

第三届 FEDAY(前端开发日)

Ucloud

100offer

upyun

Brandview

国兴教育

GenePlan

StuQ

CSDN

W3C

掘金

向端学院

中世科技

segmentfault

码云

WebAssembly 在白鸚引擎中的实践

王泽

白鸚引擎 首席架构师

背景知识 - 白鹭引擎

- 可视化开发工具
 - Egret Wing
 - Dragonbones
 - Egret Paper
- 命令行脚本
 - 项目模板
 - 编译器 (Base on TypeScript)
 - 资源管理框架
- 核心 JavaScript 运行时库
 - 2D / 3D 渲染库
 - 动画 / 粒子 / 物理库
 - 其他

背景知识 - 白鹭引擎

- HTML5 游戏引擎

- 开发效率

- 加载效率

- 运行效率

背景知识 - 白鹭引擎

- HTML5 游戏引擎


- 开发效率

- 加载效率

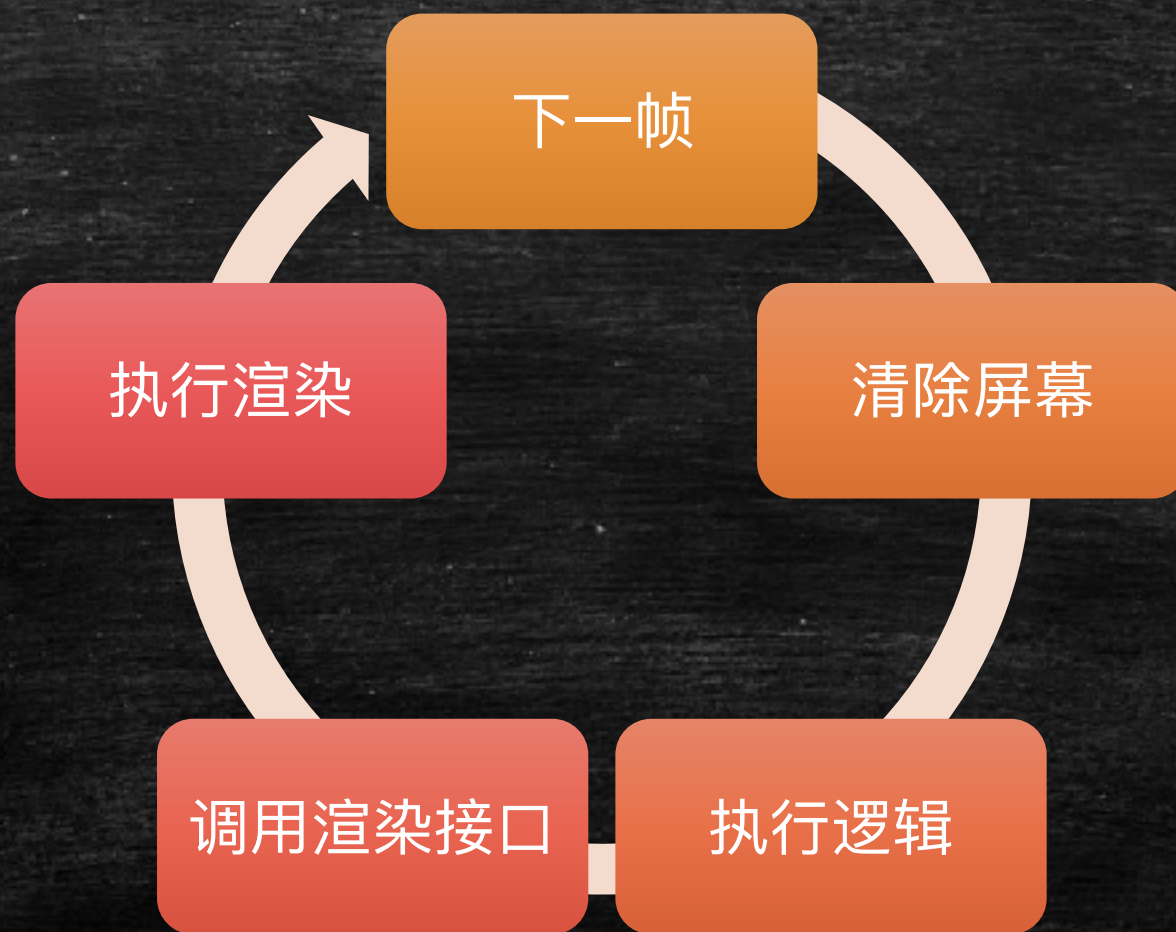
- 运行效率

背景知识：渲染内核原理

```
1 <body>
2   <canvas width="640" height="960" />
3   <script>
4     var canvas = document.getElementById("gameCanvas");
5     var ctx = canvas.getContext("2d");
6     canvas.width = 640;
7     canvas.height = 960;
8     canvas.style.backgroundColor = "red";
9   </script>
10 </body>
```



