

Веб-сборка

WebAssembly — это новый тип кода, который можно запускать в современных веб-браузерах. Это низкоуровневый ассемблерный язык с компактным двоичным форматом, который работает с производительностью, близкой к исходной, и предоставляет такие языки, как C/C++, C# и Rust. с целью компиляции, чтобы их можно было запускать в Интернете. Он также предназначен для работы вместе с JavaScript, что позволяет им работать вместе.

В двух словах

WebAssembly имеет огромное значение для веб-платформы — он предоставляет возможность запускать в Интернете код, написанный на нескольких языках, со скоростью, близкой к естественной, при этом клиентские приложения работают в Интернете, что раньше было невозможно.

WebAssembly предназначен для дополнения и работы вместе с JavaScript — с помощью API-интерфейсов JavaScript WebAssembly вы можете загружать модули WebAssembly в приложение JavaScript и совместно использовать функциональность между ними. Это позволяет вам воспользоваться преимуществами производительности и мощи WebAssembly, а также выразительности и гибкости JavaScript в одних и тех же приложениях, даже если вы не знаете, как писать код WebAssembly.

И что еще лучше, он разрабатывается как веб-стандарт через [рабочую группу W3C WebAssembly](#) и [группу сообщества](#) при активном участии всех основных поставщиков браузеров.

Путеводители



Концепции веб-сборки

Начните с прочтения общих концепций, лежащих в основе WebAssembly — что это такое, почему это так полезно, как оно вписывается в веб-платформу (и за ее пределами) и как ее использовать.



Компиляция нового модуля C/C++ в WebAssembly

Написав код на C/C++, вы можете затем скомпилировать его в Wasm с помощью такого инструмента, как [Emscripten](#). Давайте посмотрим, как это работает.



Компиляция существующего модуля C в WebAssembly

Основной вариант использования WebAssembly — взять существующую экосистему библиотек C и позволить разработчикам использовать их в Интернете.



Компиляция из Rust в WebAssembly

Если вы написали код на Rust, вы можете скомпилировать его в WebAssembly! В этом руководстве вы узнаете все, что вам нужно знать, чтобы скомпилировать проект Rust в Wasm и использовать его в существующем веб-приложении.

[Загрузка и запуск кода WebAssembly](#)

Если у вас есть модуль Wasm, в этой статье рассказывается, как его получить, скомпилировать и создать экземпляр, сочетая API [JavaScript WebAssembly](#) с API-интерфейсами [Fetch](#) или [XHR](#).

[Использование API JavaScript WebAssembly](#)

Загрузив модуль Wasm, вы захотите его использовать. В этой статье мы покажем вам, как использовать WebAssembly через API JavaScript WebAssembly.

[Экспортированные функции WebAssembly](#)

Экспортированные функции WebAssembly — это отражения функций WebAssembly в JavaScript, которые позволяют вызывать код WebAssembly из JavaScript. В этой статье описывается, что они из себя представляют.

[Понимание текстового формата WebAssembly](#)

В этой статье объясняется текстовый формат Wasm. Это низкоуровневое текстовое представление модуля Wasm, отображаемое в инструментах разработчика браузера при отладке.

[Преобразование текстового формата WebAssembly в Wasm](#)

В этой статье представлено руководство по преобразованию модуля WebAssembly, написанного в текстовом формате, в двоичный файл Wasm.

Справочник по API

Справочник инструкций WebAssembly Справочная документация с интерактивными примерами набора операторов WebAssembly.

JavaScript-интерфейс WebAssembly Этот объект действует как пространство имен для всех функций, связанных с WebAssembly.

WebAssembly.Global() Объект WebAssembly.Global представляет собой экземпляр глобальной переменной, доступный как из JavaScript, так и импортируемый/экспортируемый в один или несколько WebAssembly.Module экземпляров. Это позволяет динамически связывать несколько модулей.

WebAssembly.Module() Объект WebAssembly.Module содержит код WebAssembly без сохранения состояния, который уже скомпилирован браузером и может эффективно использоваться совместно с Workers и создаваться несколько раз.

`WebAssembly.Instance()` Объект `WebAssembly.Instance`— это исполняемый экземпляр файла `Module`. `Instance` объекты содержат все экспортированные функции `WebAssembly`, которые позволяют вызывать код `WebAssembly` из `JavaScript`.

`WebAssembly.compile()` Функция `WebAssembly.compile()` компилирует двоичный код `WebAssembly` в `WebAssembly.Module` объект.

`WebAssembly.compileStreaming()` Функция `WebAssembly.compileStreaming()` компилирует `WebAssembly.Module` непосредственно из потокового базового источника.

`WebAssembly.instantiate()` Функция `WebAssembly.instantiate()` позволяет компилировать и создавать экземпляры кода `WebAssembly`.

`WebAssembly.instantiateStreaming()` Функция `WebAssembly.instantiateStreaming()` является основным API для компиляции и создания экземпляра кода `WebAssembly`, возвращая как `Module` так и его первый файл `Instance`.

`WebAssembly.validate()` Функция `WebAssembly.validate()` проверяет заданный типизированный массив двоичного кода `WebAssembly`.

`WebAssembly.Memory()` Объект `WebAssembly.Memory`— это объект изменяемого размера `ArrayBuffer`, который содержит необработанные байты памяти, к которым обращается объект `Instance`.

`WebAssembly.Table()` Объект `WebAssembly.Table` представляет собой типизированный массив изменяемого размера непрозрачных значений, таких как ссылки на функции, к которым обращается объект `Instance`.

`WebAssembly.Tag()` Объект `WebAssembly.Tag` определяет тип исключения `WebAssembly`, которое может быть выброшено в код `WebAssembly` или из него.

`WebAssembly.Exception()` Объект `WebAssembly.Exception` представляет собой исключение во время выполнения, генерируемое из `WebAssembly` в `JavaScript` или генерируемое из `JavaScript` в обработчик исключений `WebAssembly`.

`WebAssembly.CompileError()` Создает новый `CompileError` объект `WebAssembly`.

`WebAssembly.LinkError()` Создает новый `LinkError` объект `WebAssembly`.

`WebAssembly.RuntimeError()` Создает новый `RuntimeError` объект `WebAssembly`.

Примеры `WASM` Собел См. наш репозиторий `webassembly-examples` для получения ряда других примеров.

From:

<https://book51.ru/> - **book51.ru**

Permanent link:

<https://book51.ru/doku.php?id=software:development:web:docs:webassembly&rev=1709036361>

Last update: **2024/02/27 15:19**

