Writing Node.js Modules in Rust

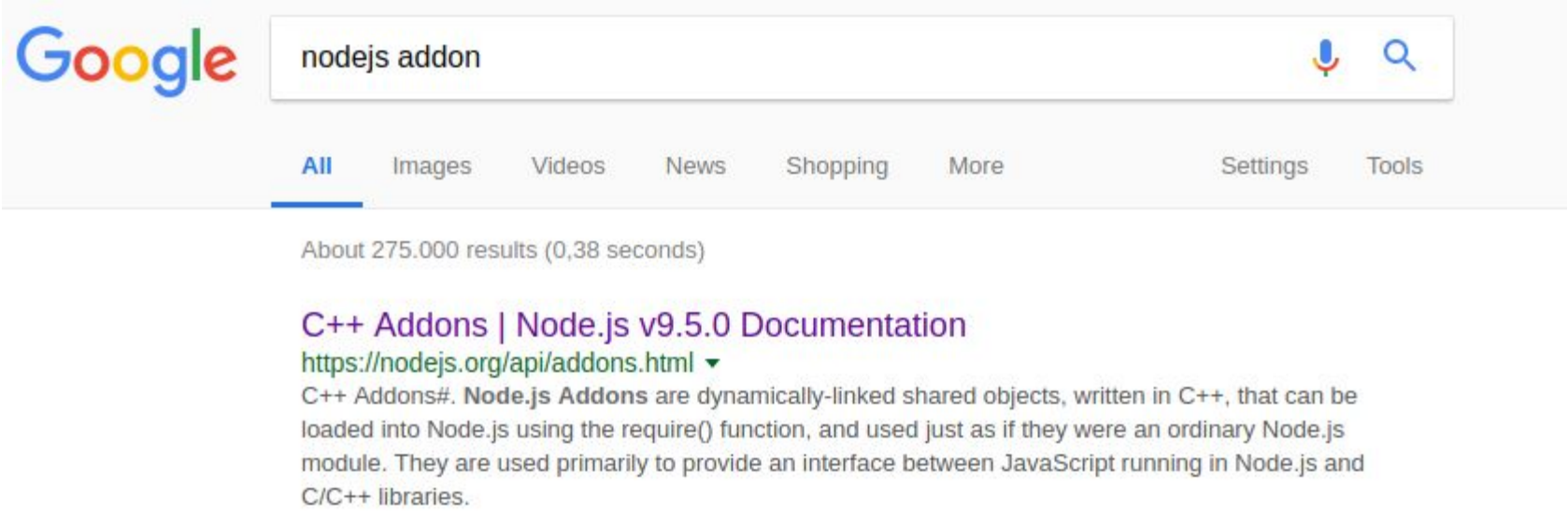
Farid Nouri Neshat
Software Engineer @ Olindata

OlinData

You need a Node.js Native Addon because

- Performance
- Native Library
- Freedom in memory management
- OS Low level APIs
- You don’t like Javascript!

Well… How to write a node.js addon?



OlinData

But you probably don’t like C/C++ as well!

OlinData

Image Taken from https://www.neon-bindings.com/

```
farid@Dellizium:~/repos/nr$ npm install neon-cli -g
/home/farid/.nvm/versions/node/v8.9.4/bin/neon -> /home/farid/.nvm/versions/node/v8.9.4/lib/node_modules/neon-cli/bin/cli.js
+ neon-cli@0.1.22
updated 1 package in 5.001s
farid@Dellizium:~/repos/nr$ neon new hi
This utility will walk you through creating the hi Neon project.
It only covers the most common items, and tries to guess sensible defaults.

Press ^C at any time to quit.
? version 0.1.0

? description

? node entry point lib/index.js

? git repository

? author Farid Neshat

? email FaridN_SOAD@yahoo.com

? license MIT


Woo-hoo! Your Neon project has been created in: hi

The main Node entry point is at: hi/lib/index.js
The main Rust entry point is at: hi/native/src/lib.rs

To build your project, just run npm install from within the hi directory.
Then you can test it out with node -e 'require("./")'.

Happy hacking!
farid@Dellizium:~/repos/nr$ cd hi/
farid@Dellizium:~/repos/nr/hi$ ls
lib  native  package.json  README.md
```

```rust
#[macro_use]
extern crate neon;

use neon:: ...

fn hello(call: Call) -> JsResult<JsString> {
    let scope = call.scope;
    Ok(JsString::new(scope, "hello node").unwrap())
}

register_module!(m, {
    m.export("hello", hello)
});
```

```javascript
var addon = require('../native');

console.log(addon.hello());
```

```
farid@Dellizium:~/repos/nr/hi$ neon build
neon info running cargo
   Compiling hi v0.1.0 (file:///home/farid/repos/nr/hi/native)
    Finished release [optimized] target(s) in 0.36 secs
neon info generating native/index.node
farid@Dellizium:~/repos/nr/hi$ node lib/index.js
hello node
```

OlinData

Let’s talk about performance

- Don’t underestimate Javascript
- Try to minimize crossing the Rust-JS bridge
- Interacting with Javascript objects can potentially invoke Javascript
- Sometimes (node.js) buffers are faster than strings

```
fn get_object_sync(call: Call) -> JsResult<JsObject> {

        let scope = call.scope;
        let object = JsObject::new(scope);

        try!(object.set("prop1", JsInteger::new(scope, 1)));
        try!(object.set("prop2", JsInteger::new(scope, 2)));
        try!(object.set("prop3", JsInteger::new(scope, 3)));

        Ok(object)
}
```

OlinData

Scope?

