資工導論報告—WebAssembly

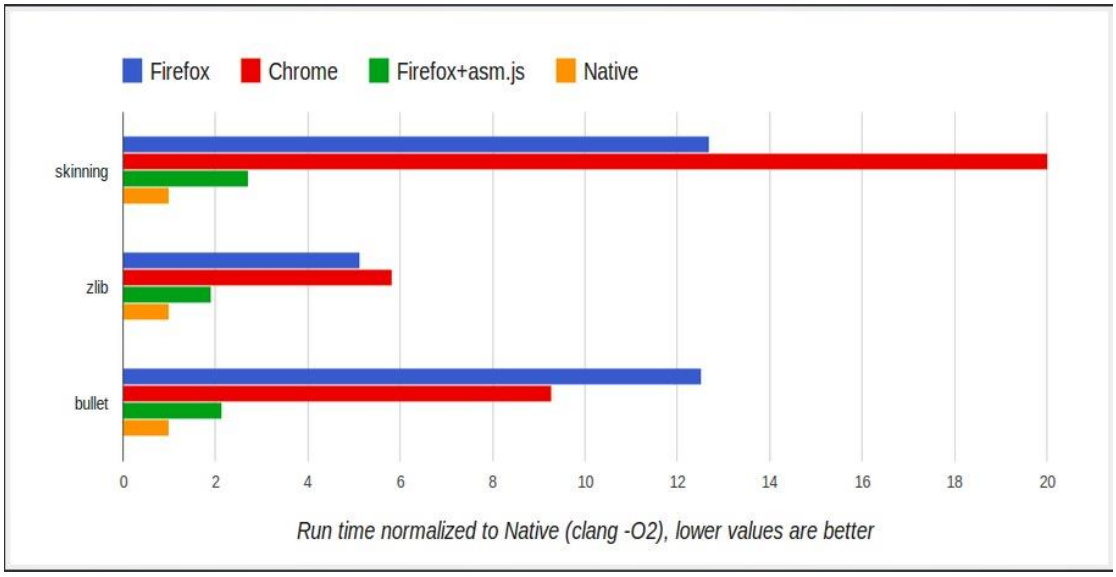
B10815057 廖聖郝

WebAssembly 或稱 wasm 是一種新的低階程式語言，應用於瀏覽器內，被設計來提供比 JavaScript 更快速的編譯及執行，WebAssembly 將讓開發者能運用自己熟悉的程式語言編譯，支援 Rust、C/C++、C#、.Net、Java、Python、Elixir、Go...，再藉虛擬機器引擎在瀏覽器內執行，WebAssembly 於 2019 年 12 月 5 日成為全球資訊網協會（W3C）的推薦，與 HTML，CSS 和 JavaScript 一起，成為 Web 的第四種語言，2017 年 11 月四大瀏覽器 Firefox、Chrome、Microsoft Edge、Safari，WebAssembly 也被設計來與 JavaScript 協同工作，使用 WebAssembly 的 JavaScript API，可以把 WebAssembly 載入至 JavaScript 程式，兩者間共享功能 LLVM。

1.webassembly 的發展背景

JavaScript 是為了補足 html 過於靜態的缺點所發展出來的動態直譯式語言，但

javascript 也有其缺點，如: 執行效能遠不如靜態程式語言，許多高效能需求的程式，如 3D 即時遊戲，影音剪輯，都難以單靠 javascript 實作，2012 年，Mozilla 的工程師突發奇想，如果把 C/C++編譯成 javascipt，就能在瀏覽器中運行，因此有了 Emscripten 編譯器，這個編譯器可以將 C/C++轉換成一種名為 asm.js 的 javascript 子集，透過各種限制，避開 JS 中最難最佳化的部分，讓程式可以接近原生(native)的執行效能，asm.js 就是 webassembly 的前身。



2.瀏覽器大戰的終結

(1)第一次瀏覽器大戰:

網景（Mosaic Netscape）在 1994 年 10 月發行第一版網景瀏覽器，發表後四個月就佔據了 75%的市場，網景開發了 JavaScript、制定安全通訊協定(SSL)、cookie，1995 年微軟推出 Internet Explorer，透過綑綁在 Windows 內讓使用者免費使用，網景無法對抗微軟的「免費程式」策略，因此於 1999 年被 AOL 收購，1998 年網景開始 Mozilla 開放原始碼計畫，但後來發現繼續開發有很多困

