WebAssembly & Socket Introduction

by 贺恩泽

WebAssembly

https://webassembly.org

Why not JavaScript?

- 语言过于灵活，导致大型项目开发困难
- 需要解释执行，且执行时需要大量类型推导，性能牺牲太大
- 通过文本方式传输，即使应用了各种压缩技术也只是杯水车薪(如对普通的 1 mb 的 js 文件采用最新的 Brotli 算法压缩，通常只能压缩到 100~200 kb 左右，虽然有很理想的压缩率，但还是很大)

Why Assembly?

- 体积小
  wasm 采用二进制编码，类似 Java 程序编译后生成的字节码
- 安全
  运行在沙箱中，不会对系统产生影响
- 便于调试和优化
  每条指令有对应的二进制值，类似汇编
  存在 wat 文本 WebAssembly 格式，便于调试优化
- 不存在版本，标准化，不限语言
  WebAssembly 没有版本一说，向后兼容。只要是支持编译到 wasm 的语言都可以使用
- 高性能
  无需解释执行，直接编译为目标架构的机器代码后执行

Example

加法

C++ 代码

```
int add(int num1, int num2) {
  return num1 + num2;
}
```

Example

加法

wat 文本代码

```
(module
  (export "add" (func $add))
  (func $add (param $0 i32)
    (param $1 i32)
    (result i32)
    (i32.add
      (get_local $1)
      (get_local $0)
    )
  )
)
```

Example

加法

wasm 数据

```
00 61 73 6d 01 00 00 00 01 07 01 60 02 7f 7f 01
7f 03 02 01 00 07 07 01 03 61 64 64 00 00 0a 09
01 07 00 20 01 20 00 6a 0b
```

Example

加法

在 console 中调用

```
let p = new Uint8Array([
0x00, 0x61, 0x73, 0x6d, 0x01, 0x00, 0x00, 0x00, 0x01,
0x07, 0x01, 0x60, 0x02, 0x7f, 0x7f, 0x01, 0x7f, 0x03,
0x02, 0x01, 0x00, 0x07, 0x07, 0x01, 0x03, 0x61, 0x64,
0x64, 0x00, 0x00, 0x0a, 0x09, 0x01, 0x07, 0x00, 0x20,
0x01, 0x20, 0x00, 0x6a, 0x0b]);

WebAssembly.instantiate(p) //编译成目标架构的机器码并实例化
  .then(x => console.log(x.instance.exports.add(4,5)));
```

输出

```
9
```

Tool chain

https://github.com/WebAssembly/wabt

Browser support

- Edge 16+
- Firefox 52+
- Chrome 57+
- Safari 11+
- Opera 44+

Language support

- C/C++
- Rust
- TypeScript
- C#
- Java
- Go
- Swift

  .....

Shortcoming

- 目前还没有原生的 WebAssembly DOM API，需要导入 JavaScript DOM API 进行调用
- 和 JavaScript 一样还没有原生多线程
- 目前在网页中需要引入一个 js 来获取和编译 wasm 文件后执行。但是以后则可以使用以下的形式来引入 WebAssembly 文件

  ```
  <script src="xxx.wasm"></script>
  ```

  并且有了 WebAssembly DOM API 后可以彻底去 js 化

  ……

Limit

- 单个程序内存分配不能超过 2032 Mb，否则会失败

Roadmap

- 多线程
- SIMD
- 异常处理
- 引用类型
- 垃圾回收
- 尾调用
- WebAssembly DOM API

  ……

Socket

What is Socket

网络上两个程序通过一种双向连接实现通信，其中的一端称为一个socket。

Server's output stream --> Client's input stream
Client's output stream --> Server's input stream

Type

- Network Socket (TCP/UDP)
- Unix File Socket
- Web Socket

...

TCP Socket Server

- Create
  - Create socket
  - Set properties
  - Bind IP Address, Port
  - Start listening
- Incoming connection
  - Accept
- Data transfer
  - Send/Write
  - Receive/Read
- Close
  - Close connection
  - Stop listening

TCP Socket Client

- Create
  - Create socket
  - Set properties
  - Set IP Address, Port
  - Connect
- Data transfer
  - Send/Write
  - Receive/Read
- Close
  - Close connection

Web Socket

Server: ws(s)://.....

Connect WebSocket

- Request

```
GET url HTTP/1.1

Host: ...
Upgrade: websocket

Connection: Upgrade
Sec-WebSocket-Key: ...
Origin: ...
Sec-WebSocket-Version: 13
```

Sec-WebSocket-Key 是浏览器随机生成的 base64 值，用来询问服务器是否是支持 WebSocket

Web Socket

Connect to WebSocket

- Response

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: ...
```

Sec-WebSocket-Accept 是将 Request 的 Sec-WebSocket-Key 与 "258EAFA5-E914-47DA-95CA-C5AB0DC85B11" 这个字符串进行拼接，然后对拼接后的字符串进行 sha-1 运算，再进行 base64 编码的，用来说明自己支持 WebSocket

HTML5 Web Socket API

```
let ws = new WebSocket(url);
ws.onmessage = msg => { ... };
ws.onerror = err => { ... };
ws.onclose = rsn => { ... };
ws.send(msg);
ws.close();
```

Keep alive

Heartbeat package

```
        echo
Client -----> Server
        echo
Server -----> Client
```

对于 TCP Socket/Web Socket，内置了心跳包机制 (TCP Socket 为可选启用)，当连接意外断开时会自动告知用户。一般检测时间为 30 秒/次。

Type

- 长连接
- 短连接

Status

- 连接
- 半连接
- 断开连接

I/O Multiplexing

- select()
- poll()
- epoll()

Note

- 读取数据的方法会阻塞等待对方应答，如果没有收到数据则会一直阻塞下去
- 粘包问题解决方法：封包
- 关闭流会导致连接断开，需要重新建立 WebSocket 连接才能继续使用

封包

最基本的封包结构：

```
Package: |Header|Data|
Header: |Id|Length|
```

感谢聆听

中山大学微软学生俱乐部 贺恩泽

2018/11/25