MIGU 咪咕

WebAssembly 浅谈及使用实践

赵磊 @flyingzl

目录
CONTENTS

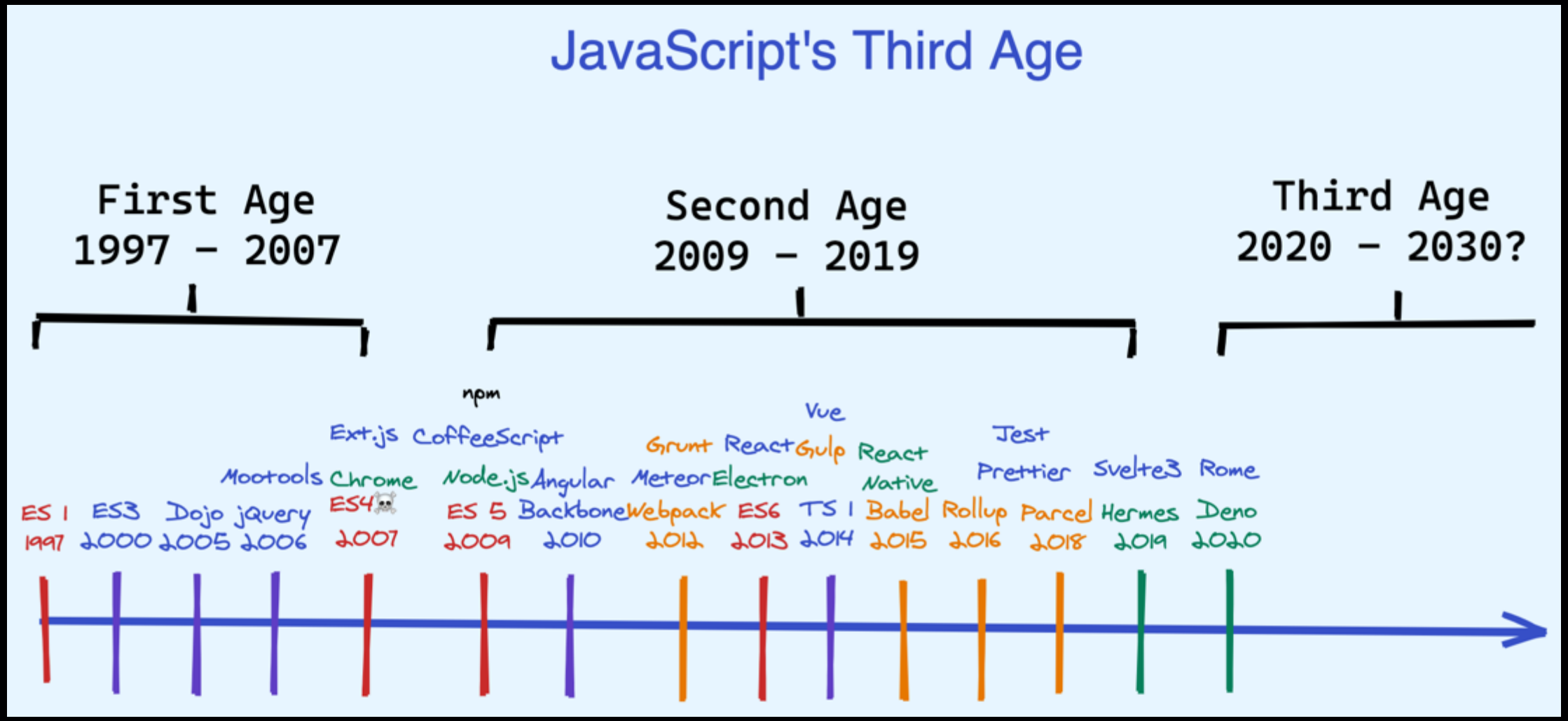
MIGU 咪咕

01 WebAssembly 简介

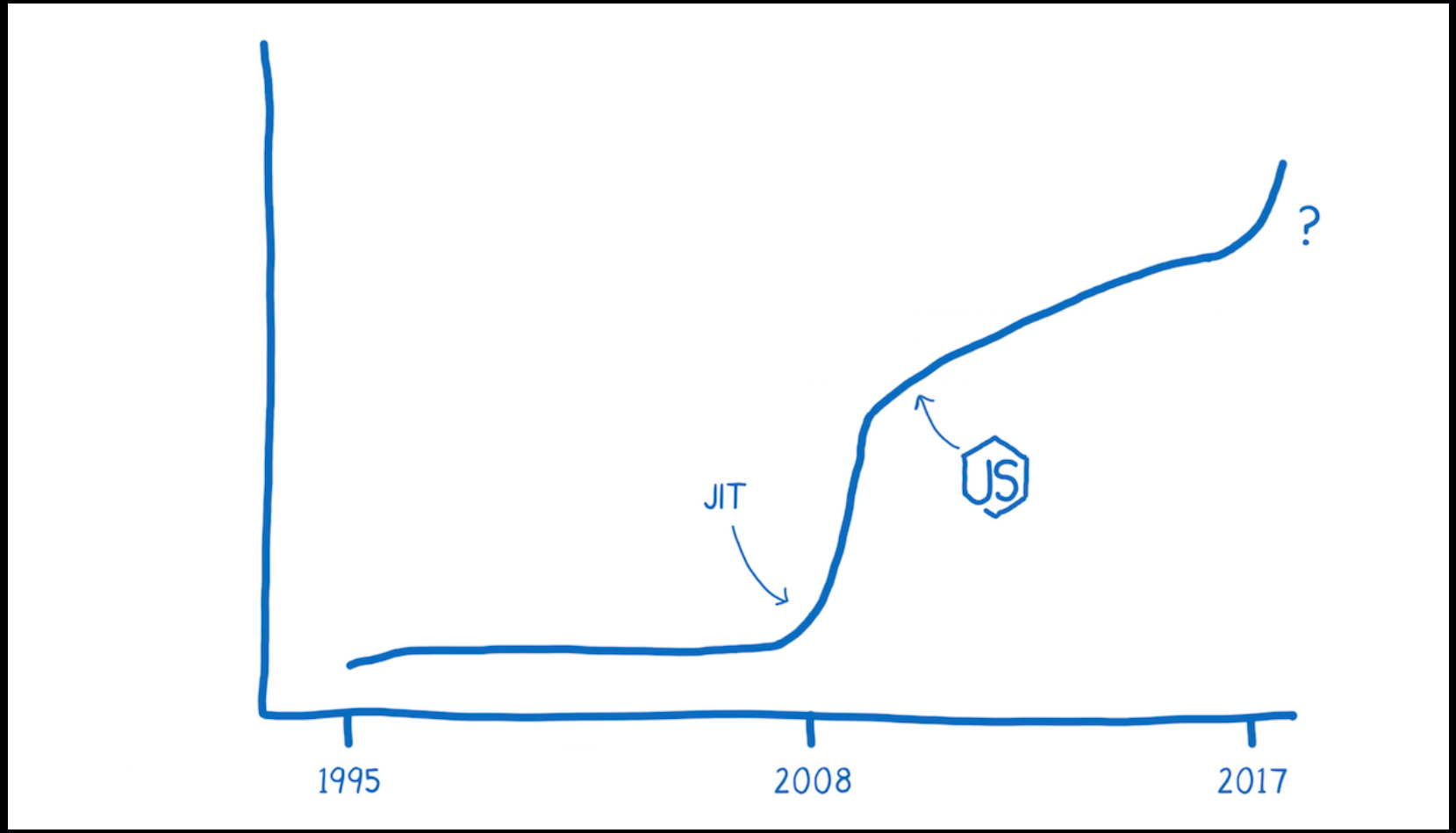
02 WebAssembly 构建方式

03 WebAssembly 使用实践

JavaScript 生态图鉴

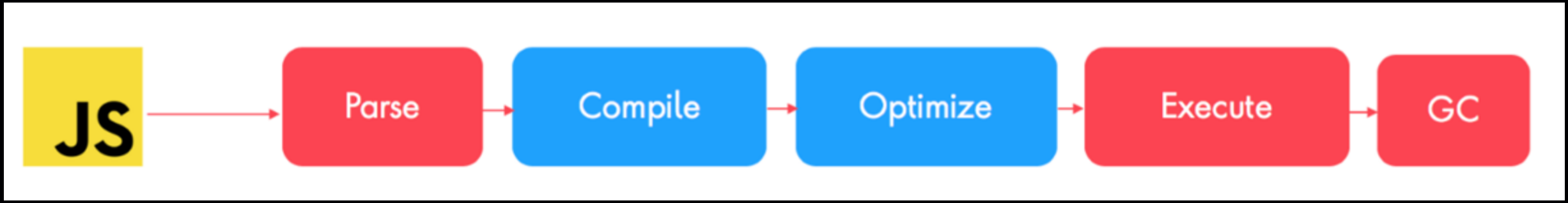


下一个爆点/转折点是啥？



from a-cartoon-intro-to-webassembly

JavaScript 执行生命周期



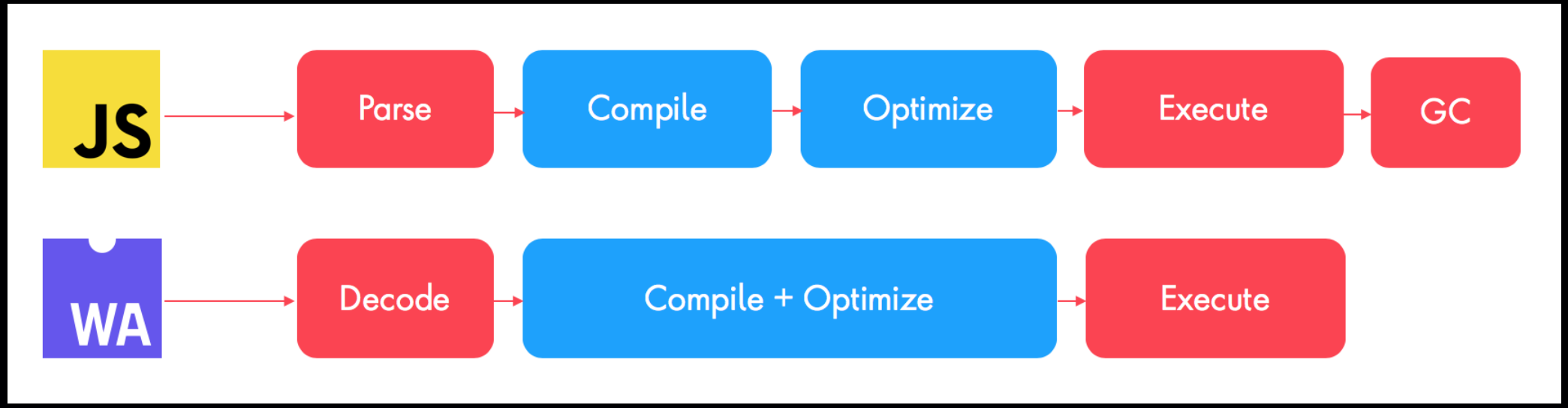
