打破C/C++与Web世界的隔阂
构建面向未来的极速Web应用

Broadview
www.broadview.com.cn

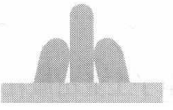
深入浅出 WebAssembly

于航 / 著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

深入浅出 WebAssembly

于航 / 著



電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

WebAssembly 是一种新的二进制格式，它可以方便地将 C/C++等静态语言的代码快速地“运行”在浏览器中，这一特性为前端密集计算场景提供了无限可能。不仅如此，通过 WebAssembly 技术，我们还可以将基于 Unity 等游戏引擎开发的大型游戏快速地移植到 Web 端。WebAssembly 技术现在已经被计划设计成 W3C 的标准，众多浏览器厂商已经提供了对其 MVP 版本标准的支持。在 Google I/O 2017 大会上，Google 首次针对 WebAssembly 技术进行了公开演讲和推广，其 Post-MVP 版本标准更是对诸如 DOM 操作、多线程和 GC 等特性提供了支持。WebAssembly 所带来的 Web 技术变革势不可挡。

本书力求从一些简单的实践入手，深入理论，到复杂的具有实际业务价值的综合实践，深入浅出地介绍 Wasm 技术发展至今，其背后所涉及的各种底层设计原理与实现、相关工具链以及未来发展方向等多方面内容。本书内容包括：WebAssembly 技术的发展历程，从 PNaCl 到 ASM.js 再到 WebAssembly，以及这些技术的基本应用方法与性能对比；WebAssembly 的标准上层 API、底层堆栈机的设计原理，以及对 MVP 标准理论的深入解读；与 WebAssembly 标准相关的进阶内容，如单指令多数据流（SIMD）、动态链接（DL）等；LLVM 工具链与 WAT 可读文本格式的相关内容；基于 Emscripten 工具链开发 WebAssembly 应用的基本流程，以及工具链的一些基本常用功能和特性；基于 Emscripten 工具链实现 C/C++语言动态关系绑定技术；Emscripten 工具链所提供的一些如 WebGL 支持、虚拟文件系统、应用优化以及 HTML 5 事件系统等高级应用特性；构建一个具有实际业务价值的 WebAssembly 应用，现阶段 Wasm 生态的发展情况，以及在 Post-MVP 标准中制订的一些 WebAssembly 未来发展规划。

本书的目标读者为 Web 前端开发人员、C/C++开发人员和对 WebAssembly 技术感兴趣的人员。

图书在版编目（CIP）数据

深入浅出 WebAssembly / 于航著. —北京：电子工业出版社，2018.12
ISBN 978-7-121-35217-1

Ⅰ. ①深… Ⅱ. ①于… Ⅲ. ①编译软件 Ⅳ. ①TP314

中国版本图书馆 CIP 数据核字（2018）第 238873 号

策划编辑：张春雨
责任编辑：葛 娜
印 刷：北京京科印刷有限公司
装 订：北京京科印刷有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编：100036
开 本：787×980 1/16 印张：34.25 字数：741 千字
版 次：2018 年 12 月第 1 版
印 次：2018 年 12 月第 1 次印刷
定 价：128.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）51260888-819，faq@phei.com.cn。

序言（一）

I'm very excited to see this book, which covers in great detail a wide range of topics regarding WebAssembly. At this point in time WebAssembly is around one year old - if we count from when it shipped in all major browsers - so it's still fairly young, and the industry is just starting to figure out how revolutionary it is going to be. The potential is there for huge impact, and good documentation is necessary for that.

Why is WebAssembly's potential impact so large? For several reasons:

- **WebAssembly helps make the Web fast**: WebAssembly is designed for small download size, fast startup, and predictably fast execution. The improvement compared to JavaScript can be very significant, over 2x in many cases, and especially in startup, where the speedup can be 10x.

- **WebAssembly makes the Web competitive with native**: WebAssembly is designed as a compiler target for multiple languages. That includes C and C++, and in many areas of software the best implementations are in those languages, for example, game engines like Unity and Unreal, design software like AutoCAD, etc. It would take many years to write comparable products in JavaScript; instead, by compiling them to WebAssembly, the Web can be on par with native platforms today.

