Node.js N-API for Rust

Alexey Orlenko
@aqrln

github.com/aqrln/jsfest2019-napi-rust

N-API is an API for building native addons that is independent from the underlying JavaScript runtime and is maintained as part of Node.js itself. It also allows modules compiled for one major version of Node.js to run on later ones without recompiling.

See https://nodejs.org/dist/latest-v13.x/docs/api/n-api.html

Pure C API allows to write native addons in virtually any compiled programming language, from C++ and Rust to OCaml and Haskell.

* although using languages with managed memory and their own runtimes might be a bit tricky, probably

In fact, there's even an official C++ wrapper around N-API:

https://github.com/nodejs/node-addon-api

Rust:

- fast
- modern
- integrates seamlessly with C APIs with no overhead
- great ecosystem and even greater community

https://neon-bindings.com

- + stable and mature
- + good documentation
- - N-API backend is not ready yet
- - and maintaining own C wrappers for raw C++ Node.js API has certain drawbacks

N-API

+ can be used directly from Rust literally the same way one would use it from C
- too low-level to write application code directly in terms of its API
+ but flexible enough to build a nice type-safe Rust API on top of it

If you want to follow the presentation on your own machine or to tweak with the code later, clone the repo (github.com/aqrln/jsfest2019-napi-rust) and then either (a, b or c):

a) If you use VS Code, install the Remote Development plugin (ms-vscode-remote.vscode-remote-extensionpack) and choose to open the folder in container.

b) If you don't use VS Code, but you do use Docker, run:

```
$ docker build . -t napi-rust
$ docker run --it -v
  $(pwd):/workspaces/napi-rust
  napi-rust bash
```

c) Alternatively, install the required software on your host machine:

- Node.js
- Rust
- npm install node-gyp benchmark
- cargo install bindgen

Our plan:

1. Get started with using N-API in general, discuss JavaScript values representation and error handling
2. Learn how to generate Rust bindings from C headers

Our plan:

3. Implement type-safe convenience API on top of N-API
4. Utilize heavy metaprogramming in form of procedural macros to eliminate boilerplate code

Our plan:

5. Do some parallel computations and benchmark what we got
6. Release a new version of https://crates.io/crates/napi and pretend it didn’t take almost two years

Now let's switch to the editor

Recap:

- N-API, initially conceived to make Node.js API VM-agnostic and forward-compatible (both API and ABI wise), has a nice side-effect of allowing to use languages other than C++ effortlessly

Recap:

- Rust can interoperate with C APIs directly with no overhead. We only used a tiny bit of C to integrate with node-gyp for the sake of simplicity, but that is not strictly necessary: Neon has its own build tool in pure Rust, for example.

Recap:

- Procedural macros in Rust 2018 allowed us to to declare N-API callbacks with automatic parameters validation and all necessary error handling extremely concisely

(cont)

Rust 2015 only supported procedural macros that automatically derive traits for structs, so we had to pack all parameters of a callback into a structure and create a wrapper function with a separate macro

PSA:

Contact me if

- you might want to be a maintainer of https://crates.io/crates/napi or
- you are learning Rust and you are looking for a task and a mentor

@aqrln

GitHub/Twitter