

Go WebAssembly (Wasm) 简明教程

Go 简明教程系列文章链接:

- [Go 语言简明教程](#) (Aug 6, 2019)
- [Go Gin 简明教程](#) (Aug 7, 2019)
- [Go2 新特性简明教程](#) (Aug 15, 2019)
- [Go Protobuf 简明教程](#) (Jan 11, 2020)
- [Go RPC & TLS 鉴权简明教程](#) (Jan 13, 2020)
- [Go WebAssembly \(Wasm\) 简明教程](#) (Jan 23, 2020)
- [Go Test 单元测试简明教程](#) (Feb 10, 2020)
- [Go Mock \(gomock\)简明教程](#) (Feb 14, 2020)
- [Go Mmap 文件内存映射简明教程](#) (Apr 20, 2020)
- [Go Context 并发编程简明教程](#) (Apr 20, 2020)

源代码/数据集已上传到 [Github - 7days-golang](#)



1 WebAssembly 简介

WebAssembly是一种新的编码方式，可以在现代的网络浏览器中运行 - 它是一种低级的类汇编语言，具有紧凑的二进制格式，可以接近原生的性能运行，并为诸如C / C ++等语言提供一个编译目标，以便它们

可以在Web上运行。它也被设计为可以与JavaScript共存，允许两者一起工作。—— MDN web docs - mozilla.org

从 MDN 的介绍中，我们可以得出几个结论：

- 1) WebAssembly 是一种二进制编码格式，而不是一门新的语言。
- 2. WebAssembly 不是为了取代 JavaScript，而是一种补充（至少现阶段是这样），结合 WebAssembly 的性能优势，很大可能集中在对性能要求高（例如游戏，AI），或是对交互体验要求高（例如移动端）的场景。
- 3) C/C++ 等语言可以编译 WebAssembly 的目标文件，也就是说，其他语言可以通过编译器支持，而写出能够在浏览器前端运行的代码。

Go 语言在 1.11 版本(2018年8月) 加入了对 WebAssembly (Wasm) 的原生支持，使用 Go 语言开发 WebAssembly 相关的应用变得更加地简单。Go 语言的内置支持是 Go 语言进军前端的一个重要的里程碑。在这之前，如果想使用 Go 语言开发前端，需要使用 [GopherJS](#)，GopherJS 是一个编译器，可以将 Go 语言转换成可以在浏览器中运行的 JavaScript 代码。新版本的 Go 则直接将 Go 代码编译为 wasm 二进制文件，而不再需要转为 JavaScript 代码。更巧的是，实现 GopherJS 和在 Go 语言中内置支持 WebAssembly 的是同一拨人。

Go 语言实现的函数可以直接导出供 JavaScript 代码调用，同时，Go 语言内置了 [syscall/js](#) 包，可以在 Go 语言中直接调用 JavaScript 函数，包括对 DOM 树的操作。

2 Hello World

如果对 Go 语言不熟悉，推荐 [Go 语言简明教程](#)，一篇文章快速入门。

接下来，我们使用 Go 语言实现一个最简单的程序，在网页上弹出 Hello World。

第一步，新建文件 main.go，使用 `js.Global().get('alert')` 获取全局的 alert 对象，通过 `Invoke` 方法调用。等价于在 js 中调用 `window.alert("Hello World")`。

```
1 // main.go
2 package main
3
4 import "syscall/js"
5
6 func main() {
7     alert := js.Global().Get("alert")
8     alert.Invoke("Hello World!")
9 }
```

第二步，将 main.go 编译为 static/main.wasm

如果启用了 `GO MODULES`，则需要使用 `go mod init` 初始化模块，或设置 `GO111MODULE=auto`。

```
1 $ GOOS=js GOARCH=wasm go build -o static/main.wasm
```

第三步, 拷贝 `wasm_exec.js` (JavaScript 支持文件, 加载 `wasm` 文件时需要) 到 `static` 文件夹

```
1 $ cp "$(go env GOROOT)/misc/wasm/wasm_exec.js" static
```