- **WebAssembly also fills an industry need outside the Web**: WebAssembly is fast, portable, sandboxed, has multiple excellent open source implementations, and just like the Web itself it is an industry standard expected to be supported for the long term. As a result, it's not surprising that WebAssembly is starting to be used outside of browsers, for example in the blockchain and content delivery network (CDN) spaces.

Looking back, it's remarkable that our industry has gotten to this point. Just a few years ago, there was no cross-browser collaboration on getting native code to run on the Web. Instead, there were multiple options, including Native Client, Adobe Alchemy, and ASM.js, each with its own advantages and disadvantages. I believe it was the momentum of ASM.js that got the industry to focus on fixing

things: ASM.js started out in Firefox, and by virtue of being a subset of JavaScript it immediately ran in all browsers - just not as efficiently. That led top companies in the video game industry and elsewhere to adopt ASM.js, together with Emscripten, the open source compiler to JavaScript that I started in 2010, and which could emit ASM.js. That adoption led to ASM.js support in Edge and later Chrome, at which point there was consensus that the industry should produce a proper standard in this space, which turned into WebAssembly. As the spec was designed and implementations started to appear, we added WebAssembly support to Emscripten, which allowed people to compile to both ASM.js and WebAssembly by just flipping a switch, making it easy for people to use the new technology. Finally, as of May 2018 Emscripten emits WebAssembly by default, and today WebAssembly has robust and stable support both in all major browsers and in the toolchain projects that emit it.

It's been a complicated path to get here, but the future looks bright. It is especially worth noting that WebAssembly is expected to add features like multithreading, SIMD, GC, and others, which will open up even more interesting opportunities.

Alon Zakai

Alon is a researcher at Mozilla, where he works on compile-to-Web technologies. Alon co-created WebAssembly and ASM.js, and created the Emscripten and Binaryen open source projects which are part of the primary WebAssembly compiler toolchain.

译文：

我很高兴能够看到这本书的出版，作者在书中详细地介绍了有关 WebAssembly 的各种主题。在这本书即将出版之际，WebAssembly 差不多一岁了——如果从所有主流的 Web 浏览器开始支持 WebAssembly 算起，那么这项技术仍然相当年轻，业界也才刚刚开始意识到它将多么具有革命性。WebAssembly 所拥有的潜力将会在未来对 IT 行业产生巨大的影响，但在此之前，我们需要有优秀的文档。

为什么 WebAssembly 的潜在影响力会如此之大？有以下几个原因。

- WebAssembly 让 Web 应用运行更快。WebAssembly 是一种新的格式，文件体积更小，启动速度更快，运行速度也更快。与使用 JavaScript 构建的 Web 应用相比，性能提升非常明显。在大部分情况下，运行速度提升两倍以上，特别是在启动速度方面，速度提升

可以达到 10 倍。

- WebAssembly 让 Web 应用能够与原生应用展开竞争。WebAssembly 是多种编程语言的编译器目标，包括 C 和 C++。基于这些编程语言实现的优秀软件，如游戏引擎 Unity、Unreal，设计软件 AutoCAD 等，如果使用 JavaScript 开发在功能上与这些软件旗鼓相当的产品可能需要很多年时间。但如果将它们编译成 WebAssembly，这些原生应用就可以直接运行在 Web 平台上。因此，Web 能够与原生平台相提并论。

- WebAssembly 还在 Web 领域之外为行业带来了其他可能性。WebAssembly 运行速度快、可移植，提供了沙箱机制，并拥有众多优秀的开源实现，就像 Web 本身一样，它将会是一个被长期支持的行业标准。因此，WebAssembly 开始被应用在 Web 浏览器之外的领域也就不足为奇了，例如区块链和内容分发网络（CDN）。

回首过去，我们的行业能够取得如此的成就已经很了不起了。几年前，还没有人去进行这种跨浏览器协作，以便让原生代码运行在 Web 平台上。不过有很多不同的项目，如 Native Client、Adobe Alchemy 和 ASM.js，它们都在尝试做同样的事情，只是每个项目都有各自的优缺点。而我认为，是 ASM.js 的出现让业界开始专注于解决这个问题——ASM.js 最初出现在 Firefox 中，由于它是 JavaScript 的一个子集，因此可以无缝地运行在所有浏览器中，但运行效率不高。视频游戏等行业的一些顶级的公司开始尝试使用 ASM.js 和 Emscripten（我在 2010 年开源的编译器工具链，可以将代码编译成 ASM.js）。由于在这些领域的广泛应用，Edge 以及后来的 Chrome 均开始支持 ASM.js。此时，人们一致认为这个领域需要一个行业标准，于是 WebAssembly 出现了。