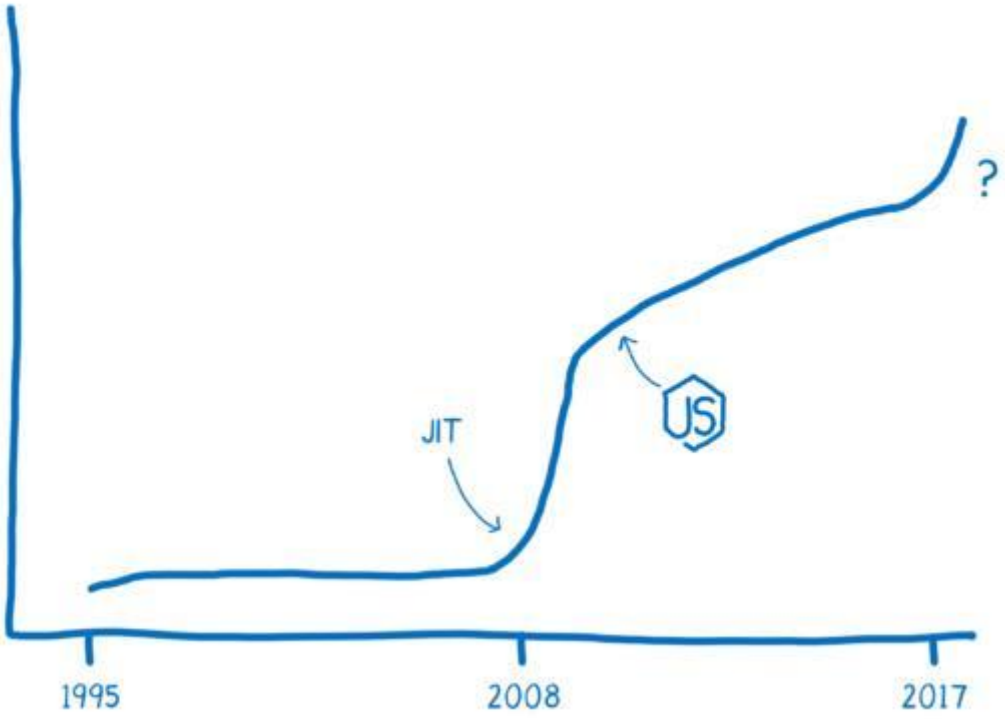
難，於是 Mozilla 放棄原始碼，開發出 Geoko 排版引擎作為新一代開源瀏覽器：「Phoenix」，也就是 Firefox 的前身，剛發佈幾年人氣快速竄升到 30% 市佔率，使微軟不得不重起 IE 開發專案來拯救 IE 6。

(2)第二次瀏覽器大戰:

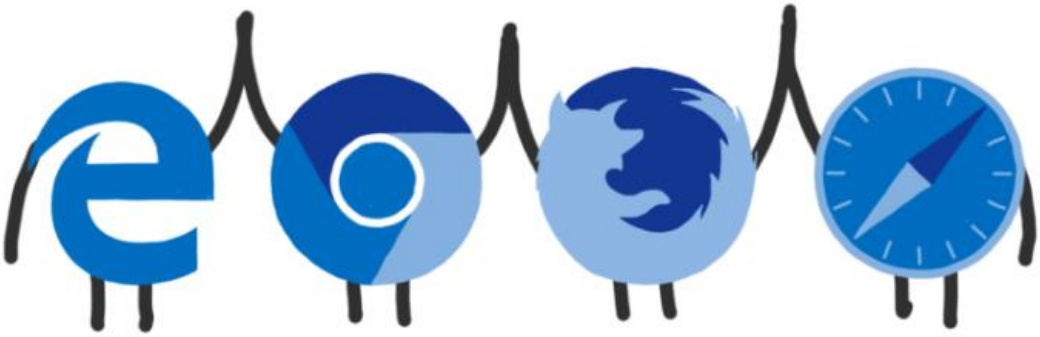
2008 年 Google 推出 Chrome 瀏覽器，其卓越的性能、簡潔的介面以及捆綁 Google 搜尋的優勢，脫胎於 KHTML 內核的 V8 引擎基於新興 WebKit 內核開發，透過減少 JavaScript 進行屬性存取時在記憶體動態搜尋、降低由垃圾收集造成的停頓、優化直譯器對於 JavaScript 的即時編譯，也帶來更節能、順暢的網際網路瀏覽體驗，快速成長，除了侵蝕屬於 Firefox 的市場之外，也同時重創古老的微軟 IE，到了 2012 年 Google Chrome 超越 IE，成為全球第一大瀏覽器。