第四步, 创建 `index.html`, 引用 `static/main.wasm` 和 `static/wasm_exec.js`。

```
1 <html>
2 <script src="static/wasm_exec.js"></script>
3 <script>
4     const go = new Go();
5     WebAssembly.instantiateStreaming(fetch("static/main.wasm"), go.importObject)
6         .then((result) => go.run(result.instance));
7 </script>
8
9 </html>
```

第五步, 使用 `goexec` 启动 Web 服务

如果没有安装 `goexec`, 可用 `go get -u github.com/shurcool/goexec` 安装, 需要将 `$GOBIN` 或 `$GOPATH/bin` 加入环境变量

当前的目录结构如下:

```
1 demo/
2   |--static/
3     |--wasm_exec.js
4     |--main.wasm
5   |--main.go
6   |--index.html
```

```
1 $ goexec 'http.ListenAndServe(`:9999`, http.FileServer(http.Dir(`.`)))'
```

浏览器访问 `localhost:9999`, 则会有一个弹出窗口, 上面写着 `*Hello World!*`。



为了避免每次编译都需要输入繁琐的命令，可将这个过程写在 Makefile 中

```
1 all: static/main.wasm static/wasm_exec.js
2     goexec 'http.ListenAndServe(`:9999`, http.FileServer(http.Dir(`.`)))'
3
4 static/wasm_exec.js:
5     cp "$(shell go env GOROOT)/misc/wasm/wasm_exec.js" static
6
7 static/main.wasm : main.go
8     GO111MODULE=auto GOOS=js GOARCH=wasm go build -o static/main.wasm .
```

这样一个敲一下 make 就够了，代码已经上传到 [7days-golang - github.com](https://github.com/7days-golang)。

3 注册函数(Register Functions)

在 Go 语言中调用 JavaScript 函数是一方面，另一方面，如果仅仅是使用 WebAssembly 替代性能要求高的模块，那么就需要注册函数，以便其他 JavaScript 代码调用。

假设我们需要注册一个计算斐波那契数列的函数，可以这么实现。

```
1 // main.go
2 package main
3
4 import "syscall/js"
5
6 func fib(i int) int {
7     if i == 0 || i == 1 {
8         return 1
9     }
10    return fib(i-1) + fib(i-2)
11 }
12
13 func fibFunc(this js.Value, args []js.Value) interface{} {
14    return js.ValueOf(fib(args[0].Int()))
15 }
```

```

16
17 func main() {
18     done := make(chan int, 0)
19     js.Global().Set("fibFunc", js.FuncOf(fibFunc))
20     <-done
21 }

```

- fib 是一个普通的 Go 函数，通过递归计算第 i 个斐波那契数，接收一个 int 入参，返回值也是 int。
- 定义了 fibFunc 函数，为 fib 函数套了一个壳，从 args[0] 获取入参，计算结果用 js.ValueOf 包装，并返回。
- 使用 js.Global().Set() 方法，将注册函数 fibFunc 到全局，以便在浏览器中能够调用。

js.Value 可以将 Js 的值转换为 Go 的值，比如 args[0].Int()，则是转换为 Go 语言中的整型。js.ValueOf，则用来将 Go 的值，转换为 Js 的值。另外，注册函数的时候，使用 js.FuncOf 将函数转换为 Func 类型，只有 Func 类型的函数，才能在 JavaScript 中调用。可以认为这是 Go 与 JavaScript 之间的接口/约定。

js.Func() 接受一个函数类型作为其参数，该函数的定义必须是：

```

1 func(this Value, args []Value) interface{}
2 // this 即 JavaScript 中的 this
3 // args 是在 JavaScript 中调用该函数的参数列表。
4 // 返回值需用 js.ValueOf 映射成 JavaScript 的值

```

在 main 函数中，创建了信道(chan) done，阻塞主协程(goroutine)。fibFunc 如果在 JavaScript 中被调用，会开启一个新的子协程执行。

A wrapped function triggered during a call from Go to JavaScript gets executed on the same goroutine. A wrapped function triggered by JavaScript's event loop gets executed on an extra goroutine. — [FuncOf - golang.org](#)

接下来，修改之前的 index.html，在其中添加一个输入框(num)，一个按钮(btn) 和一个文本框(ans，用来显示计算结果)，并给按钮添加了一个点击事件，调用 fibFunc，并将计算结果显示在文本框(ans)中。

```

1 <html>
2 ...
3 <body>
4     <input id="num" type="number" />
5     <button id="btn" onclick="ans.innerHTML=fibFunc(num.value * 1)">Click</button>
6     <p id="ans">1</p>
7 </body>
8 </html>