随着规范设计和实现的不断演进，我们在 Emscripten 中加入了 WebAssembly 支持——只需要在编译命令中加入一个“开关”，便可选择性地将编译目标设置为 ASM.js 或 WebAssembly，从而可以更轻松地使用这项新技术。截至 2018 年 5 月，Emscripten 已经将默认的编译目标类型改为 WebAssembly。今天，WebAssembly 已经在所有主流浏览器和工具链项目中得到了强大而稳定的支持。

一路走来历经坎坷，但未来是光明的。特别值得注意的是，WebAssembly 将会在未来添加多线程、SIMD、GC 等功能，而这些新特性将会为我们带来更多有趣的可能性。

Alon Zakai

Alon 是 Mozilla 的研究员，从事与“编译到 Web 平台”相关的研究工作。Alon 参与制定了 WebAssembly 和 ASM.js 标准，并创建了 Emscripten 和 Binaryen 等开源项目，这些项目都是 WebAssembly 编译器工具链的重要组成部分。

序言（二）

前端的可玩性变得越来越高，也越来越开放了。现如今，我们不仅仅能够使用 HTML、CSS 及 Javascript 来编写各种跨端的应用程序，WebAssembly 的出现还让我们能够以极小的成本来复用其他领域已存在的成果，以此来弥补 JavaScript 在其性能与功能上的不足。

我第一次了解到 WebAssembly 是在 2017 年年初，当时沉迷于想自己制作一个基于 Node.js 环境和树莓派的语音助手。可惜对于语音处理这个领域来说，JavaScript 还是一个“新人”，大量成熟的实现成果主要集中在 C/C++领域。因此，对于当时对 Node.js 扩展及 C/C++了解甚少的我来说，这是难度颇大的一个门槛。后来通过 Twitter 我了解到 WebAssembly 的前身是 ASM.js，于是我立即尝试使用 Emscripten 将 Google Assistant 的 Linux SDK 编译为 ASM.js，并顺利地在 Node.js 环境中进行了调用，那份喜悦我记忆犹新，同时这也极大地提升了我对这项技术的信心和好感。之后在全民直播的技术提升项目中，我与另一位研发人员有幸一起对最核心的播放器组件编解码和弹幕协议加密部分进行 WebAssembly 化，并成功上线且获得了极大的性能提升。在此之后，我坚信 WebAseembly 在未来一定会大有可为。

由于 WebAssembly 是一项极新的技术，因此在最初学习 WebAssembly 的过程中常常觉得知识零碎且不成体系，经常会出现浮沙驻高塔的情形，感觉入门十分困难。好在本书的出版，让这种情形不再复现。这本书的好处就是它系统详细地讲述了 WebAssembly 的方方面面，由浅入深地构建了整个 WebAssembly 的知识体系。不管你是刚接触 WebAssembly 的新人，还是已经在工作场景中使用 WebAssembly 的“老鸟”，通过阅读这本书都能够得到极大的提升。总之，如果你想了解 WebAssembly，或者想补足相关的知识体系，它都是一本不可多得的案头好书。风雨欲来，如果现在还不进行 WebAssembly 的技术储备，更待何时？

最后，我要感谢于航让我第一时间读到如此精彩的作品，同时也感谢他对 WebAssembly 在国内的布道普及所做的工作，我相信 WebAssembly 的未来一定会更加美好，Web 的未来也会更加开放和美好。

