WebAssembly

Or: How I tried to write a web app without using Javascript

Michael Kreuzer

July 3, 2019

Overview

Disclaimers

Disclaimer

I'm not an expert in web development. Things I say in this talk could be wrong.

Disclaimers

Disclaimer

I'm not an expert in web development. Things I say in this talk could be wrong.

Disclaimer

I will focus on the usage of Rust for this talk. Who wants to program C/C++ anyway.

Disclaimers

Disclaimer

I'm not an expert in web development. Things I say in this talk could be wrong.

Disclaimer

I will focus on the usage of Rust for this talk. Who wants to program C/C++ anyway.

Disclaimer

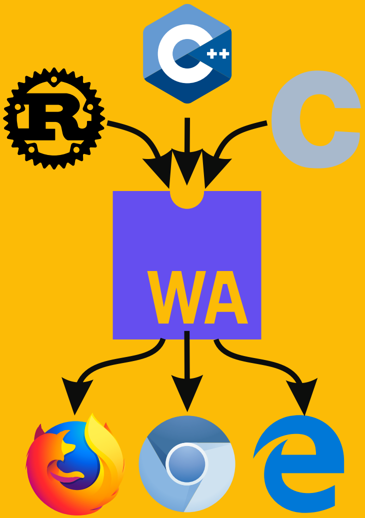
This talk might contain rants about Javascript and other web technologies.

What is WebAssembly?

From the Website

WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications.

What is WebAssembly?



Benefits

According to the Website:

- Efficient and fast
- Safe
- Part of the open web platform
- Open and debuggable

Benefits

According to the Website:

- Efficient and fast
- Safe
- Part of the open web platform
- Open and ~~debuggable~~

```
get_local 0
i64.const 0
i64.eq
if i64
  i64.const 1
else
  get_local 0
  get_local 0
  i64.const 1
  i64.sub
  call 0
  i64.mul
end
```

Benefits

My Opinion:

- Write your web code in a usable language
- Use Code from non web libraries
- Avoid the use of Javascript.

Benefits

My Opinion:

- Write your web code in a usable language
- Use Code from non web libraries
- Avoid the use of Javascript. *Almost.*

History

- 2015 - WebAssembly Community Group started and first public announcement
- 2016 - Definition of core features and Browser Preview with different implementations

History

- 2015 - WebAssembly Community Group started and first public announcement
- 2016 - Definition of core features and Browser Preview with different implementations
- March 2017 - Cross-browser consensus and end of Browser Preview
- February 2018 - Three public working drafts for the Core Specification, JavaScript Interface, and Web API published.

Current Status

Minimum Viable Product (MVP)

WebAssembly is currently in the status of a MVP. Version 1.0 *works* in all major browsers

Current Status

Minimum Viable Product (MVP)

WebAssembly is currently in the status of a MVP. Version 1.0 *works* in all major browsers

Future Work

- Stabilize LLVM backend
- Multi-threading
- Garbage collection
- Everything else...

How it works

Machine Model

WebAssembly uses a stack-based virtual machine with linear memory to evaluate instructions

How it works

Machine Model

WebAssembly uses a stack-based virtual machine with linear memory to evaluate instructions

Data Format

Normally stored in a compact binary format, but can be converted to an equivalent text format that is human readable. Code is compiled to this format via LLVM.

How it works

Machine Model

WebAssembly uses a stack-based virtual machine with linear memory to evaluate instructions

Data Format

Normally stored in a compact binary format, but can be converted to an equivalent text format that is human readable. Code is compiled to this format via LLVM.

Execution

WebAssembly can currently only be executed through a specific Javascript API. It is planned to add the possibility to use Wasm as a script module directly in the future.

A Small Function

```
(module
  (func $f (param f64) (result f64)
    get_local 0
    f64.const 1
    f64.lt
    if (result f64)
      f64.const 1
    else
      get_local 0
      get_local 0
      f64.const 1
      f64.sub
      call $f
      f64.mul
    end)
  (export "f" (func $f)))
```

A Small Example

Javascript

```
fetch('demo.wasm').then(response =>
    response.arrayBuffer()
).then(buffer =>
    WebAssembly.instantiate(buffer)
).then(({module, instance}) =>
    console.log(instance.exports.f(5))
);
```

The Rust Programming Language

- Modern system programming language funded by Mozilla
- Innovative memory management system
- Zero cost abstractions for safer code

The Rust Programming Language

- Modern system programming language funded by Mozilla
- Innovative memory management system
- Zero cost abstractions for safer code

Factorial in Rust

```
fn fac(n : u32) {
    if n == 0 {
        1
    } else {
        n * fac(n-1)
    }
}
```

The Ecosystem for Rust

Compile target

Rust natively supports Wasm with the `wasm32-unknown-unknown` compile target.