Chrome 的風行帶起了標準化網頁規範風潮，眾多瀏覽器開始以採用 HTML5、CSS3 等共通且開放的標準網頁規範，作為瀏覽器核心標準。

此時 IE 瀏覽器，成了大家調侃的「瀏覽器下載器」。過多的安全性漏洞與對新技術支援性差等等原因，迫使微軟放棄現有 IE 的，開發了新的瀏覽器：Microsoft Edge。



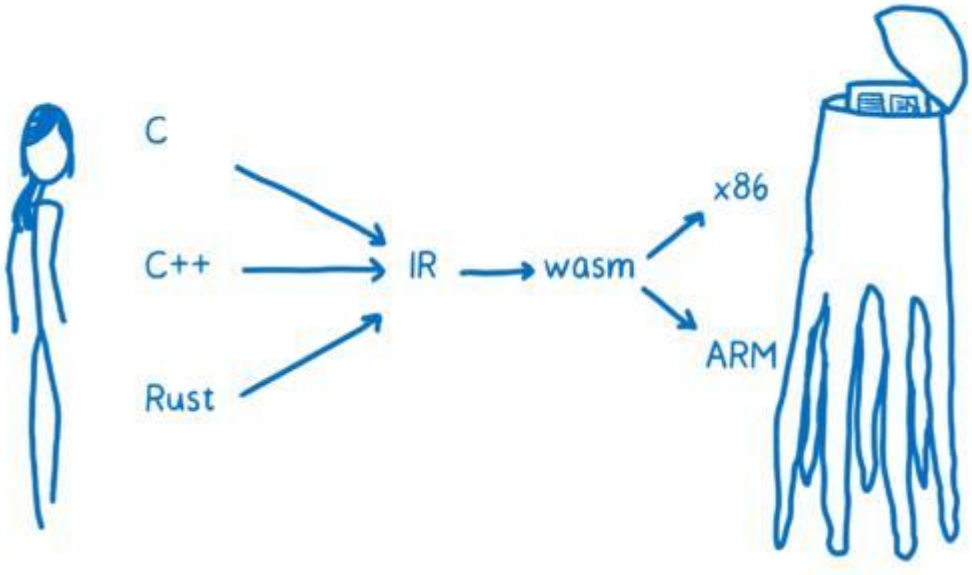
為了解決 javascipt 的技術瓶頸，各家廠商紛紛研發各自的解決方案，例如:google v8 引擎採用的 JIT(just in time) 編譯器、google 開發的 dart 語言、微軟的 typescript、firefox 的 asm.js，雖然方案眾多，但最終 Mozilla, Google, Microsoft, and Apple 覺得 Asm.js 这个方法有前途，因此將 asm.js 標準化，讓大家都能用，所以誕生了 webassembly。