背景知识：渲染内核原理



背景知识：渲染内核原理

```
1  <body>
2    <canvas id="gameCanvas" width="640" height="960" />
3    <script type="text/javascript">
4      var canvas = document.getElementById("gameCanvas");
5      var canvasContext = canvas.getContext("2d");
6      var index = 0;
7      function onEnterFrame() {
8        index++;
9        canvasContext.clearRect(0, 0, 640, 960)
10       canvasContext.fillStyle = "red";
11       canvasContext.fillRect(0, index, 100, 100);
12       requestAnimationFrame(onEnterFrame);
13     }
14     requestAnimationFrame(onEnterFrame);
15   </script>
16 </body>
```


背景知识：渲染内核原理

用户逻辑 JavaScript



引擎逻辑 JavaScript

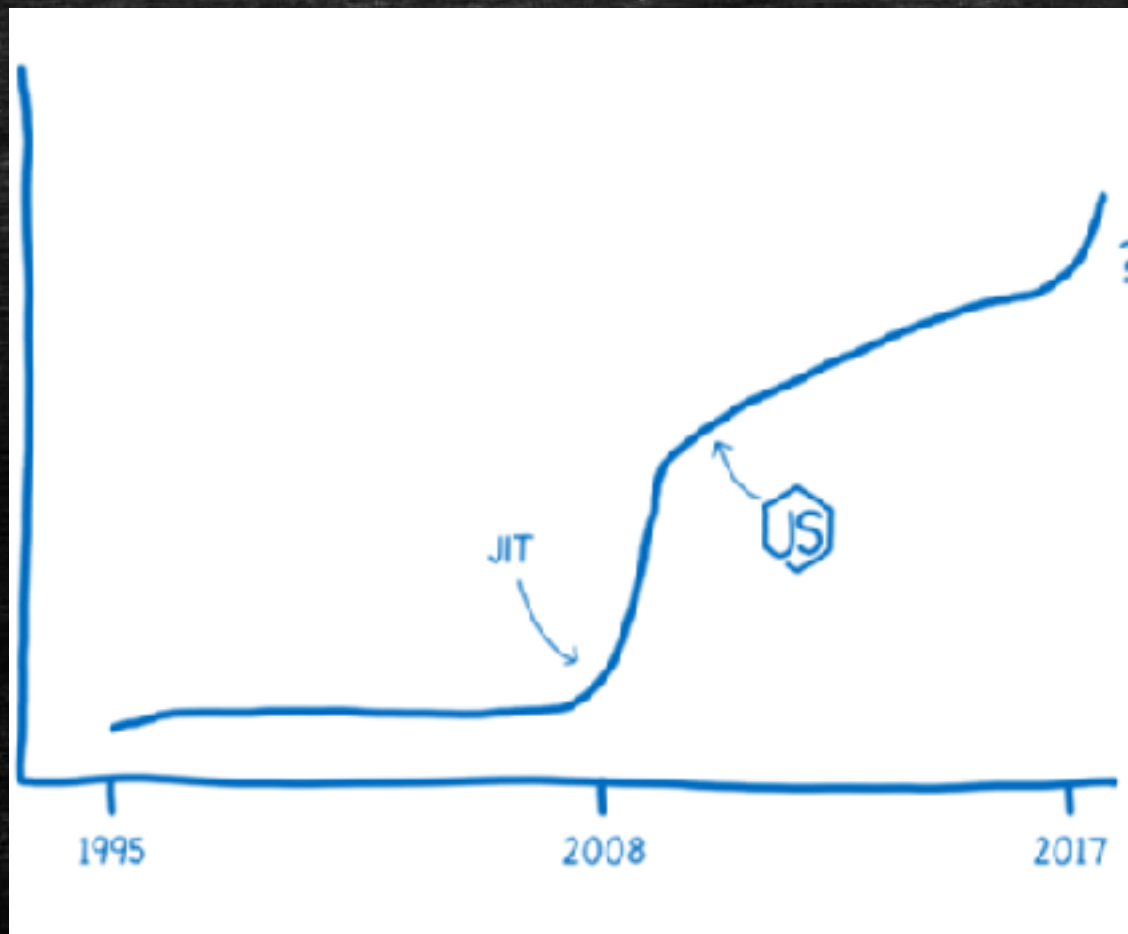


渲染逻辑 WebGL

如何进一步提升 JavaScript 的运行效率



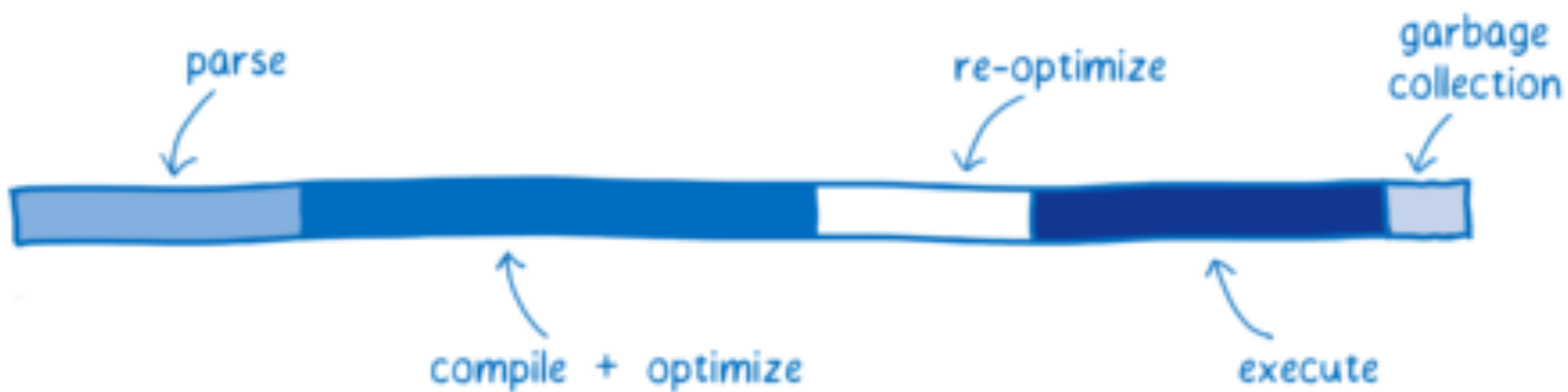
如何进一步提升 JavaScript 的运行效率



如何进一步提升 JavaScript 的运行效率

```
1  sum(1, 2);  
2  
3  function sum(a, b) {  
4      |    return a + b;  
5  }
```

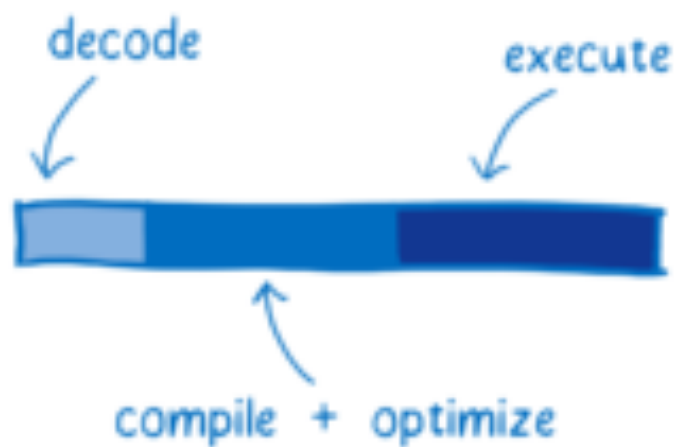

如何进一步提升 JavaScript 的运行效率



如何进一步提升 JavaScript 的运行效率

```
1    sum(1, 2);  
2    sum("1", "2");  
3  
4    function sum(a, b) {  
5        |    return a + b;  
6    }
```