- Javascript is garbage collected
- V8 needs hints on objects created outside Javascript

```
fn hello(call: Call) -> JsResult<JsString> {
    Ok(JsString::new(call.scope, "hello node").unwrap())
}
```

OlinData

Buffers?

```
use neon::vm::{Lock, JsResult, Call};
use neon::js::binary::JsBuffer;

pub fn hello(call: Call) -> JsResult<JsBuffer> {
  let scope = call.scope;
  let mut arg_buf = call.arguments.require(scope, 0)?.check::<JsBuffer>().unwrap();
  let arg = arg_buf.grab(|contents| contents.as_slice());
  let mut result_buf = JsBuffer::new(scope, arg.len() as u32)?;
  result_buf.grab(|mut contents| { contents.as_mut_slice().copy_from_slice(arg) });

  Ok(result_buf)
}
```

OlinData

Classes

```
pub struct Greeter {
    greeting: String,
}

declare_types! {

    /// A class for generating greeting strings.
    pub class JsGreeter for Greeter {
        init(call) {
            let scope = call.scope;
            let greeting = call.arguments.require(scope, 0)?.to_string(scope)?.value();
            Ok(Greeter {
                greeting: greeting
            })
        }

        method hello(call) {
            let scope = call.scope;
            let name = call.arguments.require(scope, 0)?.to_string(scope)?.value();
            let msg = call.arguments.this(scope).grab(|greeter| {
                format!("{}, {}!", greeter.greeting, name)
            });
            Ok(JsString::new_or_throw(scope, &msg[..])?.upcast())
        }
    }

}
```

OlinData

Use Tasks for background jobs

```rust
impl Task for SuccessTask {
    type Output = i32;
    type Error = String;
    type JsEvent = JsNumber;

    fn perform(&self) -> Result<Self::Output, Self::Error> {
        Ok(CalculateFibonacci())
    }

    fn complete<'a, T: Scope<'a>>(
        self,
        scope: &'a mut T,
        result: Result<Self::Output, Self::Error>,
    ) -> JsResult<Self::JsEvent> {
        Ok(JsNumber::new(scope, result.unwrap() as f64))
    }
}
```

```rust
pub fn perform_async_task(call: Call) -> JsResult<JsUndefined> {
    let f = call.arguments.require(call.scope, 0)?
        .check::<JsFunction>()?;
    SuccessTask.schedule(f);
    Ok(JsUndefined::new())
}
```

OlinData

One more tip: Try neon-serde!

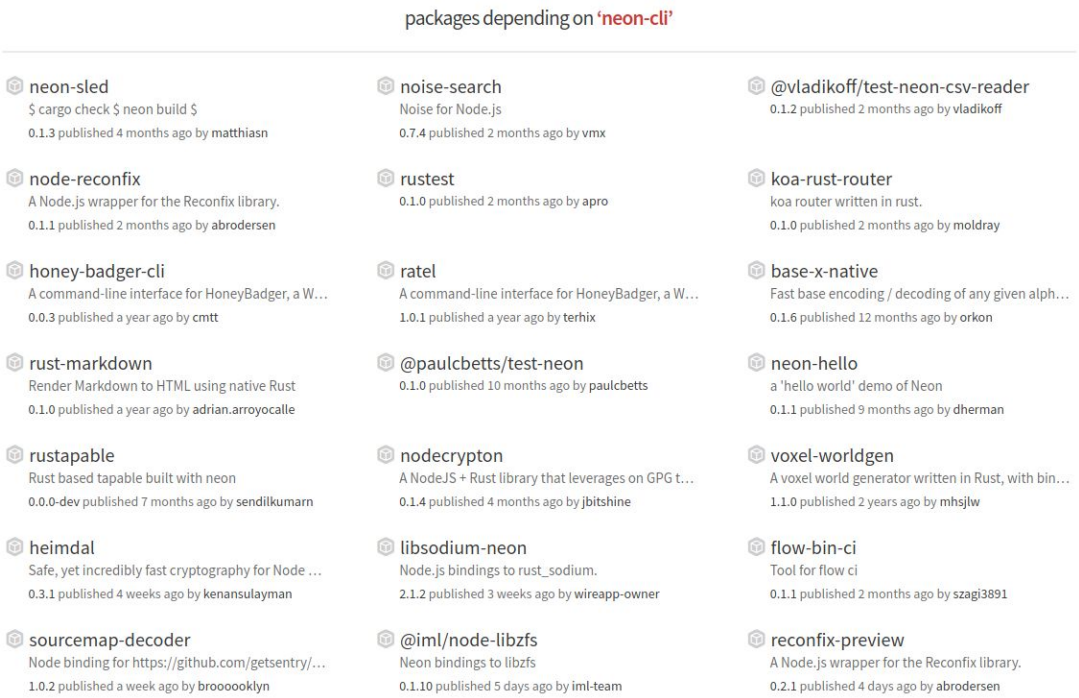
```
export! {
    fn say_hello(name: String) -> String {
        format!("Hello, {}!", name)
    }

    fn greet(user: User) -> String {
        format!("{} is {} years old", user.name, user.age)
    }

    fn fibonacci(n: i32) -> i32 {
        match n {
            1 | 2 => 1,
            n => fibonacci(n - 1) + fibonacci(n - 2)
        }
    }
}
```

OlinData

Who uses neon?



OlinData

Links

- https://neon-bindings.com
- https://github.com/GabrielCastro/neon-serde
- Examples:
  - https://github.com/dherman/neon-examples
  - https://github.com/nswbmw/neon-fibonacci
  - https://github.com/nswbmw/neon-class
  - https://github.com/neon-bindings/neon/issues/260
  - https://github.com/neon-bindings/neon/tree/master/test/dynamic/native/src/js

OlinData

Questions?

One last thing

- All credits goes to the Neon maintainer Dave Herman and the Neon contributers and community.