```

使用之前的命令重新编译 main.go，并在 9999 端口启动 Web 服务，如果我们已经将命令写在 Makefile 中了，只需要运行 make 即可。

接下来访问 localhost:9999，可以看到如下效果。输入一个数字，点击 Click，计算结果显示在输入框下方。

Click

233

4 操作 DOM

在上一个例子中，仅仅是注册了全局函数 fibFunc，事件注册，调用，对 DOM 元素的操作都是在 HTML 中通过原生的 JavaScript 函数实现的。这些事情，能不能全部在 Go 语言中完成呢？答案可以。

首先修改 index.html，删除事件注册部分和对 DOM 元素的操作部分。

```
1  <html>
2  ...
3  <body>
4      <input id="num" type="number" />
5      <button id="btn">Click</button>
6      <p id="ans">1</p>
7  </body>
8  </html>
```

修改 main.go:

```
1  package main
2
3  import (
4      "strconv"
5      "syscall/js"
6  )
7
8  func fib(i int) int {
9      if i == 0 || i == 1 {
10         return 1
11     }
12     return fib(i-1) + fib(i-2)
13 }
14
15 var (
16     document = js.Global().Get("document")
17     numEle   = document.Call("getElementById", "num")
18     ansEle   = document.Call("getElementById", "ans")
```

```

19     btnEle    = js.Global().Get("btn")
20 )
21
22 func fibFunc(this js.Value, args []js.Value) interface{} {
23     v := numEle.Get("value")
24     if num, err := strconv.Atoi(v.String()); err == nil {
25         ansEle.Set("innerHTML", js.ValueOf(fib(num)))
26     }
27     return nil
28 }
29
30 func main() {
31     done := make(chan int, 0)
32     btnEle.Call("addEventListener", "click", js.FuncOf(fibFunc))
33     <-done
34 }

```

- 通过 `js.Global().Get("btn")` 或 `document.Call("getElementById", "num")` 两种方式获取到 DOM 元素。
- `btnEle` 调用 `addEventListener` 为 `btn` 绑定点击事件 `fibFunc`。
- 在 `fibFunc` 中使用 `numEle.Get("value")` 获取到 `numEle` 的值（字符串），转为整型并调用 `fib` 计算出结果。
- `ansEle` 调用 `Set("innerHTML", ...)` 渲染计算结果。

重新编译 `main.go`，访问 `localhost:9999`，效果与之前是一致的。

5 回调函数(Callback Functions)

在 JavaScript 中，异步+回调是非常常见的，比如请求一个 Restful API，注册一个回调函数，待数据获取到，再执行回调函数的逻辑，这个期间程序可以继续做其他的事情。Go 语言可以通过协程实现异步。

假设 `fib` 的计算非常耗时，那么可以启动注册一个回调函数，待 `fib` 计算完成后，再把计算结果显示出来。

我们先修改 `main.go`，使得 `fibFunc` 支持传入回调函数。

```

1  package main
2
3  import (
4      "syscall/js"
5      "time"
6  )
7
8  func fib(i int) int {
9      if i == 0 || i == 1 {
10         return 1

```

```

11     }
12     return fib(i-1) + fib(i-2)
13 }
14
15 func fibFunc(this js.Value, args []js.Value) interface{} {
16     callback := args[len(args)-1]
17     go func() {
18         time.Sleep(3 * time.Second)
19         v := fib(args[0].Int())
20         callback.Invoke(v)
21     }()
22
23     js.Global().Get("ans").Set("innerHTML", "Waiting 3s...")
24     return nil
25 }
26
27 func main() {
28     done := make(chan int, 0)
29     js.Global().Set("fibFunc", js.FuncOf(fibFunc))
30     <-done
31 }

```

- 假设调用 fibFunc 时，回调函数作为最后一个参数，那么通过 args[len(args)-1] 便可以获取到该函数。这与其他类型参数的传递并无区别。
- 使用 go func() 启动子协程，调用 fib 计算结果，计算结束后，调用回调函数 callback，并将计算结果传递给回调函数，使用 time.Sleep() 模拟 3s 的耗时操作。
- 计算结果出来前，先在界面上显示 Waiting 3s...