3.原理

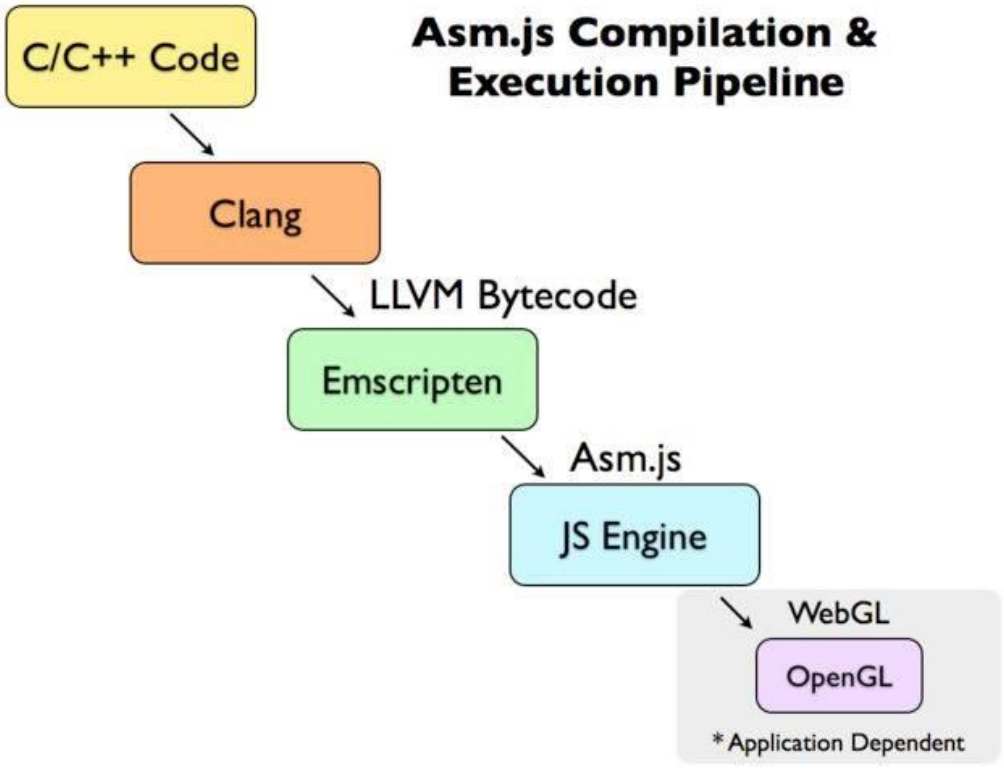
(1)WebAssembly 處於哪個環節?

高級語言(C/C++)用編譯器轉換成中介碼後再進一步轉成 X86、arm 架構的組合語言，WebAssembly 就像是另一種的目標組合語言，但 webassembly 與一般組合語言不同，它不依賴於實際的機器，而更像是虛擬機器的機器語言



(2)編譯成.wasm 檔案

Emscripten 將 C/C++轉換為 asm.js。



WebAssembly 是編譯器生成的主要目標，所以它的運行主要包含兩個部分：生成它的編譯器和運行它的虛擬機。

想從 C 語言到 WebAssembly，我們就需要 clang 前端來把 C 程式碼變成 LLVM 中間程式碼。它會對程式碼自動地做一些優化。為了從 LLVM 中間程式碼 生成 WebAssembly，還需要後端編譯器。在 LLVM 的工程中有正在開發中的後端，而且應該很快就開發完成了，但現在暫時還看不到它是如何起作用的。

Emscripten 還包含了許多額外的工具和庫來包容整個 C/C++ 程式碼庫，所以它更像是一個軟體開發者工具包(SDK)而不是編譯器。例如系統開發者需要檔案

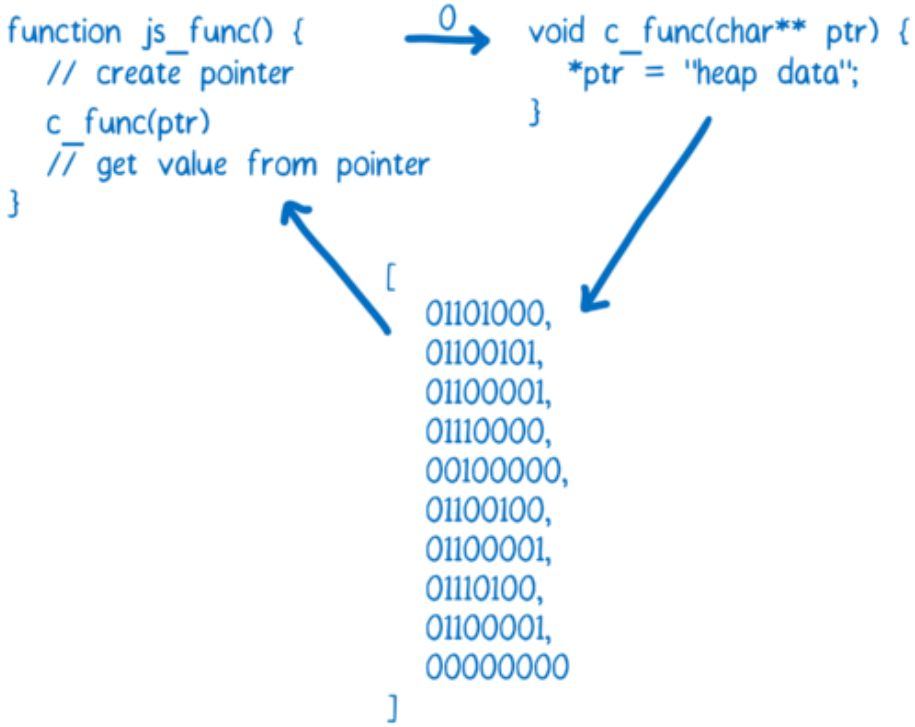
系統以對檔案進行讀寫，Emscripten 就有一個 IndexedDB 來模擬檔案系統

