

# СОДЕРЖАНИЕ

<i>От издательства</i> .....	10
<i>Об авторе</i> .....	11
<i>О техническом рецензенте</i> .....	11
<i>Предисловие</i> .....	12
<i>Благодарности</i> .....	13
<i>Введение</i> .....	14
<b>Глава 1. Введение в WebAssembly</b> .....	19
Что такое WebAssembly? .....	20
Причины использовать WebAssembly .....	21
Повышение производительности.....	22
Интеграция существующих библиотек.....	22
Портируемость на другие платформы и безопасность .....	23
Противники JavaScript .....	23
Связь WebAssembly с JavaScript.....	24
Зачем учить WAT?.....	25
Стили кодирования WAT.....	26
Среда встраивания.....	30
Браузер .....	31
WASI .....	31
Visual Studio Code .....	32
Node.js.....	33
Наше первое приложение WebAssembly с помощью Node.js.....	35
Вызов модуля WebAssembly из Node.js.....	36
Синтаксис <code>.then</code> .....	37
Удачное время.....	38
<b>Глава 2. Основы работы с WebAssembly Text</b> .....	39
Написание простейшего модуля .....	40
Hello World в WebAssembly.....	40
Создание WAT-модуля .....	41
Создание файла JavaScript.....	43
Переменные WAT .....	46
Глобальные переменные и преобразование типов.....	46
Локальные переменные .....	50
Распаковка S-выражений .....	52
Переменные с индексами.....	54
Преобразование между типами .....	54
Условные операторы <code>if/else</code> .....	56
Операторы цикла и блока .....	58

Оператор блока (block) .....	59
Оператор цикла (loop).....	61
Совместное использование операторов блока и цикла .....	62
Переход с помощью br_table .....	64
Заключение .....	66
<b>Глава 3. Функции и таблицы</b> .....	<b>67</b>
Когда следует вызывать функции из WAT .....	68
Разработка функции is_prime .....	68
Передача параметров .....	68
Создание внутренних функций .....	69
Функция is_prime .....	71
Код на стороне JavaScript .....	75
Объявление импортированной функции .....	77
Числа JavaScript .....	78
Передача типов данных .....	78
Объекты в WAT .....	78
Влияние вызовов внешних функций на производительность .....	79
Таблицы функций .....	83
Создание таблицы функций в WAT .....	83
Заключение .....	92
<b>Глава 4. Низкоуровневые битовые операции</b> .....	<b>93</b>
Системы счисления: двоичная, десятичная и шестнадцатеричная .....	94
Арифметические операции над целыми числами и числами с плавающей запятой .....	95
Целые числа .....	96
Числа с плавающей запятой.....	98
Биты старшего и младшего разрядов.....	101
Битовые операции .....	103
Сдвиг и вращение битов .....	103
Маскирование битов с помощью AND и OR .....	105
Инверсия битов с помощью XOR.....	108
Обратный vs. прямой порядок байтов .....	109
Заключение .....	110
<b>Глава 5. Строки в WebAssembly</b> .....	<b>111</b>
ASCII и Unicode.....	112
Строки в линейной памяти.....	112
Передача длины строки в JavaScript.....	113
Строки с завершающим нулем.....	114
Строки с префиксом длины .....	117
Копирование строк.....	120
Создание числовых строк.....	126
Создание шестнадцатеричной строки .....	131
Создание двоичной строки .....	136
Заключение .....	139

<b>Глава 6. Линейная память</b> .....	140
Линейная память в WebAssembly .....	141
Страницы .....	142
Указатели .....	144
Объект памяти JavaScript.....	146
Создание объекта памяти WebAssembly .....	146
Запись в консоль в цвете.....	148
Создание JavaScript в store_data.js .....	149
Обнаружение столкновений.....	151
Начальный адрес, шаг и сдвиг.....	152
Загрузка структур данных из JavaScript .....	153
Отображение результатов .....	155
Функция обнаружения столкновений .....	156
Заключение.....	164
<b>Глава 7. Веб-приложения</b> .....	166
DOM .....	167
Создание и настройка простого сервера Node.....	167
Первое веб-приложение WebAssembly .....	169
Определение HTML-заголовка .....	170
JavaScript.....	170
HTML-тег <body>.....	173
Готовое веб-приложение .....	173
Шестнадцатеричные и двоичные строки.....	175
HTML.....	175
WAT.....	178
Компиляция и запуск.....	183
Заключение.....	184
<b>Глава 8. Работа с Canvas</b> .....	186
Рендеринг HTML-страницы на холсте .....	187
Определение холста в HTML.....	187
Определение констант JavaScript в HTML .....	188
Создание случайных объектов .....	190
Данные растрового изображения.....	191
Функция requestAnimationFrame.....	192
Модуль WAT .....	194
Импортируемые значения .....	194
Очистка холста .....	195
Функция вычисления абсолютного значения .....	196
Установка цвета пикселя.....	197
Рисуем объект.....	200
Установка и получение атрибутов объекта .....	202
Функция \$main.....	204
Компиляция и запуск приложения .....	213
Заключение.....	214

