http://mizzy.org/fkrk02.pdf

スライドはこちら。コード多めなのでお手元で見た方が見やすいかもしれません。

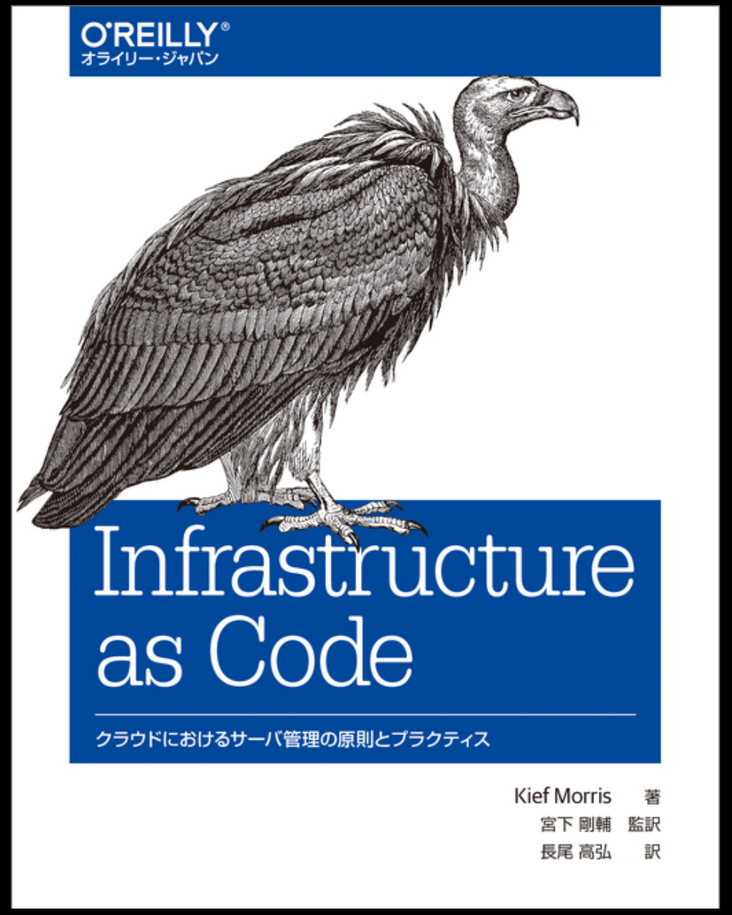
Rustで書いたライブラリをRuby/mrubyから呼び出す実践的な方法

自己紹介

- 宮下 剛輔
- 合同会社Serverspec Operations代表
- フリーランスのソフトウェアエンジニア
- http://mizzy.org/
- mizzy@github, gosukenator@twitter

著書・監訳書


O'REILLY
オライリー・ジャパン
Serverspec
宮下 剛輔 著


O'REILLY
オライリー・ジャパン
Infrastructure as Code
クラウドにおけるサーバ管理の原則とプラクティス
Kief Morris 著
宮下 剛輔 監訳
長尾 高弘 訳

今日の発表に至る経緯

- libspecinfraというOSSプロジェクトをやっている
- このプロジェクトは以下の構成要素を持つ
  - Rustで実装した共有ライブラリ
  - 共有ライブラリを利用するための各種言語バインディング

今日の発表に至る経緯（つづき）

- Ruby/mrubyバインディングを開発するための方法を調べた
- 初歩的な情報はたくさんあるが実践的な情報は案外少ない
- 色々試行錯誤したのでまとめておこうと思った
- 間違いやより良いやり方があれば識者が指摘してくれそう
- **久々に福岡に行く口実が欲しかった**

アジェンダ

- RustとFFI
- 入門Rust + Ruby/mruby
- 実践Rust + Ruby/mruby
- libspecinfraプロジェクトの概要（時間があれば）
- ※Rust等の言語仕様的な話はしません

RustとFFI

FFIとは

- Foreign Function Interface
- あるプログラミング言語から他のプログラミング言語で定義された関数などを利用するための機構
- Rust、Ruby、mrubyに限らない

