Daniele D'Orazio
Sviluppatore

Rust e WebAssembly

Develer webinar

16/09/2020

develer

CHI SIAMO

Develer un’Azienda che che progetta e realizza soluzioni hardware e software in ambiti industriali innovativi.

COSA FACCIAMO

- Sviluppo software
- Sistemi embedded
- Corsi
- Eventi

Hai bisogno dei nostri servizi?

www.develer.com/servizi

Vuoi lavorare con noi?

www.develer.com/lavora-con-noi

Piano del webinar

- Introduzione a WebAssembly
  - Idea generale
  - Vantaggi e Limitazioni
- Rust e WebAssembly
- Demo
- Q&A

WebAssembly (WASM) is a binary instruction format designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.

Vantaggi WebAssembly

- Velocità
- Sicurezza
- Interoperabilità
- Portabilità

Velocità

- Quasi velocità nativa
- Instruction set simile a quello delle CPU reali
- Streaming compilation

Sicurezza

- Sandbox
- Linear memory model
- Esecuzione quasi del tutto deterministica

Interoperabilità

- Language agnostic
- Opt-in

Portabilità

- Browser
- Server(less)
- Desktop

Limitazioni WebAssembly

- Gestione manuale della memoria
- Instruction set limitato (no SIMD, mutex, etc...)
- Non del tutto completo

Casi d'uso

- Multimedia
- AR/VR
- Blockchain
- Giochi
- CAD

Rust e WebAssembly

- No Garbage Collector o Runtime
- Zero Cost Abstractions
- Best in class tooling

Getting started - Requisiti

- Rust (Rustup)
- npm
- wasm-pack

Getting started - Crates

- wasm-bindgen
- web-sys
- js-sys

Getting started - Setup

```
$ npm init rust-webpack hello-wasm
$ npm install
$ npm start
```

Getting started - Setup

```
$ tree -L 1
.
├── Cargo.lock
├── Cargo.toml
├── js
├── node_modules
├── package.json
├── package-lock.json
├── pkg
├── README.md
├── src
├── static
├── target
├── tests
└── webpack.config.js
```

Getting started – index.{html,js}

```
<!DOCTYPE html>
<html>
 <head>
   <meta charset="UTF-8">
   <title>My Rust + Webpack project!</title>
 </head>
 <body>
   <script src="index.js"></script>
 </body>
</html>
```

```
import("../pkg/index.js").catch(console.error);
```

Getting started – src/lib.rs (redacted)

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

#[wasm_bindgen(start)]
pub fn main_js() -> Result<(), JsValue> {
    console::log_1(&JsValue::from_str("Hello world!"));

    Ok(())
}
```

Exporting a function to JS

```rust
#[wasm_bindgen]
pub fn fast_add(a: u32, b: u32) -> u32 {
    a + b
}
```

```js
import("../pkg/index.js")
    .catch(console.error)
    .then((w) => {
        console.log(w.fast_add(41, 1));
    });
```

Exporting a function to JS

```rust
#[wasm_bindgen]
pub fn fast_sort(mut v: Vec<u8>, msg: &str) -> Vec<u8> {
    console::log_2(&JsValue::from_str("msg:"), &JsValue::from_str(msg));
    v.sort();
    v
}
```

```
console.log(w.fast_sort([4, 5, 7, 9, 20], "sorting"));
```

Using DOM APIs

```
use std::sync::atomic::{AtomicU32, Ordering};

static COUNTER: AtomicU32 = AtomicU32::new(0);

#[wasm_bindgen]
pub fn increment_counter() {
    let prev_val = COUNTER.fetch_add(1, Ordering::SeqCst);

    let document = web_sys::window().unwrap().document().unwrap();
    let num_div = document.get_element_by_id("num").expect("num div not found");

    num_div.set_inner_html(&format!("counter: {}", prev_val + 1));
}
```

Using DOM APIs

```
[dependencies.web-sys]
features = ["console", "Document", "Window", "Element"]
```

Using DOM APIs

```
<button id="num-btn">Click me!</button>
<div id="num"></div>
```

```
document.getElementById("num-btn").onclick = w.increment_counter;
```

Passing structured data to JS

```
use serde::Serialize;

#[derive(Serialize)]
pub struct Circle {
    pub x: f32,
    pub y: f32,
    pub r: f32,
}
#[wasm_bindgen]
pub fn pack_circles(width: f32, height: f32, coverage: f32, minr: f32, padding: f32) -> JsValue {
    let circles = pack_circles_impl(width, height, coverage, minr, padding);
    JsValue::from_serde(&circles).unwrap()
}

fn pack_circles_impl(width: f32, height: f32, coverage: f32, minr: f32, pad: f32) -> Vec<Circle>;
```

Passing structured data to JS

```
const canv = document.getElementById("packing-canvas");
const ctx = canv.getContext("2d");

canv.onclick = () => {
    const circles = w.pack_circles(ctx.canvas.width, ctx.canvas.height, 0.8, 5.0, 5.0);
    ctx.clearRect(0, 0, width, height);

    for (const c of circles) {
        ctx.beginPath();
        ctx.arc(c.x, c.y, c.r, 0.0, 2.0 * Math.PI);
        ctx.fill();
    }
};
```

Demo

- Demo
- Repo

Risorse utili

- https://webassembly.org/
- https://rustwasm.github.io/
- https://bytecodealliance.org/

Daniele D'Orazio

daniele@develer.com

Vuoi rimanere aggiornato sugli eventi Develer?
Seguici nei nostri canali social:

develer

www.develer.com