(3)加載一個.wasm 模組到 JavaScript

當前的 WebAssembly 只能使用數字(整數或者浮點數)作為參數或者回傳值。對於任何其他的複雜類型，比如字串，就必須得用 WebAssembly 模組的內存操作了。

為了實現這個功能，它使用了 JavaScript 中稱為 ArrayBuffer 的資料結構。ArrayBuffer 是一個字節數組，所以它的索引(index)就相當於內存地址了。

如果你想在 JavaScript 和 WebAssembly 之間傳遞字串，可以利用 ArrayBuffer 將其寫入內存中，這時候 ArrayBuffer 的索引就是整數了，可以把它傳遞給 WebAssembly 函數。此時，第一個字符的索引就可以當作指標來使用。



這就好像在開發 WebAssembly 模組時，把這個模組包裝了一層外衣。這樣其

他使用者在使用這個模組的時候，就不用關心內存管理的細節。

(4).wasm 檔案結構

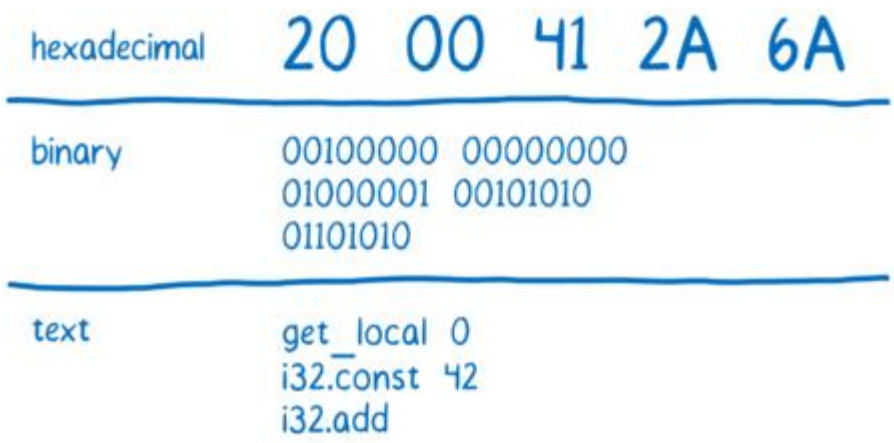
這段代碼是即將生成 WebAssembly 的 C 代碼：

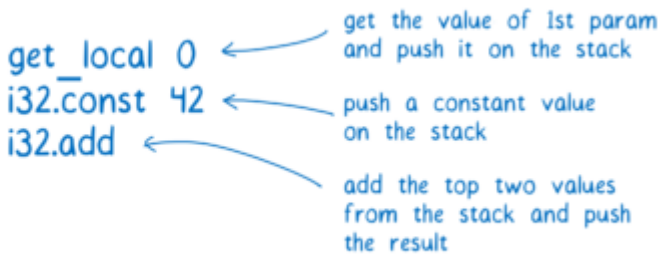
```
int add42(int num) {
    return num + 42;
}
```

使用 WASM Explorer 來編譯，打開.wasm 檔案，可以看到以下:

```
00 61 73 6D 0D 00 00 00 01 86 80 80 80 00 01 60
01 7F 01 7F 03 82 80 80 80 00 01 00 04 84 80 80
80 00 01 70 00 00 05 83 80 80 80 00 01 00 01 06
81 80 80 80 00 00 07 96 80 80 80 00 02 06 6D 65
6D 6F 72 79 02 00 09 5F 5A 35 61 64 64 34 32 69
00 00 0A 8D 80 80 80 00 01 87 80 80 80 00 00 20
00 41 2A 6A 0B
```