RustとFFI

- RustにはFFIが組み込まれており公式ドキュメントもある
- 主な機能は大きく2つ
  - 他言語で書かれたライブラリの関数を呼ぶ機能
  - **他言語（主にC）から関数を呼べるようにする機能**

他言語で書かれたライブラリ関数を呼ぶ

```
#[link(name = "snappy")]
extern {
    fn snappy_max_compressed_length(source_length: size_t) -> size_t;
}

fn main() {
    let x = unsafe { snappy_max_compressed_length(100) };
    println!("max compressed length of a 100 byte buffer: {}", x);
}
```

https://doc.rust-lang.org/book/first-edition/ffi.html#calling-foreign-functions

他言語から関数を呼べるようにする

```
#[no_mangle]
pub extern "C" fn hello_rust() -> *const u8 {
    "Hello, world!\0".as_ptr()
}
```

https://doc.rust-lang.org/book/first-edition/ffi.html#calling-rust-code-from-c

hello_rust()をCから呼び出す

```
#include <stdio.h>

extern char *hello_rust(void);

int main() {
    printf("%s\n", hello_rust());
}
```

```
cc hello.c -L. -lhello
LD_LIBRARY_PATH=. ./a.out
Hello, world!
```

RustとFFIまとめ

- FFI = Foreign Function Interface
- あるプログラミング言語から他のプログラミング言語で定義された関数などを利用するための機構
- RustにはFFIが組み込まれていて、他の言語で書かれたライブラリの関数を呼び出したり、他の言語からRustで書いたライブラリの関数を呼び出したりできる

入門Rust

+

Ruby/mruby

Rustで書かれたライブラリをどう呼ぶか

- Ruby
  - native extension
  - **ffi gemを利用してlibffi経由で呼び出す**
- mruby
  - **Cで拡張(mrbgem)を書く**

入門Rust + Ruby/mruby

- RustとRuby/mrubyをどう繋ぐか具体的なコードで示します
- 自分がlibspecinfraのバインディング開発のために学んだのと同じステップを辿ってみます
- 文字を大きくしてスライドを見やすくするためコードを省略したり推奨されない書き方をしてる部分もあります
- 完全なコードはGitHubに置いてあります
  - https://github.com/mizzy/fukuokark02_code_examples
  - スライドと微妙に違う部分もありますが

整数

整数の足し算を行うRustコード

```
extern crate libc;
use libc::uint32_t;

#[no_mangle]
pub extern "C" fn addition(a: uint32_t, b: uint32_t)
    -> uint32_t {
    a + b
}
```

Rustの関数を呼び出すRubyコード

```ruby
require 'ffi'

module Integers
  extend FFI::Library
  ffi_lib 'addition'
  attach_function :addition, [:uint32, :uint32], :uint32
end

puts Integers.addition(1, 2)
```

Rustの関数を呼び出すmrbgemコード

```
extern uint32_t addition(uint32_t, uint32_t);
static mrb_value addition_(mrb_state *mrb, mrb_value self)
{
    mrb_int a, b;
    mrb_get_args(mrb, "ii", &a, &b);
    return mrb_fixnum_value(addition(a, b));
}
void mrb_mruby_addition_gem_init(mrb_state *mrb)
{
    struct RClass *i = mrb_define_module(mrb, "Integers");
    mrb_define_class_method(
      mrb, i, "addition", addition_, MRB_ARGS_REQ(2)
    );
}
```

mrbgemを利用したmrubyコード

```
puts Integers.addition(1, 2)
```

文字列

文字列に!!!を付加するRustコード

```
pub extern "C" fn emphasize(ptr: *const c_char)
    -> *const c_char {
    let s = unsafe {
            assert!(!ptr.is_null());
            CStr::from_ptr(ptr)
        }.to_str().unwrap();
    let str = s.to_owned() + "!!!";
    CString::new(str).unwrap().into_raw()
}
```

Rustの関数を呼び出すRubyコード

```ruby
module Strings
  extend FFI::Library
  ffi_lib 'emphasize'
  attach_function :emphasize, [:string], :string
end

puts Strings.emphasize("Hello")
```