<b>Глава 9. Оптимизация производительности</b> .....	216
Использование профилировщика.....	217
Профилировщик Chrome .....	217
Профилировщик Firefox.....	224
wasm-opt.....	228
Установка Binaryen .....	228
Запуск wasm-opt.....	228
Взглянем на оптимизированный код WAT.....	230
Приемы повышения производительности .....	231
Встраивание функций .....	231
Умножение и деление vs. сдвиг.....	235
DCE .....	237
Сравнение приложения обнаружения столкновений с JavaScript .....	238
Оптимизация WAT вручную .....	241
Запись производительности в лог.....	242
Более сложное тестирование с помощью benchmark.js .....	247
Сравнение WebAssembly и JavaScript с флагом --print-bytecode.....	253
Заключение.....	256
<b>Глава 10. Отладка WebAssembly</b> .....	258
Отладка из консоли.....	259
Запись сообщений в консоль .....	264
Предупреждения об ошибках.....	268
Трассировка стека .....	269
Отладчик Firefox .....	275
Отладчик Chrome .....	279
Заключение.....	282
<b>Глава 11. AssemblyScript</b> .....	283
Интерфейс командной строки в AssemblyScript.....	284
Приложение Hello World на AssemblyScript .....	286
Код JavaScript для приложения Hello World.....	288
Приложение Hello World в загрузчике AssemblyScript .....	290
Объединение строк AssemblyScript.....	291
Объектно-ориентированное программирование на AssemblyScript .....	293
Приватные атрибуты .....	295
Среда встраивания JavaScript .....	297
Загрузчик AssemblyScript.....	298
Расширение классов в AssemblyScript .....	301
Сравнение производительности загрузчика и прямых вызовов WebAssembly .....	303
Заключение.....	306
<i>Послесловие</i> .....	307
<i>Предметный указатель</i> .....	308

## Об авторе

**Рик Батталлини** (Rick Battagline) – разработчик игр и автор книги «Hands-On Game Development with WebAssembly» (Packt Publishing, 2019). Занимается браузерными технологиями с 1994 г. В 2006 г. он основал независимую студию разработки веб-игр BattleLine Games LLC. Разработанная студией игра Epoch Star была номинирована на награду конкурса Slamdance Guerilla Games Competition. На сегодняшний день под его авторством созданы сотни игр, созданных с использованием различных веб-технологий, таких как WebAssembly, HTML5, WebGL, JavaScript, TypeScript, ActionScript и PHP.

## О техническом рецензенте

**Конрад Уатт** (Conrad Watt) – научный сотрудник Питерхауса, Кембриджский университет (Peterhouse, University of Cambridge). Прежде чем получить степень доктора компьютерных наук в Кембриджском университете, он успешно прошел обучение по программе бакалавриата в Имперском колледже Лондона (Imperial College London). Исследования Конрада Уатта сосредоточены преимущественно на формальной семантике и характеристиках безопасности WebAssembly. Он является разработчиком WasmCert-Isabelle – первой в мире механизации семантики языка WebAssembly с помощью инструмента Isabelle/HOL.

Конрад является активным членом WebAssembly Community Group и продолжает участвовать в создании спецификаций новых языковых возможностей, делая упор на многопоточность и параллелизм. Его исследование характеристик слабых моделей памяти JavaScript и WebAssembly является ключевым компонентом спецификации потоков WebAssembly. Вне профессиональной жизни он увлекается хоровым пением, а также много путешествует (напоминаем, что ввиду ситуации, сложившейся в мире на момент написания этой книги, заниматься тем и другим не рекомендуется).

# ПРЕДИСЛОВИЕ

Сегодня, когда большинство языков успешно компилируются в JavaScript, WebAssembly представляет собой новый виток развития технологий, который позволит выйти за пределы устоявшихся рамок. WebAssembly является универсальным эффективным инструментом для выполнения кода на вашем любимом языке в браузере, который к тому же позволяет переосмыслить способы взаимодействия многократно используемых программных компонентов не только в сети, но и на других платформах, начиная от блокчейна и заканчивая граничными вычислениями интернета вещей (IoT – Internet of things).

Безусловно, WebAssembly – это молодая технология, на развитие которой уйдет немало времени. Однако уже сегодня ее явный потенциал вдохновил огромное количество самых разных людей. В качестве примера хотелось бы привести разрабатываемый совместно с Риком (Rick) проект AssemblyScript. Здесь мы рассматриваем WebAssembly не как язык системного программирования, а как инструмент для объединения лучших концепций данной технологии с JavaScript. AssemblyScript позволяет компилировать варианты кода JavaScript, схожие с TypeScript, в WebAssembly, формируя при этом сверхмалые и эффективные модули. Все это дает возможность ощутить преимущества WebAssembly тем, кто работает с JavaScript.

WebAssembly обладает множеством интересных функций и аспектов, которые будут интересны тем, кто стремится исследовать новые технологии и внести свою лепту в их развитие. Книга «Искусство WebAssembly» поможет овладеть базовыми знаниями для дальнейшего более глубокого погружения в тонкости работы технологий, способных совершить революцию в области вычислений и во всей сети.

