Ruby on Rust

Joe Corcoran · corcoran.io · @josephcorcoran

What?

What is a dynamic library?

Why shared libraries?

Why in Ruby?

libffi

→ Foreign function interface

→ Created in 1996

→ Commonly used to connect compiled -> interpreted languages

DL

```
module Something
  extend DL::Importer
  dlload './something.so'
  extern 'int foo(int)'
end
```

Ruby already has a library called DL... [it] is a bit arcane though...

— rubyinside.com/ruby-ffi-library-calling-external-libraries-now-easier-1293.html

FFI

github.com/ffi/ffi

```
module Something
  extend FFI::Library
  ffi_lib './something.so'
  attach_function :foo, [:int], :int
end
```

FFI

```
pointer = FFI::MemoryPointer.new(8)
pointer.write_array_of_int([1, 2])
pointer.read_array_of_int(2)
#=> [1, 2]
```

Fiddle

```
module Something
  extend Fiddle::Importer
  dlload './something.so'
    extern 'int foo(int)'
end
```

Rust

libc

```
extern crate libc;
```

Rust

libc

```
extern crate libc;

use libc::c_int;
```

Rust

Tests

```
#[cfg(test)]
mod tests {
    use super::*;

    // tests go here
}
```

Integers

Rust

```rust
#[no_mangle]
pub extern fn add_one(a: c_int) -> c_int {
    a + 1
}
```

Integers

Rust

```rust
#[no_mangle]
pub extern fn add_one(a: c_int) -> c_int {
    a + 1
}

#[test]
fn test_add_one() {
    assert_eq!(2, add_one(1));
}
```

Integers

Rust

```
$ cargo test

running 1 test
test tests::test_add_one ... ok
test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured
```

Integers

Rust

```
$ cargo build --release
```

Integers

Ruby

```ruby
require 'fiddle'
lib = Fiddle.dlopen(
  File.expand_path('../rust/target/release/libfoo.so', __FILE__)
)

add_one = Fiddle::Function.new(
  lib['add_one'],
  [Fiddle::TYPE_INT],
  Fiddle::TYPE_INT
)

add_one.call(1)   #=> 2
```

Arrays

Rust

```
#[repr(C)]
pub struct IntArray {
    length: c_int,
    members: *const c_int
}
```

Arrays

Ruby

```
require 'fiddle/struct'

IntArray = Fiddle::CStructBuilder.create(
  Fiddle::CStruct,
  [Fiddle::TYPE_INT, Fiddle::TYPE_VOIDP],
  ['length', 'members']
)
```

Arrays

Rust

```
use std::slice;

#[no_mangle]
pub extern fn head(a: &IntArray) -> c_int {
    unsafe {
        let slice = slice::from_raw_parts(a.members, a.length as usize);
        *slice.first().unwrap()
    }
}
```

Arrays

Ruby

```
head = Fiddle::Function.new(
  lib['head'],
  [Fiddle::TYPE_VOIDP],
  Fiddle::TYPE_INT
)
```

Arrays

Ruby

```
array = [3, 2, 1]
packed = IntArray.malloc
packed.length = array.length
packed.members = array.pack('l*')

head.call(packed)   #=> 3
```

More information

→ github.com/joecorcoran/talks

→ Using Rust with Ruby, a deep dive with Yehuda Katz

→ github.com/steveklabnik/rust_example

→ rust-lang.org