是否存在更多优化可能？

什么是 WebAssembly

MIGU 咪咕

- WebAssembly 是一个可移植、体积小、加载快并且兼容 Web 的全新二进制格式
- 可以通过C/C++/Rust等静态语言编译生成，后缀名为.wasm（0x6d736100）
- 类汇编的一种新的抽象栈式虚拟机
- 可以通过W3C 标准Web API在浏览器中加载、解析和运行

WebAssembly 为什么相对会更快？



WebAssembly 为什么相对会更快？

MIGU 咪咕

- wasm为紧凑的二进制文件，解码时间更快, 无需生成抽象语法树
- wasm更为接近机器码，已经高度优化过，无需浏览器进行更多的优化和编译
- wasm已经拥有静态类型校验，无需再次优化或者去优化操作
- 没有垃圾回收机制

目录
CONTENTS

MIGU 咪咕

01 WebAssembly 简介

02 WebAssembly 构建方式

03 WebAssembly 使用实践

WebAssembly 可以通过多种方式生成

- WAT (WebAssembly Text Format)
- AssemblyScript
- Golang
- Emscripten

WAT生成 wasm

MIGU 咪咕

```
(module
  (func $sum(param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.add
  )

  (export "addTwo" (func $sum))
)
```

```
wat2wasm sum.wat -o sum.wasm
```