赵洋

赵洋是“全民直播”的前端研发经理，曾经主导全民直播播放器编解码核心模块及弹幕协议加密过程 WebAssembly 化。

前　言

为什么要写这本书

自从 JavaScript（后面简称 JS）脚本语言于 1995 年诞生以来，人们便一直在使用该语言以及 HTML 与 CSS 来编写和开发以浏览器为主的 Web 应用。近年来，随着 JS 的不断流行，以及 Node.js 的出现，JS 也开始逐渐向除 Web 前端之外的其他领域发力，比如开发后端应用、机器学习应用乃至硬件编程等领域。但就 JS 本身而言，所不能无视的是它是一种弱类型语言，因此，相比于 C/C++等强类型语言，尽管 Chrome V8、SpiderMonkey 等 JS 引擎已经通过诸如 JIT 等多种技术手段来优化 JS 脚本代码的整体执行效率，但引擎每一次版本优化的迭代速度（所花费的时间）却远远跟不上当今各类 Web 应用的复杂程度变化。因此，发明一种能够从根本上解决该问题的技术便显得迫在眉睫。

曾昙花一现的 ASM.js、NaCl 与 PNaCl 等技术都尝试以其各自的方式来优化 Web 应用的执行效率，但由于其所存在的诸如“浏览器兼容性不佳”以及“性能优化不彻底”等问题，导致它们最终并没有被广泛推广。而在 2015 年出现的 WebAssembly（简称 Wasm）技术，便是在吸取了前者经验教训的基础上而被设计和发明出来的。现在，我们可以看到该项技术所具有的潜力——W3C 成立了专门的 WWG 工作组来负责 Wasm 技术的标准迭代与实现，四大主流浏览器（Google Chrome、FireFox、Edge 和 Safari）已经全部实现 WebAssembly 技术在其 MVP 标准中制定的所有特性，C/C++、Go 和 Rust 等高级语言已经逐渐开始支持编译到 Wasm 格式。这一系列的发展和变化都说明了人们对该项技术所寄予的厚望。

如今世间百态，万物的发展速度越来越快，而前端技术领域也同样如此，正在转向技术融合的道路——从 2000 年专门指代 PC 网页技术的 Web 前端，到 2010 年左右包含有 H5 技术的前端，再到融合了移动端甚至部分后端技术的“大前端”。“前端”一词所指代的技术实体正变得越来越模糊，已经不单单是指我们所熟知的 JS、HTML 与 CSS 了，正如大学里生物学专业与化学专业两者融合后所形成的“生物化学专业”一样。技术本身并无好坏之分，只有能否适用于某些业务场景。而技术的融合则正好能够发挥各项技术本身所具备的优势，达到“1+1>2”的效果。WebAssembly 技术便正是如此。

在写作本书的过程中，笔者曾与 WWG 的核心成员 Alon、Ben 和 JF 等专家进行了多次交

流，以力求保证书中各个技术细节的正确性。但 Wasm 技术发展非常之快，比如 Emscripten 工具链每天都会有众多的“commit”被提交到主分支中，新版本的发布也是以“周”甚至“天”为单位进行的。因此，书中所述内容并不保证会在今后的半年甚至一年时间里都具有时效性。而对于相关内容的时效性变化，笔者也会在对应于本书的 Github 仓库（详见“勘误和支持”部分）中及时进行标注。作为国内第一本介绍 WebAssembly 的技术书籍，希望本书的内容能够为国内互联网基础技术的发展做出微小的贡献。虽然现阶段我们还无法完全地自主创新，或者参与到各项国际技术标准的制定过程中，但唯有紧跟其脚步，才能够伺机超越。

本书特色

作为市面上第一本深入介绍 WebAssembly 技术的相关书籍，笔者尝试由浅入深地来介绍与 Wasm 技术相关的各种底层理论知识，以及相关编译器工具链的内部实现结构与使用方法。WebAssembly 技术从其第一版 MVP 标准诞生至今，时间过去并不久，但抽象的英文官方文档并不适合各类 Web 前端开发工程师直接进行阅读。从另一个方面来看，虽然我们可以在国内如“百度”等中文搜索引擎上找到部分与 Wasm 实践相关的介绍文章，但它们大都不会深入该技术标准的背后，探寻该技术的底层设计本质。因此，本书力求从一些简单的实践入手，深入理论，再到复杂的具有实际业务价值的综合实践，深入浅出地介绍 Wasm 技术发展至今，其背后所涉及的各种底层设计原理与实现、相关工具链以及未来发展方向等多方面内容。