Rustの関数を呼び出すmrbgemコード

```
extern char *emphasize(char *);
static mrb_value emphasize_(mrb_state *mrb, mrb_value self)
{
    char *arg, *ret;
    mrb_int len;
    mrb_value str;
    mrb_get_args(mrb, "s", &arg, &len);
    ret = emphasize(arg);
    str = mrb_str_buf_new(mrb, sizeof(ret));
    return mrb_str_cat_cstr(mrb, str, ret);
}
```

mrbgemを利用したmrubyコード

```
puts Strings.emphasize("Hello")
```

構造体

構造体

- これは省略
- 一応書きかけのコードはサンプルリポジトリにあります
- 以下の記事が参考になります
  - Ruby-FFIについて調べてみた。（その2） - いものやま。
    - http://yamaimo.hatenablog.jp/entry/2015/05/22/200000
  - mruby で C 言語の構造体をラップしたオブジェクトを作る正しい方法 - Qiita
    - https://qiita.com/tsahara@github/items/86610a696f8ca792db45

構造体

- 割と面倒くさい
- Rust側とRuby/mrubyで同じ構造体を定義する必要がある
- 構造体の中身はRuby/mrubyから見ればどうでもいい（自分が開発してるlibspecinfraでは）
- オブジェクトとして扱いたい
- Rust側のstructを示すポインタとそれを扱う関数だけあれば十分では？

入門Rust + Ruby/mrubyまとめ

- 整数引数と返り値、文字列引数と返り値を扱うコード例を示した
- 構造体については、libspecinfra開発のために調べてる途中で、これは自分のやりたいこととは違う、と思ったので略
- 構造体(Rustのstruct)を構造体として扱いたいのではなく、オブジェクトとして扱いたい
- そのやり方は次で解説します

実践Rust

+

Ruby/mruby

実践Rust + Ruby/mruby

- ここからは実際のlibspecinfraのコードをベースとした解説をします
- 完全に同じコードではなく簡略化してあります
- こちらのコードもGitHubにあります
  - https://github.com/mizzy/fukuokark02_code_examples

オブジェクト

オブジェクトを扱うコード(Rust)

こんな感じでファイルのパーミッションを取得したい

```
let s = Specinfra::new();
let f = s.file("/etc/passwd");
println!("{:o}", f.mode());
```

Specinfraオブジェクトの定義(Rust)

```rust
pub struct Specinfra;

impl Specinfra {
    pub fn new() -> Specinfra {
        Specinfra
    }

    pub fn file(self, name: &str) -> File {
        File { name: name }
    }
}
```

Fileオブジェクトの定義(Rust)

```
pub struct File<'a> {
    name: &'a str,
}

impl<'a> File<'a> {
    pub fn mode(self) -> i32 {
        // パーミッションを取得して返す処理
    }
}
```

Specinfraオブジェクトの外部用定義(Rust)

```
pub extern "C" fn specinfra_new() -> *const Specinfra {
    let s = Specinfra::new();
    Box::into_raw(Box::new(s))
}
pub extern "C" fn specinfra_file<'a>(ptr: *const Specinfra,
                                     name: *const c_char)
                                     -> *const File<'a> {
    let s = unsafe { &*ptr };
    let name = unsafe { CStr::from_ptr(name) };
    Box::into_raw(Box::new(s.file(name.to_str().unwrap())))
}
```

Fileオブジェクトの外部用定義(Rust)

```
pub extern "C" fn file_mode(ptr: *const File)
    -> int32_t {
    let f = unsafe {
        assert!(!ptr.is_null());
        &*ptr
    };
    f.mode()
}
```

オブジェクト開放用の関数(Rust)

```
pub extern "C" fn specinfra_free(ptr: *mut Specinfra) {
    unsafe { Box::from_raw(ptr); }
}

pub extern "C" fn file_free(ptr: *mut File) {
    unsafe { Box::from_raw(ptr); }
}
```

オブジェクトを扱うコード(Ruby)

こんな感じでファイルのパーミッション取得したい

```
s = Specinfra::Binding.new
f = s.file("/etc/passwd")
printf("%#o\n", f.mode)
```