JavaScript 调用 wasm

MIGU 咪咕

```
const fetchAndInstantiate = async (url, importObject = {}) => {
    const response = await fetch(url)
    const bytes = await response.arrayBuffer()
    const results = await WebAssembly.instantiate(bytes, importObject)
    return results.instance.exports
}

(async () => {
    const { addTwo } = await fetchAndInstantiate('sum.wasm')
    console.log(addTwo(1, 2))
})()
```

WAT样例 – 计算阶乘

MIGU 咪咕

vim fib.wat

```
(module
    (func $fac (export "fib")(param $n f64)(result f64)
        ;; if (n < 1) return 1
        (if
            (f64.lt
                (local.get $n)
                (f64.const 1)
            )
            (return (f64.const 1) )
        )

        ;; return n * fib(n-1)
        (return
            (f64.mul
                (local.get $n)
                (call $fac
                    (f64.sub
                        (local.get $n)
                        (f64.const 1)
                    )
                )
            )
        )
    )
)
```

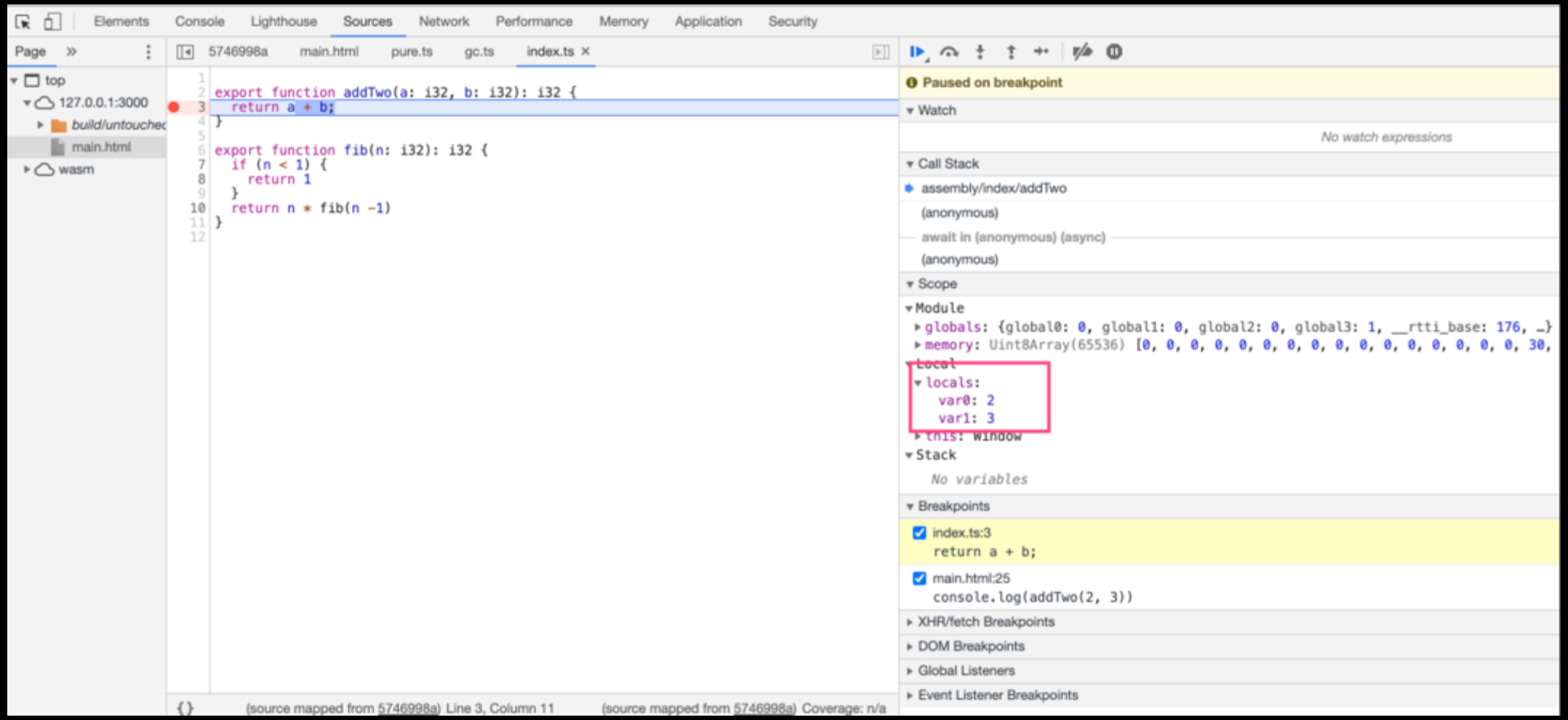
F(n) = n * F(n-1)