The Ecosystem for Rust

Compile target

Rust natively supports Wasm with the `wasm32-unknown-unknown` compile target.
Unfortunately this is not very useful on its own.

The Ecosystem for Rust

Compile target

Rust natively supports Wasm with the `wasm32-unknown-unknown` compile target.
Unfortunately this is not very useful on its own.

What do we need to make it useful?

- Some way of making the file run in our browser
- Accessing common WebAPI functions (DOM manipulation, events, ...)
- Interop with other Javascript code

Two Choices

Wasm-Bindgen

Cargo-Web / Stdweb

Two Choices

Wasm-Bindgen

- Official Wasm toolkit of the Rust foundation

Cargo-Web / Stdweb

- Unofficial Wasm toolkit, but has been around a bit longer

Two Choices

Wasm-Bindgen

- Official Wasm toolkit of the Rust foundation
- Only supports Rust's native Wasm toolchain

Cargo-Web / Stdweb

- Unofficial Wasm toolkit, but has been around a bit longer
- Supports Rust's native but also the emscripten build

Two Choices

Wasm-Bindgen

- Official Wasm toolkit of the Rust foundation
- Only supports Rust's native Wasm toolchain
- Very rapid development

Cargo-Web / Stdweb

- Unofficial Wasm toolkit, but has been around a bit longer
- Supports Rust's native but also the emscripten build
- Still actively developed but slower

Two Choices

Wasm-Bindgen

- Official Wasm toolkit of the Rust foundation
- Only supports Rust's native Wasm toolchain
- Very rapid development
- Aims at providing a tight integration between Javascript and Rust

Cargo-Web / Stdweb

- Unofficial Wasm toolkit, but has been around a bit longer
- Supports Rust's native but also the emscripten build
- Still actively developed but slower
- Aims at providing a tool to write a full-Rust web app.

Wasm-Bindgen

Deployment Options

bundler	Suitable for loading in bundlers like Webpack
web	Directly loadable in a web browser
nodejs	Loadable via require as a Node.js module

Wasm-Bindgen

Deployment Options

bundler	Suitable for loading in bundlers like Webpack
web	Directly loadable in a web browser
nodejs	Loadable via require as a Node.js module

Wasm-Pack

Can magically pack your whole code and publish it to the npm repositories...

Wasm-Bindgen Hello World

src/main.rs

```rust
use wasm_bindgen::prelude::*;

#[wasm_bindgen]
extern "C" {
    fn alert(s: &str);
}

#[wasm_bindgen]
pub fn greet(name: &str) {
    alert(&format!("Hello, {}!", name));
}
```

index.html

```
<html>
  <body>
    <script type="module">
      import init, { greet } from './pkg/hello.js';

      async function run() {
        await init();
        greet("Michael");
      }
      run();
    </script>
  </body>
</html>
```

Accessing the DOM

```
#[wasm_bindgen(start)]
pub fn main() -> Result<(), JsValue> {
    let window = web_sys::window().unwrap();
    let document = window.document().unwrap();
    let body = document.body().unwrap();

    let val = document.create_element("p")?;
    val.set_inner_html("Hello from Rust!");
    body.append_child(&val)?;

    Ok(())
}
```

State of the Art

- Low level access to nearly all browser and Javascript APIs via the web-sys and js-sys crates.
- replacing some functions in your Javascript code with Rust is easy.
- writing a complete web app in Rust is a bit more difficult especially when using javascript libraries, but still doable.

State of the Art

- Low level access to nearly all browser and Javascript APIs via the web-sys and js-sys crates.
- replacing some functions in your Javascript code with Rust is easy.
- writing a complete web app in Rust is a bit more difficult especially when using javascript libraries, but still doable.

The Future

The WebAssembly Working Group is currently working on *glue* which is intended to provide increasingly higher level abstractions for all use cases.

Where to go from here

- WebAssembly Main Site: https://webassembly.org/
- Rust and WebAssembly: https://rustwasm.github.io/
- Learn Rust: https://doc.rust-lang.org/book/