– ДЭНИЕЛ ВИРТЦ (DANIEL WIRTZ),  
создатель AssemblyScript

# БЛАГОДАРНОСТИ

**В** первую очередь хотелось бы выразить благодарность Лиз Чедвик (Liz Chadwick), работавшей над первыми набросками этой книги. Благодаря неутомимой работе и приложенным усилиям она воплотила мой смутный поток мыслей в связную письменную форму и превратила его в полноценный проект. Если вам понравится читать эту книгу, будьте уверены, что это заслуга Лиз (Liz) и результат ее усилий, приложенных на каждом этапе создания.

Я хочу поблагодарить Конрада Уатта (Conrad Watt), который является приглашенным экспертом рабочей группы W3C WebAssembly, за предоставление экспертных технических рецензий. Я искренне восхищаюсь его талантом. Его технический опыт в данной области невозможно переоценить. Конрад Уатт (Conrad Watt) провел глубокий и доскональный технический анализ данной книги.

Также я хочу сказать спасибо Катрине Тейлор (Katrina Taylor) и Энн Мари Уокер (Anne Marie Walker). Я искренне признателен за вашу работу по подготовке книги к печати. И конечно, я благодарю своих друзей Винита Капура (Vineet Kapur), Стива Тэка (Steve Tack) и Терри Коэна (Terri Cohen), которые нашли время, чтобы ознакомиться с моим первым черновиком и дать по этому поводу свои комментарии. Все вы помогли мне сделать эту книгу лучше.

Особую благодарность хотелось бы выразить Биллу Поллоку (Bill Pollock), чья помощь в некоторых критических моментах помогла мне продвинуться дальше и завершить эту книгу.

# ВВЕДЕНИЕ



Добро пожаловать в «Искусство WebAssembly». Цель этой книги – показать, как читать, понимать и писать код WebAssembly на уровне виртуальной машины. Здесь вы узнаете, как WebAssembly взаимодействует с JavaScript, веб-браузером и средой встраивания. После прочтения книги вы поймете, что такое WebAssembly, узнаете об оптимальных вариантах его применения, а также научитесь писать коды WebAssembly с высокой скоростью их исполнения в браузере.

## Для кого предназначена эта книга

Данная книга предназначена для веб-разработчиков, желающих понять, когда и зачем стоит использовать WebAssembly. Если вы действительно хотите понять WebAssembly, вам необходимо изучить его подробно. На сегодняшний день существует множество книг о различных наборах инструментальных средств WebAssembly. Однако данная книга создана не для того, чтобы научить вас писать коды на C/C++, Rust или другом языке. Вместо этого она расскажет вам о механизмах и возможностях WebAssembly.

Также данная книга предназначена для пользователей, которые хотят понять, что такое WebAssembly, на что он способен и как применять его наиболее оптимальным образом. WebAssembly превосходит JavaScript по части производительности, объема загружаемых данных и потребления памяти. Однако разработка высокопроизводительных приложений WebAssembly представляет собой более сложный процесс, нежели просто написание приложения на C++/Rust или AssemblyScript и его дальнейшая компиляция в WebAssembly. Для создания приложения, в три раза превосходящего по скорости свой эквивалент в JavaScript, необходимо погрузиться в работу WebAssembly чуть глубже.



Читателю необходимы базовые знания о веб-технологиях, таких как JavaScript, HTML и CSS. При этом быть экспертом в какой-либо из них вовсе необязательно. Изучать WebAssembly в нынешнем воплощении, не имея фундаментальных знаний о работе сетей, будет крайне непросто. Безусловно, в рамках этой книги у меня не получится раскрыть все аспекты функционирования веб-страниц. Однако я предполагаю, что большинство читателей уже знакомы с данной темой.

## Чем интересен WebAssembly для пользователей

На первом саммите WebAssembly Эшли Уильямс (Ashley Williams) (@ag\_dubs) представила результаты проведенного ею опроса в Твиттере (Twitter). Первый вопрос для пользователей WebAssembly звучал как «Чем вас привлекает WebAssembly?». Результаты были следующие:

- возможность поддержки нескольких языков, 40,1 %;
- сокращение объема кода и более быстрое его исполнение, 36,8 %;
- изолированность (безопасность), 17,3 %.

Затем она попросила пользователей, выбравших первый вариант (возможность поддержки нескольких языков), рассказать, почему для них важен этот аспект. В результате ответы были следующими:

- недостаточность JavaScript для удовлетворения всех потребностей, 43,5 %;
- возможность повторного использования существующих библиотек, 40,8 %;
- наличие дистрибутива, 8,1 %.

Далее пользователей, выбравших первый вариант ответа (недостаточность JavaScript для удовлетворения всех потребностей), спросили, почему они так считают. Среди причин были:

- низкая или нестабильная производительность, 42 %;
- неудовлетворенность экосистемой, 17,4 %;
- нежелание или невозможность разобраться, 31,3 %.

