure
Appearance & Behavior
Appearance
Menus and Toolbars
System Settings
File Colors
Scopes
Notifications
Quick Lists
Path Variables
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
Build Tags
Following options describe target system for the project. IDE will use these values to decide which files should be ignored during validation, resolving and suggesting symbols. See Go Constraint documentation for details.
OS
js
Arch
Default (amd64)
Go version
Compiler
CGO support
Custom tags
ppc64le
riscv
riscv64
s390
s390x
sparc
sparc64
wasm
Vendor Experiment
With this option enabled IDE will resolve and complete symbols according to Go Vendoring Experiment Proposal.
Also corresponding value of GO15VENDOREXPERIMENT environment variable will be set for all go executions from IDE.
Enable vendoring
Default for SDK (Enabled)
Go 1.15.2 doesnt support disabling vendor experiment
OK
Cancel
Apply
Run:
Externally added files can be added to Git
View Files
Always Add
Don't Ask Again
4: Run
6: Problems
9: Git
Terminal
TODO
Event Log
Externally added files can be added to Git // View Files // Always Add // Don't Ask Again (today 2:14 PM)
145:44
LF
UTF-8
Tab
master

Edit run configurations

- One last thing to do, go to <run> menu and <edit configurations>
  - Hit the ‘+’ button to add a new configuration
  - Give it a new name (I like to put wasm in the name)
  - Make run kind *directory*
  - Uncheck ‘run after build’
  - Add two new environment variables GOOS=js;GOARCH=wasm
  - Change go tool arguments to -o <your program>.wasm
  - And hit ok
  - Then run your new configuration, nothing run this time, but you should have a wasm file in your folder, move it to your web project folder

Another 1000 words



Run it

- Run your compiled web server in the folder with your index.html and wasm program
- Then go to http://localhost:8080
  - And lets see your program in the web browser.

Index.html

- Super simple one from article in the news:
- <html>
- <head>
- <meta charset="utf-8"/>
- <script src="wasm_exec.js"></script>
- <script>
- const go = new Go();
- WebAssembly.instantiateStreaming(fetch("json.wasm"), go.importObject).then((result) => {
- go.run(result.instance);
- });
- </script>
- </head>
- <body></body>
- </html>

Lets stop here

- Lets look at the review
- exam next week. (or last week)
- Now you can put your simple games on the web using a web assembly project (mostly likely)