AssemblyScript 生成wasm

```
export function addTwo(a: i32, b: i32): i32 {
  return a + b;
}

export function fib(n: i32): i32 {
  if (n < 1) {
    return 1
  }
  return n * fib(n -1)
}
```

https://github.com/AssemblyScript/assemblyscript

AssemblyScript SourceMap 调试



Golang 生成wasm

MIGU 咪咕

```go
package main

import (
    "syscall/js"
)

func addTwo(this js.Value, args []js.Value) interface{} {
    a, b := args[0].Int(), args[1].Int()
    return a + b
}

func fib(this js.Value, args []js.Value) interface{} {
    value := args[0].Int()
    var fibInner func(n int) int
    fibInner = func(n int) int {
        if n < 1 {
            return 1
        }
        return n * fibInner(n-1)
    }
    return fibInner(value)
}

func main() {
    js.Global().Set("addTwo", js.FuncOf(addTwo))
    js.Global().Set("fib", js.FuncOf(fib))
    select {}
}
```

```sh
# 复制胶水脚本
cp $(go env GOROOT)/misc/wasm/wasm_exec.js .

# 编译为wasm
GOARCH=wasm GOOS=js go build -o main.wasm main.go
```

调用 Golang生成的wasm

```
<script src="wasm_exec.js"></script>
<script>
    const fetchAndInstantiate = async (url, importObject = {}) => {
        const response = await fetch(url)
        const bytes = await response.arrayBuffer()
        const results = await WebAssembly.instantiate(bytes, importObject)
        return results.instance
    }

    (async () => {
        const go = new Go()
        const instance = await fetchAndInstantiate('main.wasm', go.importObject)
        await go.run(instance)
    })()

    setTimeout(() => {
        console.log(window.addTwo(1, 2))
        console.log(window.fib(5))
    }, 1000)
</script>
```

Emscripten 生成 wasm

MIGU 咪咕

```c
#include <emscripten.h>

int EMSCRIPTEN_KEEPALIVE addTwo(int a, int b) {
  return a + b;
}

int EMSCRIPTEN_KEEPALIVE fib(int n) {
  if (n < 1){
    return 1;
  }
  return n * fib(n-1);
}
```

```
emcc main.c -s EXTRA_EXPORTED_RUNTIME_METHODS='["cwrap", "ccall"]' -o main.js
```

调用 Emscripten生成的wasm

MIGU 咪咕

```
<script src='main.js'></script>
<script>
    Module.onRuntimeInitialized = () => {
        // 调用addTwo
        const addTwo = Module.cwrap('addTwo', 'number', ['number', 'number'] )
        console.log(addTwo(1, 2))

        // 通过Module.cwrap调用fib
        const fib = Module.cwrap('fib', 'number', ['number'])
        console.log(`fib(5)=${fib(5)}`)

        // 通过Module.ccall调用
        const result = Module.ccall('fib', 'number', ['number'], [5])
        console.log(`fib(5)=${result}`)
    }
</script>
```