如何进一步提升 JavaScript 的运行效率

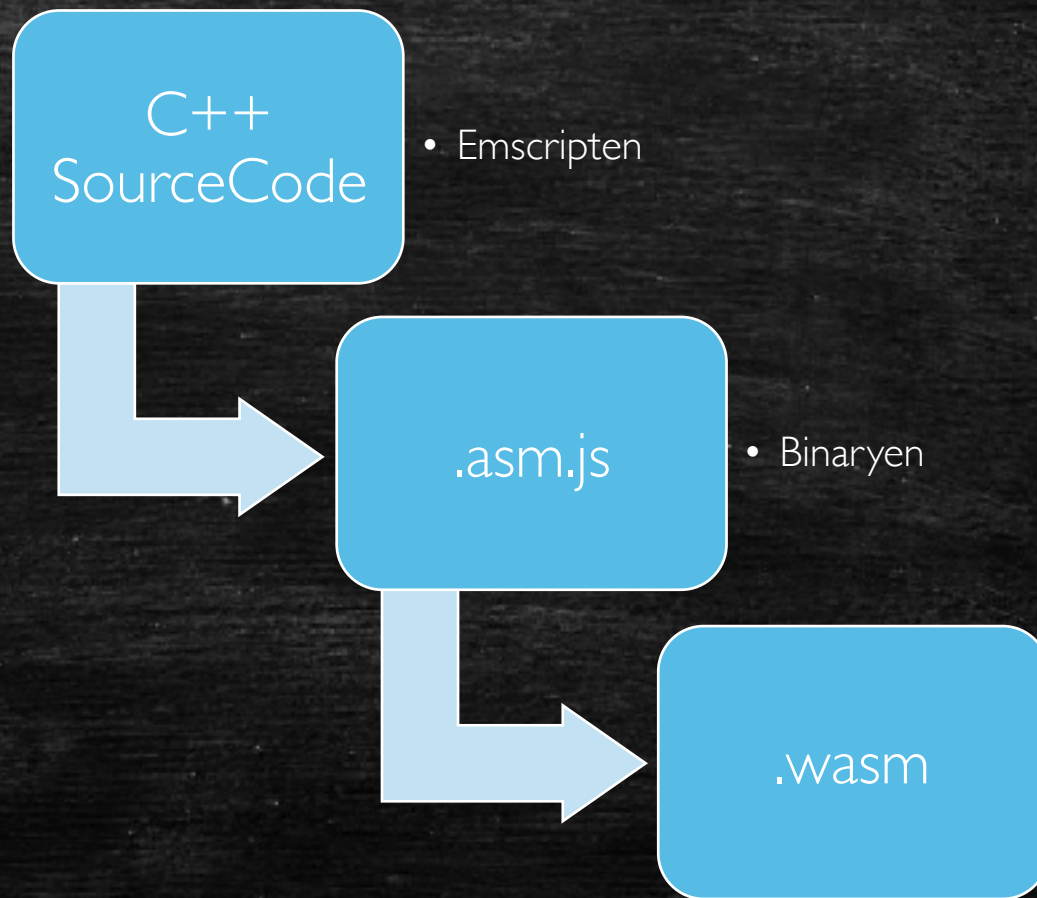


如何进一步提升 JavaScript 的运行效率

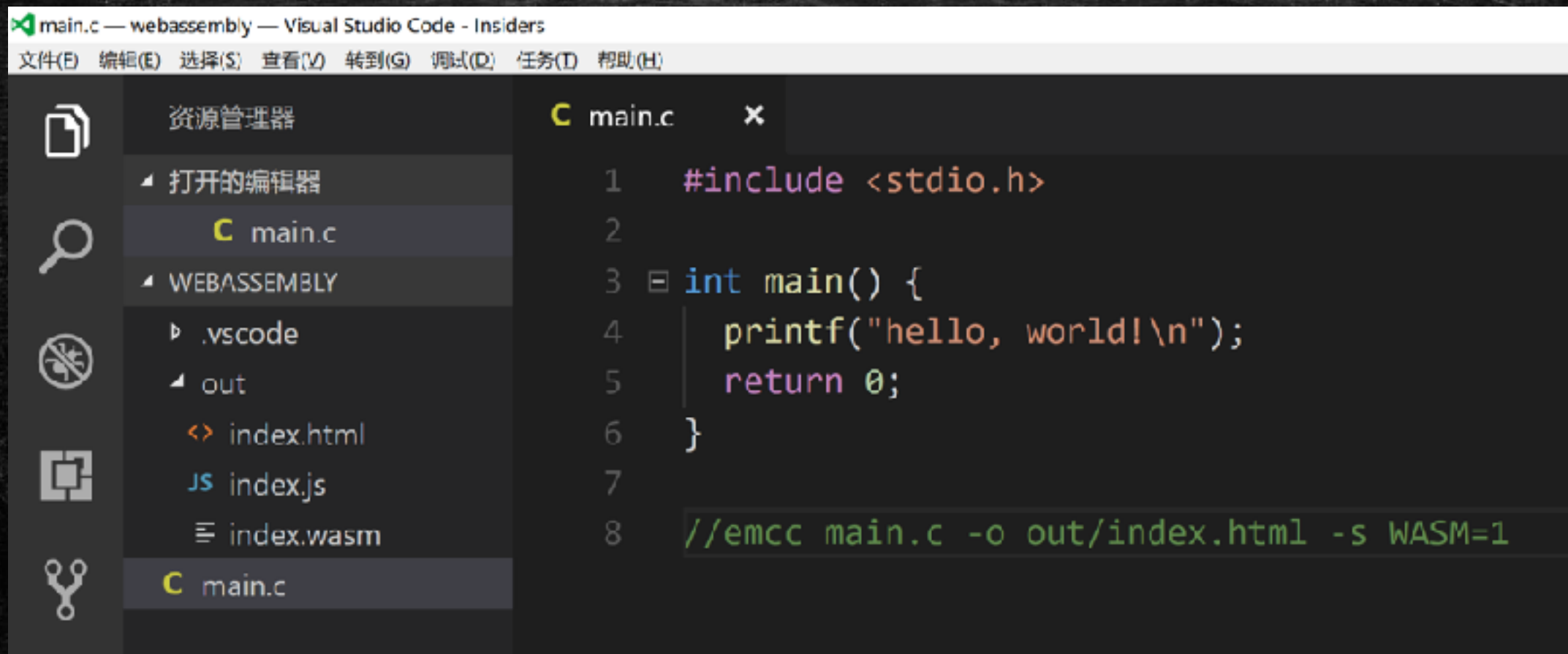
- 面向机器，而非面向开发者
- 强类型，而非运行时推断类型
- 使用更可控的垃圾回收机制

WebAssembly 架构

- WebAssembly 运行时
- 编译器 – Emscripten
- 优化器 - Binaryen



WebAssembly 架构



The screenshot shows the Visual Studio Code editor with a project named 'webassembly'. The left sidebar displays the 'Resource Manager' (资源管理器) with a tree view containing folders like '.vscode' and 'out', and files like 'index.html', 'index.js', 'index.wasm', and 'main.c'. The 'main.c' file is selected. The main editor area shows the code for 'main.c' with the following content:

```
1  #include <stdio.h>
2
3  int main() {
4      printf("hello, world!\n");
5      return 0;
6  }
7
8  //emcc main.c -o out/index.html -s WASM=1
```


WebAssembly 在游戏引擎中的应用

- Native浏览器插件
- JavaScript +HTML5 API
- Native Code to WebAssembly
- JavaScript API + WebAssembly Core

WebAssembly 在游戏引擎中的应用

- 白鹭引擎对外提供 JavaScript API
- 开发者编写的 JavaScript 逻辑代码会汇总为一组命令队列发送给 WebAssembly 层
- 然后 WebAssembly 建立对渲染节点的抽象封装，并在每一帧对这些渲染节点进行矩阵计算、渲染命令生成等逻辑
- 生成一组 ArrayBuffer 数据流
- 最后 JavaScript 对这组数据流进行简单的解析并直接调用 DOM 的 WebGL 接口