Вы можете ознакомиться с докладом Эшли (Ashley) «Why the #wasmsummit Website Isn't Written in Wasm» на YouTube по адресу <https://www.youtube.com/watch?v=J5Rs9oG3FdI>.

Несмотря на то что описанный выше опрос не был частью научно-го исследования, он все же представляется достаточно информативным. Как минимум, по его результатам выяснилось, что более 55 % пользователей заинтересованы в повышении производительности своих приложений посредством WebAssembly. И несомненно, повышение производительности кода с помощью WebAssembly более чем возможно. В свою очередь, реализация WebAssembly – это не волшебство; для этого просто необходимо знать, что и зачем вы дела-

ете. После прочтения данной книги вы будете знать о WebAssembly достаточно, чтобы значительно повысить производительность ваших веб-приложений.

## Почему миру необходим WebAssembly

Я занимаюсь разработкой веб-приложений с середины 1990-х гг. Изначально веб-страницы были не более чем документами с изображениями. Ситуация изменилась с появлением Java и JavaScript. В то время JavaScript был «языком-игрушкой», который позволял добавлять эффекты при наведении курсора на кнопки на веб-страницах. Большинство считало Java весьма стоящим продуктом, а виртуальную машину Java (JVM – Java virtual machine) новой захватывающей технологией. Однако Java так и не смогла реализовать весь свой потенциал на веб-платформе. Дело в том, что Java требовала использования встраиваемых расширений, которые в конечном итоге вышли из моды, поскольку очень часто они были источником вредоносных ПО и не соответствовали требованиям безопасности.

К сожалению, Java является проприетарной технологией, что не позволяет интегрировать поддержку этого языка напрямую в веб-браузер. WebAssembly же отличается своим открытым форматом. Данная технология является продуктом сотрудничества между многими поставщиками аппаратного и программного обеспечения, такими как Google, Mozilla, Microsoft и Apple. Она не требует встраиваемых расширений и может быть успешно интегрирована во все современные веб-браузеры. Объединив использование данной технологии с Node.js, вы можете создавать аппаратно-независимое ПО. Ввиду открытого формата технология может быть использована без каких-либо лицензионных отчислений или разрешения на любой аппаратной либо программной платформе. Так что WebAssembly является воплощением мечты 1990-х – один двоичный код на все случаи жизни (one binary to rule them all).

## Структура книги

В этой книге мы расскажем о работе WebAssembly на низком уровне, познакомив вас с текстовым форматом WebAssembly Text. Мы рассмотрим множество вопросов низкоуровневой реализации, а также покажем, как WebAssembly взаимодействует с JavaScript в Node.js и веб-приложениях. Данную книгу следует читать последовательно, поскольку описываемые концепции опираются друг на друга. Также здесь содержатся ссылки на примеры кода, которые можно найти на <https://wasmbok.com>.

### Глава 1 «Введение в WebAssembly»

В этой главе мы подробно рассмотрим, что представляет собой технология WebAssembly, а также поговорим о наиболее оптимальных

вариантах ее применения. Вы познакомитесь с WebAssembly Text (WAT), который позволяет понять, как работает WebAssembly на самом низком уровне. Также стоит отметить, что мы создали специальную среду, включающую все примеры из данной книги.

## **Глава 2 «Основы работы с WebAssembly Text»**

Здесь мы рассмотрим основы WAT и его взаимосвязь с высокоуровневыми языками, которые могут быть развернуты в WebAssembly. Также вы напишете свою первую программу на WAT, а затем мы обсудим такую фундаментальную концепцию, как контроль потока с помощью переменных.

## **Глава 3 «Функции и таблицы»**

В этой главе мы рассмотрим создание функций в модулях WebAssembly и их вызов в JavaScript. Затем вы напишете программу для проверки простых чисел, что послужит отличной иллюстрацией для описанных концепций. После мы исследуем вызов функций из таблиц и влияние данного процесса на производительность.

## **Глава 4 «Низкоуровневые битовые операции»**

Здесь вы узнаете о низкоуровневых концепциях, которые можно использовать для повышения производительности модулей WebAssembly. Среди них: системы счисления, побитовое маскирование и дополнительный код.

## **Глава 5 «Строки в WebAssembly»**

В WebAssembly нет встроенной поддержки строкового типа данных. В этой главе вы узнаете, как представлены строки в WebAssembly и как ими управлять.

## **Глава 6 «Линейная память»**

В этой главе вы познакомитесь с линейной памятью и тем, как модули WebAssembly используют ее для обмена большими наборами данных в JavaScript или альтернативной среде. Здесь вы начнете создание программы обнаружения столкновений, которая заставляет объекты перемещаться случайным образом и проверяет наличие столкновений между ними. Мы будем обращаться к данной программе на протяжении всей книги.

## **Глава 7 «Веб-приложения»**

Здесь вы узнаете, как создаются простые веб-приложения с помощью HTML, CSS, JavaScript и WebAssembly.