由于 WebAssembly 并不是一种单方面的前端或后端技术，因此在本书中，我们将会随着内容的深入逐渐接触到除 Web 前端技术之外的诸如编译原理、V8 引擎、LLVM 以及 Linux 动态链接等多方面内容。笔者将会用最简单和直观的方式，由浅入深地介绍这些平日里可能很少接触到的技术与特性。

另外，作为市面上首本与 WebAssembly 相关的纯技术类书籍，笔者只能从自己所接触到的 Wasm 相关技术中，按照各个知识点的相关性与重要性来编排内容。相信读者在读完整本书后，一定会对 Wasm 技术背后的实现原理以及相关技术有进一步的理解。

读者对象

- Web 前端开发人员。
- C/C++开发人员。
- 对 WebAssembly 技术感兴趣的人员。

本书内容

本书分为 8 章。

第 1 章：本书的开篇，主要介绍 WebAssembly 技术的发展历程，从 PNaCl 到 ASM.js 再到 WebAssembly，以及这些技术的基本应用方法与性能对比。

第 2 章：介绍 WebAssembly 的标准上层 API、底层堆栈机的设计原理，以及对 MVP 标准理论的深入解读。

第 3 章：介绍与 WebAssembly 标准相关的进阶内容，如单指令多数据流（SIMD）、动态链接技术等。

第 4 章：由浅入深地介绍 LLVM 工具链与 WAT 可读文本格式的相关内容。

第 5 章：从理论走向实践，从本章开始介绍基于 Emscripten 工具链开发 WebAssembly 应用的基本流程，以及工具链的一些基本常用功能和特性。

第 6 章：介绍基于 Emscripten 工具链实现的 C/C++语言动态关系绑定技术。

第 7 章：从基础走向深入，继续介绍 Emscripten 工具链所提供的一些如 WebGL 支持、虚拟文件系统、应用优化以及事件系统等高级应用特性。

第 8 章：构建一个具有实际业务价值的 WebAssembly 应用，并介绍现阶段 Wasm 生态的发展情况，以及在 Post-MVP 标准中制订的一些 WebAssembly 未来发展规划。

勘误和支持

作为市面上首本介绍 WebAssembly 相关技术的书籍，本书在内容组织与编排上全部由笔者一人完成。由于笔者的知识水平有限，以及编写时间仓促，书中难免会出现一些错误或不准确的地方，恳请读者批评指正。如果你有更多的宝贵意见，欢迎在笔者的 Github 仓库“Book-DISO-WebAssembly”下创建“issue”，留下你的问题、意见或建议，笔者会在第一时间给予答复。

致谢

首先要感谢的是 Emscripten 工具链的作者 Alon，在编写 Emscripten 实践部分时，通过邮件

与他沟通了很多技术细节上的内容。Alon 是 Mozilla 的技术研究员，他主要负责维护和开发 Emscripten 与 Binaryen 两个重要的 WebAssembly 工具链。Alon 平日工作繁忙，非常感谢他能够抽出时间为我指导技术，以及本书内容组织方面的事情，并为本书写了序言。

其次要感谢的是我的女朋友，工作本已十分繁忙，而写书路漫漫，这期间曾遇到过无数次难以下笔、不知所措的时刻，而她却总是能在关键时刻鼓励我陪我共渡难关。在写书的半年多时间里，基本上所有的双休日都是在家中或咖啡馆度过的，没有时间陪伴，希望能够和你一同见证本书出版的时刻。

还要感谢的是我的好友以及家人。早早地立下写书的誓言，正是因为有了你们每天的监督，我才能督促自己完成每天规定的任务，最终完成本书所有章节的编写。

最后要感谢的是本书的策划编辑张春雨，2017 年上海 QCon 全球软件开发大会之后，他通过微信找到了作为讲师的我，并给予我这次写书的机会。但十分抱歉的是，由于我的日程和时间安排不当，导致本书的出版比预定时间晚了近三个月。但张老师并没有放弃我，这里真的要说一声：“感谢您对我的信任”。

关于源码

关于本书第 3 章、第 4 章和第 5 章中所涉及的相关源代码示例，读者可以在笔者 Github 账号（https://github.com/Becavalier）下的“Cinderella”和“Book-DISO-WebAssembly”仓库中找到完整的代码文件。其他章节中所涉及的部分源代码文件，也会被同时整理并放置到上面的“Book-DISO-WebAssembly”仓库中。

目　录