例如，下面是 num + 42 的各種表示方法。





從圖中我們可以注意到加法操作並沒有指定哪兩個數字進行相加。這是因為 WebAssembly 是採用基於 stack 的虛擬機的機制。意思是一個運算子所需要的所有值，在操作進行之前都已經存放在 stack 中。

所有的運算子，比如加法，都知道自己需要多少個值。加需要兩個值，所以它從 stack 頂部取兩個值就可以了。那麼加法指令就可以變的更短，因為指令不需要指定來源暫存器和目的暫存器。這也使得.wasm 檔案變得更小，進而使得加載.wasm 檔案更快。

儘管 WebAssembly 使用基於 stack 的虛擬機，但是並不是說在實際的物理機器上它就是這麼運作的。當瀏覽器翻譯 WebAssembly 到機器碼時，瀏覽器會使用暫存器，而 WebAssembly 程式碼並不指定用哪些暫存器，這樣做的好處是給瀏覽器最大的自由度，讓其自己來進行暫存器的最佳分配。

4.優點

(1)比起 JavaScript 的處理速度可以快上 20 倍

(2)可透過多種語言編寫

(3)體積小 由於瀏覽器運行時只加載編譯後的字節碼，一樣的邏輯比字串描述的

JS 檔案體積小很多。

(4)加載快 由於檔案體積小，再加上無需解釋執行，webAssembly 能更快的加載並實例化，減少運行前的等待時間。

(5)相容問題少 webAssembly 是非常底層的字節碼規範，制定好以後很少變動，就算發生變化，只需要從高級語言編譯成字節碼過程做相容。可能出現兼容問題是 JS 和 webAssembly 橋接的 JS 接口。

5.缺點

與 WebAssembly 相比，JavaScript 可以訪問更多的 API。 DOM（文檔對像模型）是這些 API 中最重要的。 基本上，為了更改網頁或對應用戶輸入，WebAssembly 必須透過 Javascript 運作。

WebAssembly 沒有 JavaScript 具有的一些功能，例如字串和垃圾回收。 這可以通過將任何語言編譯成 WebAssembly 進行管理，但是垃圾回收代碼是下載程式的一部分，而不是環境，這會增加下載程式的大小。

現在的 WebAssembly 還必須配合“JavaScript glue code”來使用，也就是必須使用 JavaScript 來 fetch WebAssembly 的檔案，然後調用 window.WebAssembly.instantiate、window.WebAssembly.instantiateStreaming 等函數進行實例化。部分情況下還需要 JavaScript 來管理堆棧。官方推薦的編譯工具 Emscripten 雖然使用了各種技術來縮小編譯後生成的代碼的數量，但是最終生成的 JavaScript

Glue Code 檔案還是至少有 15KB。

6.應用

網頁小遊戲:https://sandspiel.club/

網頁小遊戲:http://asciicker.com/y2/

Google earth: https://www.google.com/intl/zh-TW/earth/

線上 UI 多人協作工具:https://www.figma.com/

跨平台的 OpenGL 圖形引擎 Magnum:https:/github.com/mosra/magnum

Egret Engine HTML 5 遊戲引擎: https://github.com/egret-labs/egret-core/

Blazor 讓 .NET 程式碼也能在瀏覽器運行:

https://github.com/SteveSanderson/Blazor

7.未來展望

未來，WebAssembly 將可能直接通過 HTML 標簽進行引用，比如：

<script src="./wa.wasm"></script>；或者可以通過 JavaScript ES6 模塊的方式引用，比如：import xxx from './wa.wasm';

WebAssembly 的出現，使得前端不再只能使用 JavaScript 進行開發了，C、C++、Go 等等都可以為瀏覽器前端貢獻代碼。

WebAssembly 的出現不是要取代 JavaScript，而是與 JavaScript 相輔相成，為前端開發帶來一種新的選擇。將計算密集型的部分交給 WebAssembly 來處理，讓瀏覽器發揮出最大的性能！

以下是將要實作的功能:

- 制定規格
- 執行緒
- 固定長度的 SIMD
- 例外處理
- 垃圾回收
- 記憶體區塊操作
- 網頁內容安全性政策
- ECMAScript 模組整合
- 尾端呼叫
- Non-trapping 浮點數-整數轉換
- 多值函式
- Host bindings