## **Глава 8 «Работа с canvas»**

Здесь мы обсудим, как использовать элемент HTML canvas в WebAssembly для создания быстрой веб-анимации. Также мы научимся применять canvas для улучшения созданной ранее программы обнаружения столкновений.

## **Глава 9 «Оптимизация производительности»**

В этой главе вы узнаете, почему WebAssembly хорошо подходит для выполнения ресурсоемких задач, таких как обнаружение столкновений. Также вы научитесь использовать профилировщики Chrome и Firefox и другие инструменты оптимизации для повышения производительности приложений.

## **Глава 10 «Отладка WebAssembly»**

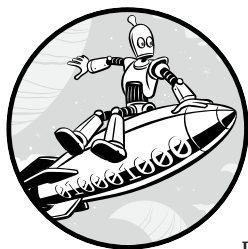
Здесь мы рассмотрим основы отладки, такие как вывод данных в веб-консоль посредством уведомлений и трассировки стека. Также вы научитесь использовать отладчики в Chrome и Firefox для пошагового исполнения кода WebAssembly.

## **Глава 11 «AssemblyScript»**

В этой главе мы обсудим использование WAT с точки зрения понимания высокоуровневых языков и оценки AssemblyScript – высокоуровневого языка, разработанного для эффективного развертывания в WebAssembly.

# 1

## ВВЕДЕНИЕ В WEBASSEMBLY



В этой главе вы получите базовые знания о технологии WebAssembly и изучите инструменты, которые понадобятся вам для начала работы с ней и ее текстовым воплощением – WebAssembly Text (WAT). Мы обсудим преимущества WebAssembly, такие как повышение производительности, возможность интеграции уже существующих библиотек, портируемость на другие платформы, безопасность и использование WebAssembly в качестве альтернативы JavaScript. Мы рассмотрим взаимосвязь JavaScript и WebAssembly, а также то, чем же на самом деле является WebAssembly. Вы научитесь синтаксису встроенных WAT- и S-выражений (символьные выражения). Мы познакомимся с концепциями среды встраивания и обсудим встраивание WebAssembly в веб-браузеры, Node.js и системный интерфейс WebAssembly (WASI – WebAssembly System Interface).

Затем мы рассмотрим преимущества использования Visual Studio Code в качестве среды разработки для WAT. Вы познакомитесь с основами Node.js и научитесь встраивать в данную платформу Web-

Assembly. Мы покажем, как использовать пакетный менеджер npm (Node Package Manager) для установки инструмента wat-wasm, с помощью которого вы сможете создавать приложения WebAssembly на основе WAT. Кроме того, вы создадите свое первое приложение WebAssembly, встроенное в среду Node.js.

## Что такое WebAssembly?

WebAssembly – это технология, которая в перспективе следующих нескольких лет позволит значительно повысить производительность веб-приложений. Поскольку WebAssembly является молодой технологией и требует некоторых пояснений, многие люди используют ее не совсем правильно. Данная книга расскажет вам, что представляет собой технология WebAssembly и как использовать ее для создания высокопроизводительных веб-приложений.

WebAssembly – это *виртуальная архитектура набора команд (ISA – Instruction Set Architecture)* для стековой машины. Как правило, ISA представляется в двоичном формате и предназначается для конкретной машины. В свою очередь, WebAssembly разработан для работы на *виртуальной* машине, то есть не предназначен для физического аппаратного обеспечения. Виртуальная машина позволяет WebAssembly работать на разнообразном аппаратном обеспечении компьютеров и цифровых устройствах. ISA для WebAssembly предполагает компактность, межплатформенность и безопасность, а также наличие небольших двоичных файлов, что позволяет сократить время загрузки при развертывании в качестве части веб-приложения. Байт-код легко перенести на самые разные аппаратные платформы, а также он может быть безопасно развернут в интернете.

Большинство современных браузеров уже поддерживают WebAssembly. По данным Mozilla Foundation, код WebAssembly работает на 10–800 % быстрее, чем эквивалентный код JavaScript. К примеру, исполнение одного проекта eBay с помощью WebAssembly было в 50 раз быстрее, чем с JavaScript. В последующих главах мы расскажем, как создать программу обнаружения столкновений, которая может использоваться для измерения производительности. При запуске проведенные тесты производительности показали, что код обнаружения столкновений WebAssembly работает более чем в четыре раза быстрее того же JavaScript в Chrome и более чем в два раза быстрее, чем JavaScript в Firefox.

WebAssembly предлагает наиболее значительное повышение производительности в сети с момента появления динамических JIT-компиляторов (JIT – just-in-time) JavaScript. Современные браузерные движки JavaScript могут загружать и анализировать двоичный формат WebAssembly на порядок быстрее, чем JavaScript. Дело в том, что WebAssembly является генератором двоичного целевого кода для среды выполнения, а не языком программирования, как JavaScript, что

позволяет разработчикам выбирать языки программирования, которые лучше соответствуют потребностям его приложения. Сегодня многие говорят, что «JavaScript – это ассемблерный язык для сети», но на самом деле формат JavaScript является не лучшей целевой платформой для компиляции. JavaScript значительно менее эффективен, чем двоичный формат WebAssembly. Также любой код целевой платформы JavaScript должен уметь справляться со всеми особенностями языка JavaScript.