目录

CONTENTS

MIGU 咪咕

01 WebAssembly 简介

02 WebAssembly 构建方式

03 WebAssembly 使用实践

MD5 加密

MIGU 咪咕

```c
// md5.c
#include <emscripten.h>
#include <openssl/md5.h>
#include <string.h>
#include <stdio.h>

EMSCRIPTEN_KEEPALIVE
void md5(char *str, char *result)
{
    MD5_CTX md5_ctx;
    int MD5_BYTES = 16;
    unsigned char md5sum[MD5_BYTES];
    MD5_Init(&md5_ctx);
    MD5_Update(&md5_ctx, str, strlen(str));
    MD5_Final(md5sum, &md5_ctx);
    char temp[3] = {0};
    memset(result, 0, sizeof(char) * 32);
    for (int i = 0; i < MD5_BYTES; i++)
    {
        sprintf(temp, "%02x", md5sum[i]);
        strcat(result, temp);
    }
    result[32] = '\0';
}
```

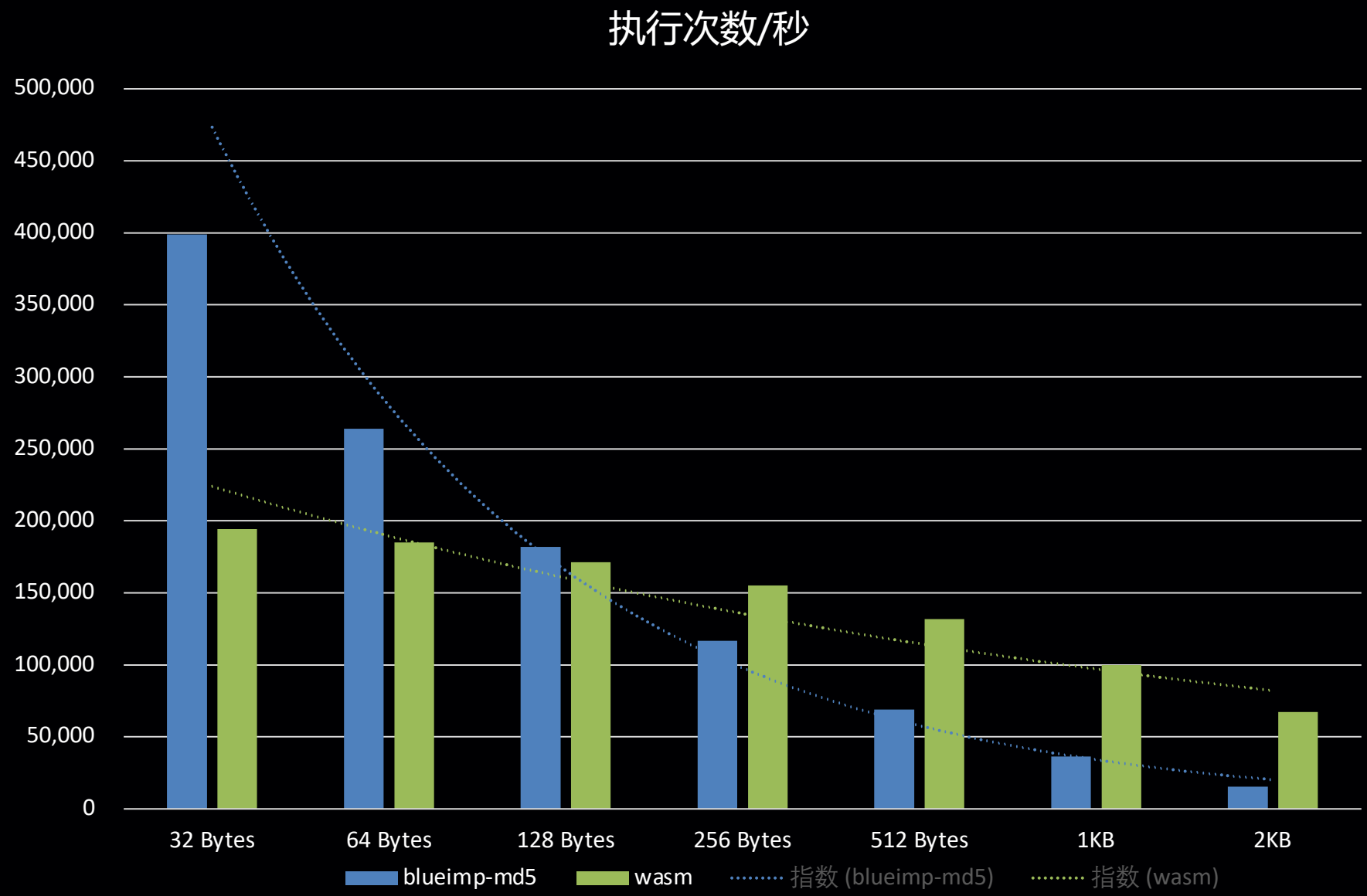
```
OPENSSL_INCLUDE = /Users/zhaolei/Downloads/openssl/include
OPENSSL_LIB = /Users/zhaolei/Downloads/openssl/libs
INPUT_FILE = md5.c
OUTPUT_FILE = md5.js
md5:
    emcc md5.c \
        -O3 \
        -I ${OPENSSL_INCLUDE} \
        -L ${OPENSSL_LIB} \
        -lcrypto \
        -s EXTRA_EXPORTED_RUNTIME_METHODS="['cwrap']" \
        -s EXPORTED_FUNCTIONS="['_malloc', '_free']" \
        -o ${OUTPUT_FILE}
clean:
    rm md5.wasm md5.js
```