接下来我们修改 index.html，为按钮添加点击事件，调用 fibFunc

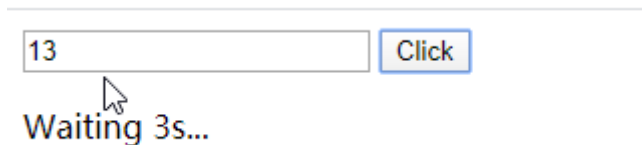
```

1  <html>
2  ...
3  <body>
4      <input id="num" type="number" />
5      <button id="btn" onclick="fibFunc(num.value * 1, (v)=> ans.innerHTML=v)">Click</butt
6      <p id="ans"></p>
7  </body>
8  </html>

```

- 为 btn 注册了点击事件，第一个参数是待计算的数字，从 num 输入框获取。
- 第二个参数是一个回调函数，将参数 v 显示在 ans 文本框中。

接下来，重新编译 main.go，访问 localhost:9999，随便输入一个数字，点击 Click。页面会先显示 Waiting 3s...，3s过后显示计算结果。



6 进一步的尝试

6.1 工具框架

- WebAssembly 的二进制分析工具 [WebAssembly Code Explorer](#)
- 使用NodeJs 或浏览器测试 Go Wasm 代码 [Github Wiki](#)
- 借鉴 Vue 实现的 Golang WebAssembly 前端框架 [Vugu](#)，完全使用 Go，不用写任何的 JavaScript 代码。

6.2 Demo/项目

- 使用 Go Assembly 前端渲染的一些例子
- [jsgo](#) 这个项目汇聚一些小而精的项目，包括 2048，俄罗斯方块等游戏，还有证明 Go 可以完整开发前端项目的 [TodoMVC](#)

6.3 相关文档

- [syscall/js 官方文档 - golang.org](#)
- [Go WebAssembly 官方文档 - github.com](#)

专题: [Go 简明教程](#)

本文发表于 2020-01-23，最后修改于 2021-02-06。

本站永久域名「[geektutu.com](#)」，也可搜索「极客兔兔」找到我。

上一篇 « [Go RPC & TLS 鉴权简明教程](#) 下一篇 » [7天用Go从零实现分布式缓存GeeCache](#)

赞赏支持



推荐阅读

pprof 性能分析

发表于2020-11-22，阅读约35分钟

动手写分布式缓存 - GeeCache第七天 使用 Protobuf 通信

发表于2020-02-17, 阅读约16分钟

WSL, Git, Mircosoft Terminal 等常用工具配置

发表于2019-12-03, 阅读约17分钟

#关于我 (9) #Go (48) #百宝箱 (2) #Cheat Sheet (1) #Go语言高性能编程 (20) #友链 (1) #Pandas (3)

#机器学习 (9) #TensorFlow (9) #mnist (5) #Python (10) #强化学习 (3) #OpenAI gym (4) #DQN (1)

#Q-Learning (1) #CNN (1) #TensorFlow 2 (10) #官方文档 (10) #Rust (1)



Gitalk 加载中 ...


Go Mock (gomock)简明教程

5 评论 • 11天前

 **zouzhiwu** —— 写错了: NewMockDB() 的定义在 db_mock.go 中, 由 mockgen 自动生成。

Go语言动手写Web框架 - Gee第五天 中间件 Middleware

18 评论 • 14天前

 **YoshieraHuang** —— 你好, 大佬写的文章很受用。有个问题想问一下, 在使用循环调用


Go语言动手写Web框架 - Gee第一天 http.Handler

17 评论 • 1天前

 **mesiyar** —— mark 跟进学习

动手写RPC框架 - GeeRPC第四天 超时处理 (timeout)

4 评论 • 24天前

 **Phoenix500526** —— 大佬你好, 我发现在 day4-timeout 时, 执行 `go test -v` 会有一

Gitalk Plus

© 2021 - 极客兔兔 - 沪ICP备18001798号-1

Powered by [Hexo](#) | Theme [Geektutu](#) 

👁 996644 📄 6786