WebAssembly позволяет значительно повысить производительность веб-приложений в двух аспектах. Один из них – скорость запуска. В настоящее время наиболее компактным форматом JavaScript является минифицированный JavaScript, который позволяет уменьшить размеры загружаемых приложений, но при этом должен анализировать, интерпретировать, осуществлять JIT-компиляцию и оптимизировать код JavaScript. Для более компактного двоичного файла WebAssembly перечисленные выше действия не требуются. WebAssembly по-прежнему нуждается в парсинге, однако этот процесс происходит гораздо быстрее, ввиду того что байт-код анализировать гораздо проще, чем текстовый формат. Также WebAssembly требуется оптимизация веб-движками, однако и этот процесс происходит намного быстрее ввиду большей четкости языка.

Более того, WebAssembly предлагает значительное улучшение пропускной способности. WebAssembly упрощает оптимизацию движка браузера. JavaScript – это очень динамичный и гибкий язык программирования, который полезен для разработчика JavaScript, но абсолютно не подходит для оптимизации кода. WebAssembly не имеет никаких веб-предпочтений (несмотря на свое название) и может использоваться вне браузера.

В скором времени WebAssembly сможет делать все, что умеет JavaScript. К сожалению, текущая версия – приложение с минимальным функционалом (MVP – Minimum Viable Product) версии 1.0 – на это неспособна. MVP-версия WebAssembly подходит для выполнения конкретных задач, но она не предназначена для замены JavaScript или фреймворков, таких как Angular, React или Vue. Если вы хотите работать с WebAssembly прямо сейчас, у вас должен быть конкретный проект с интенсивными вычислениями, который требует очень высокой производительности. Онлайн-игры, WebVR, 3D-математика и криптовалюта являются теми областями, где может эффективно применяться WebAssembly.

## Причины использовать WebAssembly

Прежде чем перейти к детальному изучению WebAssembly, давайте рассмотрим несколько причин, по которым вам может быть интересно его использовать. Вы поймете, что такое WebAssembly, а также для чего и как его использовать.



## ***Повышение производительности***

JavaScript требует от разработчиков программного обеспечения делать выбор, который повлияет на то, как они проектируют движок JavaScript. Например, вы можете оптимизировать движок JavaScript для достижения максимальной производительности с помощью оптимизирующего компилятора JIT, который может выполнять код быстрее, но требует больше времени на запуск. В качестве альтернативы вы можете использовать интерпретатор, который сразу же запускает выполнение кода, но не достигает максимальной производительности оптимизирующего компилятора JIT. Решение, которое большинство разработчиков движков JavaScript используют в своих веб-браузерах, заключается в реализации обоих подходов, но для этого требуется гораздо больший объем памяти. Каждое ваше решение – это компромисс.

WebAssembly позволяет ускорить запуск и повысить пиковую производительность без чрезмерного увеличения объема памяти. Но, к сожалению, вы все же не сможете просто переписать свой JavaScript на AssemblyScript, Rust или C++ и ожидать, что это произойдет без каких-либо дополнительных действий. WebAssembly – это не волшебство, поэтому простой перенос программы JavaScript на другой язык и его компиляция без понимания того, что делает WebAssembly на более низком уровне, могут привести к разочаровывающим результатам. Написание кода C++ и его компиляция в WebAssembly с помощью флагов оптимизации обычно занимают меньше времени, чем JavaScript. Иногда программисты жалуются на то, что они потратили весь день на переписывание своего приложения на C++, а оно работает лишь на 10 % быстрее. Если это так, то, вероятно, этим приложениям не очень поможет перенос в WebAssembly, и их код на C++ пусть и дальше компилируется в JavaScript. Выделите некоторое количество времени, чтобы изучить WebAssembly, а не C++, и заставьте свои веб-приложения работать с молниеносной скоростью.

## ***Интеграция существующих библиотек***

Две популярные библиотеки для переноса существующих библиотек на WebAssembly – это `wasm-pack` для Rust и `Emscripten` для C/C++. Использование WebAssembly идеально подходит тогда, когда у вас есть существующий код, написанный на C/C++ или Rust, который вы хотите сделать доступным для веб-приложений, или же хотите перенести любые существующие настольные приложения, чтобы сделать их доступными в сети. Набор инструментальных средств `Emscripten` особенно эффективен при переносе существующих настольных приложений C++ в сеть с помощью WebAssembly. Если вы следуете этому пути, то, вероятно, захотите, чтобы ваше приложение после переноса работало как можно ближе к исходной скорости вашего существующего приложения, что может быть осуществимо, только если прило-



жение не потребляет много ресурсов. Однако у вас также может быть приложение, для которого придется повозиться с оптимизацией быстродействия, чтобы оно работало так, как на компьютере. К концу этой книги вы сможете оценить модуль WebAssembly, созданный вашим набором инструментальных средств на основе существующего кода.