MD5 加密

MIGU 咪咕

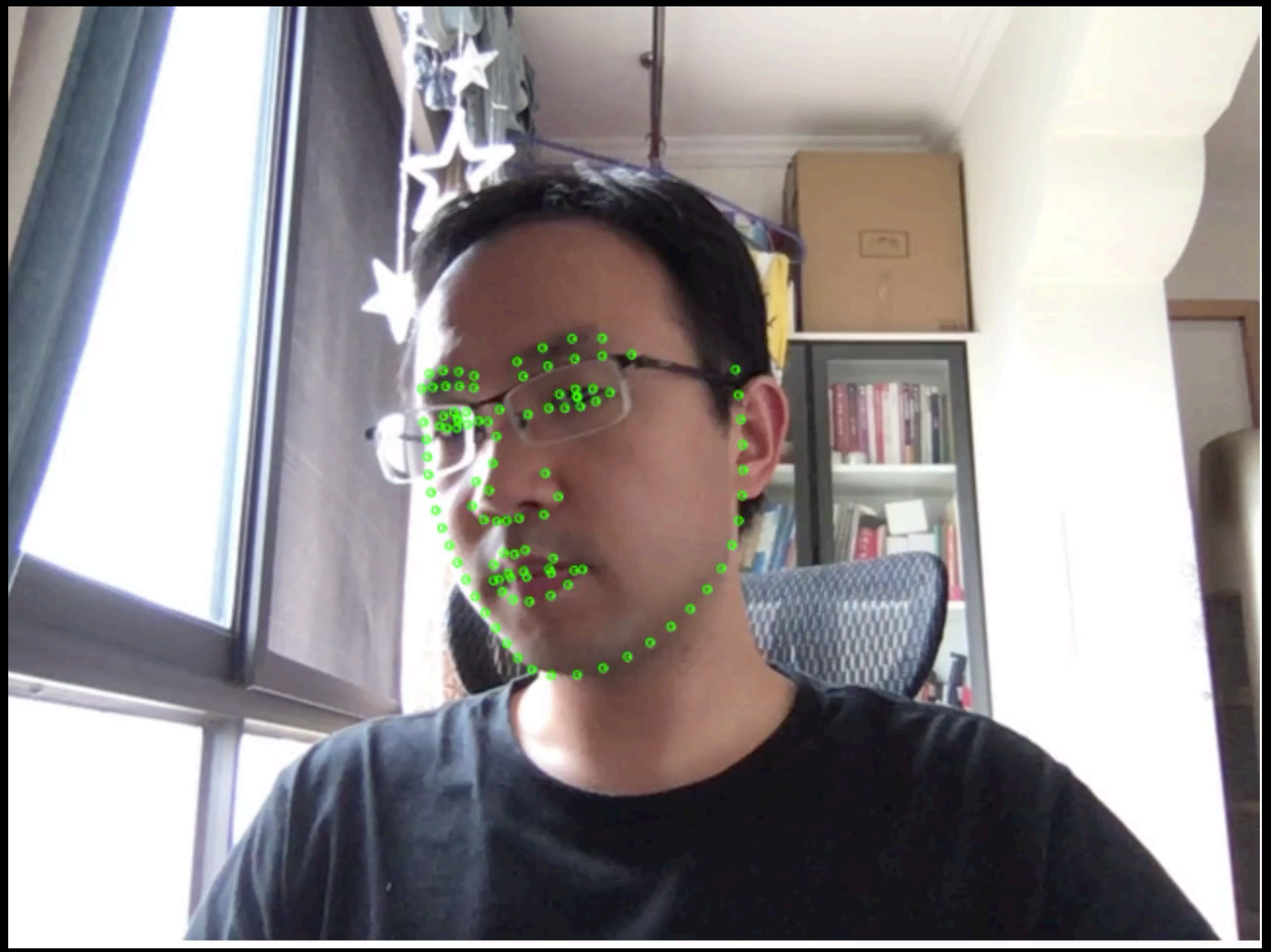
```
<script src='md5.js'></script>
<script>
    const mallocByteBuffer = len => {
        const ptr = Module._malloc(len)
        const heapBytes = new Uint8Array(Module.HEAPU8.buffer, ptr, len)
        return heapBytes
    }

    Module.onRuntimeInitialized = () => {
        const md5 = Module.cwrap('md5', null, ['string', 'number'])
        // 需要进行md5加密的字符串
        const str = 'admin'
        const outBuffer = mallocByteBuffer(32)
        // 调用wasm接口,传递指针
        md5(str, outBuffer.byteOffset)
        const result = new TextDecoder('utf-8').decode(outBuffer)
        // 输出md5值
        console.log(result)
        // 释放内存
        Module._free(outBuffer.byteOffset)
    }
</script>
```