WebAssembly 在游戏引擎中的应用

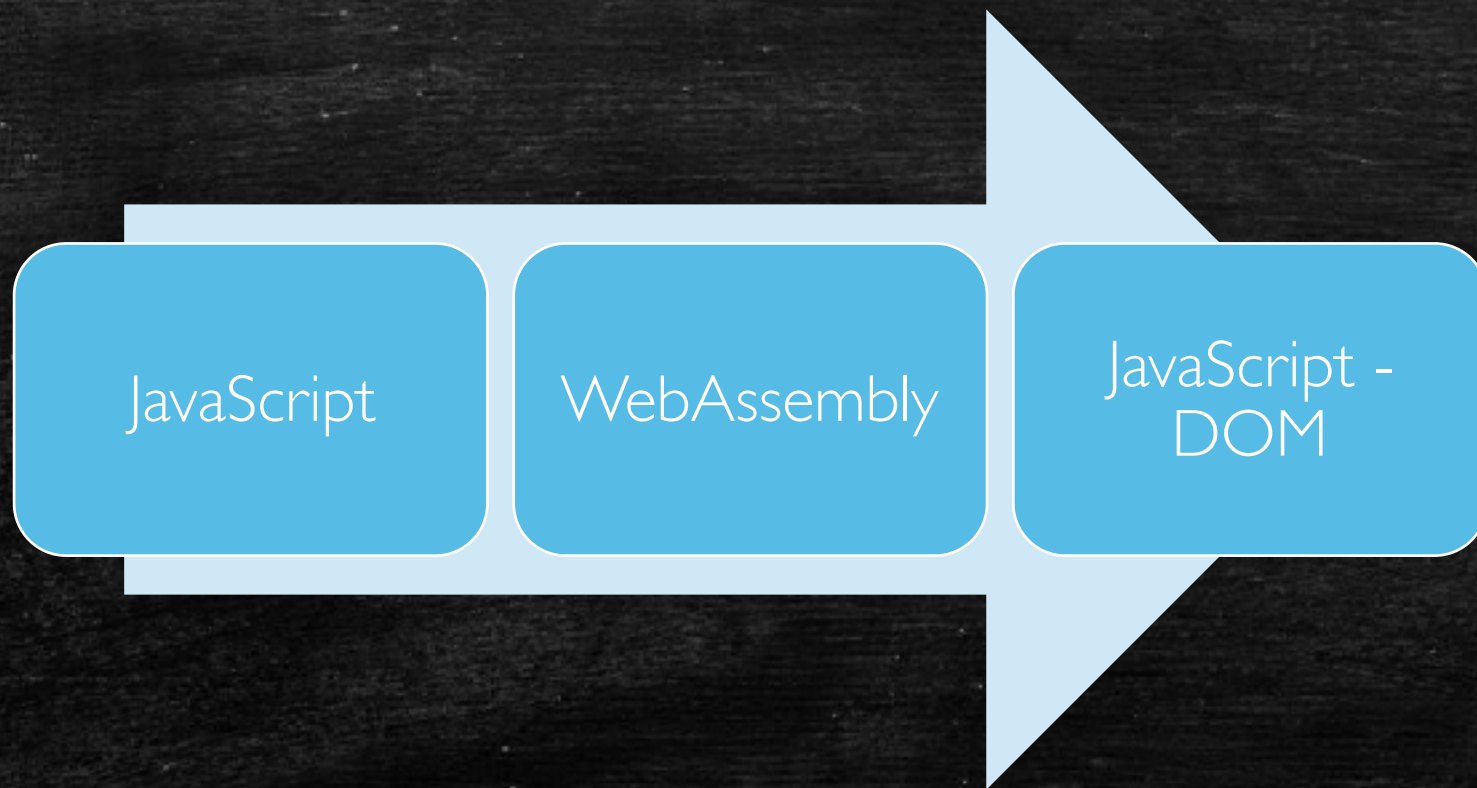
- 基准测试性能比 asm.js 提升 30%
- 代码尺寸降低约30%
- 不支持的设备自动回退到 asm.js 版本

坑

- 交互问题
- 内存问题

坑 – 交互问题

- WebAssembly 无法操作 DOM 元素



坑 – 交互问题

- JavaScript 与 WebAssembly 的对象互相调用的性能很差
- 简单的将特定几个函数编译为 WebAssembly，然后交由 JavaScript 去调用的方式反而会因为频繁的互相操作反而造成性能下降
- 进行一次性的密集计算
- 使用 ArrayBuffer 共享

坑 – 内存问题

- WebAssembly 不存在垃圾回收机制
- JavaScript 开发者已经习惯于垃圾回收机制

建议

- 把密集型大量计算工作交由 WebAssembly 解决
- 不建议初期在 Emscripten 中使用 html5.h 等高级特性
- 不建议前端工程师现在就去学习 WebAssembly

广告时间....



- 高级前端研发工程师（桌面客户端方向）
- 高级前端开发工程师（移动 Web 方向）
- 技术支持顾问（游戏开发方向）
- C++开发工程师（游戏开发方向）