## ***Портируемость на другие платформы и безопасность***

Мы объединили их в один раздел, потому что они часто зависят друг от друга. WebAssembly изначально был лишь технологией для запуска в браузере, а затем стал быстро расширяться, превращаясь в изолированную среду, которую можно запускать где угодно. Рабочая группа WebAssembly создает высокозащищенную среду выполнения, которая не позволяет злоумышленникам скомпрометировать ваш код – от серверного кода WASI до WebAssembly для встраиваемых систем и интернета вещей (IoT). Я рекомендую послушать превосходный доклад Лин Кларк (Lin Clark) о безопасности WebAssembly и повторном использовании пакетов на первом саммите WebAssembly (<https://www.youtube.com/watch?v=IBZFJzGnBoU/>).

Несмотря на то что рабочая группа WebAssembly внимательно относится к безопасности, ни одна система не является полностью безопасной. Изучение WebAssembly на низком уровне подготовит вас к любым будущим угрозам безопасности.

## ***Противники JavaScript***

Некоторым людям просто не нравится JavaScript, и они не хотят, чтобы он был доминирующим языком веб-программирования. К сожалению, WebAssembly все же не в состоянии свергнуть JavaScript. Сегодня JavaScript и WebAssembly должны мирно сосуществовать и хорошо работать вместе, как показано на рис. 1.1.

Но в мире есть и хорошие новости для противников JavaScript: наборы инструментов WebAssembly предлагают множество вариантов написания веб-приложений без использования JavaScript. Например, Emscripten позволяет писать веб-приложения на C/C++ с очень малым количеством кода JavaScript, если таковой потребуется, конечно. Вы также можете писать целые веб-приложения, используя Rust и wasm-pack. Эти наборы инструментов не только генерируют WebAssembly, но также создают обширный связующий код JavaScript для вашего приложения. Причина в том, что в настоящее время возможности WebAssembly ограничены, а наборы инструментов заполняют эти пробелы с помощью кода JavaScript. Прелесть полноценных наборов инструментов, таких как Emscripten, в том, что они делают все за вас. Если вы разрабатываете с помощью одного из этих наборов инструментов, полезно понимать, когда ваш код превратится в WebAssembly, а когда – в JavaScript. Эта книга поможет вам узнать, когда это произойдет.

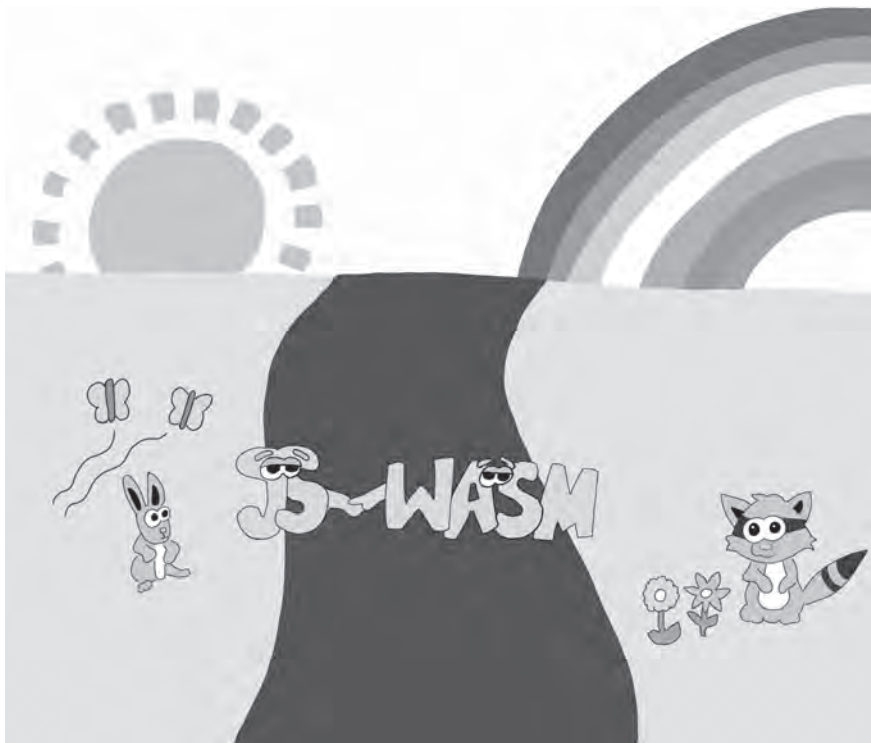


Рис. 1.1. JavaScript и WebAssembly могут сосуществовать в гармонии

## Связь WebAssembly с JavaScript

Важно прояснить, как WebAssembly используется и соотносится с JavaScript. WebAssembly не является прямой заменой JavaScript; скорее WebAssembly:

- быстрее загружается, компилируется и выполняется;
- позволяет писать приложения для сети на языках, отличных от JavaScript;
- при правильном использовании может обеспечивать скорость, близкую к исходной для вашего приложения;
- работает с JavaScript для повышения производительности ваших веб-приложений, когда используется надлежащим образом;
- не является ассемблерным языком, хотя с ним связан псевдоассемблерный язык (WAT);
- предназначен не только для сети и также может выполняться небраузерными движками Javascript, такими как Node.js, или может выполнять байт-код в средах выполнения, реализующих WASI;
- пока что не универсальное решение для создания веб-приложений.

