Node.js N-API for Rust

Alexey Orlenko
@aqrln

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-v14.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

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

https://neon-bindings.com

- ➜ stable and mature, stellar API and documentation
- ➜ N-API backend is experimental and not enabled by default, but it's happening 🎉

https://github.com/napi-rs

- ➜ Previously old and abandoned project of mine
- ➜ Now a new and flourishing project by @Brooooooklyn that we moved to the org

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. Compare how native addons relate to WebAssembly (especially now that we have WASI) and when you should use one or the other

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

But what about WebAssembly?

I'd suggest to use WASM and WASI when it is feasible — it's more portable and future-proof.

As of now though, native addons are still the preferred option if:

- → **You need (or can take significant advantage of) thread-level parallelism**

(cont)

As of now, native addons are still the preferred option if:

→ **You need to transfer complex JS objects across the boundary**

(cont)

As of now, native addons are still the preferred option if:

- → You need to call back into JavaScript a lot from the native code

@aqrln

GitHub/Twitter