MD5 加密 Benchmark



人脸识别

MIGU 咪咕



人脸识别

MIGU 咪咕

```
async faceRecognitionWithWASM () {
    // 加载模型
    await this.loadLandMarksModels()
    const next = () => {
        this.detectSingleFace()
        requestAnimationFrame(next)
    }
    requestAnimationFrame(next)
},

detectSingleFace () {
    const video = document.querySelector('video')
    // 获取视频数据
    const { data, width, height } = this.getImageData(video)
    // 将数据传递到wasm
    faceApi.setInputBuffer(data, width, height)
    // 从wasm中获取人脸信息
    let points = faceApi.getLandmarks()
    // 绘制人脸
    this.drawLandMarks(points)
}
```

人脸识别

MIGU 咪咕

```
export default function createNativeInterface (m) {
    const Module = m
    return {
        mallocByteBuffer (len) {
            const ptr = Module._malloc(len)
            const heapBytes = new Uint8Array(Module.HEAPU8.buffer, ptr, len)
            return heapBytes
        },

        mallocFloat32Buffer (len) {
            const ptr = Module._malloc(len * 4)
            const heapBytes = new Float32Array(Module.HEAPF32.buffer, ptr, len)
            return heapBytes
        },

        mallocUInt32Buffer (len) {
            const ptr = Module._malloc(len * 4)
            const heapBytes = new Uint32Array(Module.HEAPU8.buffer, ptr, len)
            return heapBytes
        },

        free (nativeBuffer) {
            if (typeof nativeBuffer === 'number') {
                Module._free(nativeBuffer)
            } else if (nativeBuffer && nativeBuffer.buffer) {
                nativeBuffer.buffer instanceof ArrayBuffer &&
                    Module._free(nativeBuffer.byteOffset)
            }
        },

        wrapNative (
            methodName,
            methodParamTypes = ['number'],
            returnType = 'number'
        ) {
            return Module.cwrap(methodName, returnType, methodParamTypes)

        }
    }
}
```

展望 WebAssembly 未来

MIGU 咪咕

- 多线程和原子操作支持
- 更完善SIMD指令支持
- ES6模块直接载入
- 直接操作DOM
- 更完善的调试
- WASI（WebAssembly System Interface）

MIGU 咪咕

THANK YOU