WebAssembly – это результат сотрудничества всех основных производителей браузеров над созданием новой платформы для распространения приложений через интернет. Язык JavaScript эволюционировал от потребностей веб-браузеров в конце 1990-х гг. до полноценного языка сценариев, которым он является сегодня. Хотя JavaScript стал достаточно быстро работающим языком, веб-разработчики заметили, что иногда он показывает плохие результаты. WebAssembly – это решение многих проблем с производительностью JavaScript.

Несмотря на то что WebAssembly не может делать абсолютно все, что умеет JavaScript, он может выполнять определенные операции намного быстрее, чем JavaScript, при этом потребляя меньше памяти. В этой книге мы сравниваем код JavaScript с WebAssembly. Мы будем многократно тестировать и профилировать код для сравнения. К концу книги вы будете уметь определять, когда вам следует использовать WebAssembly, а когда имеет смысл продолжать пользоваться JavaScript.

## Зачем учить WAT?

Многие книги и руководства по WebAssembly сосредоточены на конкретных наборах инструментов, таких как вышеупомянутый `wasm-pack` для Rust или `Emscripten` для C/C++. Наборы инструментов для других языков, таких как `AssemblyScript` (разновидность `TypeScript`) и `Go`, в настоящее время находятся в разработке. Эти наборы инструментов – основная причина, по которой программисты обращаются к WebAssembly, и постоянно становятся доступными все новые языковые наборы инструментов WebAssembly. В будущем веб-разработчики смогут выбирать язык, на котором они будут разрабатывать, исходя из потребностей проекта, а не доступности языков.

Одним из важных факторов для любого из этих языков является понимание того, что делает WebAssembly на самом низком уровне. Глубокое понимание WAT даст вам ответ на вопрос, почему код может работать не так быстро, как бы вам хотелось. Также поможет понять, как WebAssembly взаимодействует со своей средой встраивания. Написание модуля в WAT – лучший способ работать в веб-браузере как можно ближе к «железу» (на низком уровне). Знание WAT поможет вам создавать максимально производительные веб-приложения, разбирать и оценивать любое веб-приложение, написанное для платформы WebAssembly. Вы сможете оценить любые потенциальные риски безопасности в будущем. Кроме того, он позволяет писать код, который максимально приближен к скорости обычного приложения, но без написания специфичного кода для программно-аппаратной платформы.

Так что же такое WAT? WAT похож на ассемблерный язык для виртуальной машины WebAssembly. В практическом смысле это означает, что при написании программ WebAssembly на таком языке, как Rust

или C++, используется набор инструментов, который, как упоминалось ранее, компилирует двоичный файл WebAssembly, а также связующий код JavaScript и HTML, в который встроен модуль WebAssembly. Файл WebAssembly очень похож на машинный код, поскольку он включает разделы, коды операций и данные, которые хранятся в виде последовательности двоичных чисел. Когда у вас есть исполняемый файл в машинном коде, вы можете деассемблировать этот файл в машинный ассемблерный язык, который находится на самом низком уровне языков программирования. Ассемблирование заменяет числовые коды операций в двоичном формате мнемоническими кодами, которые читабельны для человека. WAT действует как ассемблерный язык для WebAssembly.

## Стили кодирования WAT

Есть два основных стиля кодирования WAT. Первый – это *стиль линейного набора команд*. Такой стиль кодирования требует, чтобы разработчик параллельно отслеживал элементы в стеке. Большинство команд WAT проталкивают элементы в стек, выталкивают их из стека или и то, и другое. Если вы решите писать в стиле линейного набора команд, то должны знать, что существует неявный стек, в котором параметры ваших команд должны быть помещены перед вызовом самих команд. Другой стиль кодирования называется *S-выражениями*. S-выражения представляют собой древовидную структуру кодирования, в которой параметры передаются в дерево способом, который похож на вызовы функций в JavaScript. В случае если у вас есть проблемы с визуализацией стека и его элементов, синтаксис S-выражения, скорее всего, подойдет больше. Вы также можете комбинировать два данных стиля в зависимости от неявного стека для менее сложных команд, а когда становится сложно отслеживать количество параметров – использовать S-выражение.

## Применение стиля линейного набора команд

Рассмотрим простую функцию сложения в листинге 1.1, которая написана на языке JavaScript.

*Листинг 1.1. Код JavaScript, выполняющий сложение переменных a\_val и b\_val*

---

```
function main() {  
    let a_val = 1;  
    let b_val = 2;  
    let c_val = a_val + b_val;  
}
```

---

После выполнения этих строк значение переменной c\_val теперь равно 3, что является результатом сложения a\_val и b\_val. Чтобы