Specinfraオブジェクト(Ruby)

```
class Specinfra < FFI::AutoPointer
  def self.release(ptr)
    Binding.free(ptr)
  end
  def file(name)
    Binding.file(self, name)
  end
  module Binding
    attach_function :new,  :specinfra_new,  [], Specinfra
    attach_function :free, :specinfra_free, [Specinfra], :void
    attach_function :file, :specinfra_file, [Specinfra, :string], File
  end
end
```

Fileオブジェクト(Ruby)

```
class File < FFI::AutoPointer
  def self.release(ptr)
    file_free(ptr)
  end
  def mode()
    file_mode(self)
  end
  attach_function :file_free, [File], :void
  attach_function :file_mode, [File], :int
end
```

オブジェクトを扱うコード(mruby)

こんな感じでファイルのパーミッション取得したい

```
s = Specinfra.new
f = s.file("/etc/passwd")
printf("%#o\n", f.mode)
```

Fileオブジェクト関連定義(mrbgem)

```
typedef int file;

extern int file_mode(file *);
extern void file_free(file *);

struct mrb_data_type mrb_file_type
    = { "File", mrb_file_free };
```

Specinfraオブジェクト関連定義(mrbgem)

```
typedef int specinfra;

extern specinfra *specinfra_new(void);
extern file *specinfra_file(specinfra *, char *);
extern void specinfra_free(specinfra *);

struct mrb_data_type mrb_specinfra_type
    = { "Specinfra", mrb_specinfra_free };
```

Specinfra.new(mrbgem)

```
mrb_value specinfra_new_(mrb_state *mrb, mrb_value self)
{
    specinfra *s = specinfra_new();
    DATA_TYPE(self) = &mrb_specinfra_type;
    DATA_PTR(self) = s;
    return self;
}
```

Specinfra#file(mrbgem)

```
mrb_value specinfra_file_(mrb_state *mrb, mrb_value self)
{
    mrb_value v;
    mrb_get_args(mrb, "S", &v);
    char *name = mrb_str_to_cstr(mrb, v);
    file *f = specinfra_file(DATA_PTR(self), name);
    struct RClass file_class = mrb_class_get(mrb, "File");
    mrb_value file_object = mrb_obj_new(mrb, file_class, 0, NULL);
    DATA_TYPE(file_object) = &mrb_file_type;
    DATA_PTR(file_object) = f;
    return file_object;
}
```

Specinfraオブジェクト開放用関数(mrbgem)

```
void mrb_specinfra_free(mrb_state *mrb, void *ptr)
{
    specinfra_free(ptr);
}
```

File#mode(mrbgem)

```
mrb_value file_mode_(mrb_state *mrb, mrb_value self)
{
    file *f;
    int m;

    f = DATA_PTR(self);
    m = file_mode(f);
    return mrb_fixnum_value(m);
}
```

Fileオブジェクト開放用関数(mrbgem)

```
void mrb_file_free(mrb_state *mrb, void *ptr)
{
    file_free(ptr);
}
```

初期化(mrbgem)

```
void mrb_mruby_object_gem_init(mrb_state *mrb)
{
    struct RClass *s
        = mrb_define_class(mrb, "Specinfra", mrb->object_class);
    mrb_define_method(mrb, s, "initialize", specinfra_new_, MRB_ARGS_NONE());
    mrb_define_method(mrb, s, "file", specinfra_file_, MRB_ARGS_REQ(1));

    struct RClass *f
        = mrb_define_class(mrb, "File", mrb->object_class);
    mrb_define_method(mrb, f, "mode", file_mode_, MRB_ARGS_NONE());
}
```

トレイト

トレイトを扱うコード(Rust)

こんな感じでBackendトレイトを実装したDirectオブジェクトを渡したい

```
let b = Direct::new();
let s = Specinfra::new(b);
```

Specinfraオブジェクトの修正(Rust)

```
pub struct Specinfra<'a> {
    backend: &'a Backend,
}

impl<'a> Specinfra<'a> {
    pub fn new(b: &Backend) -> Specinfra {
        b.detect_platform();
        Specinfra { backend: b }
    }
    ...
```

Backendトレイトの定義と実装(Rust)

```rust
pub trait Backend {
    fn detect_platform(&self) -> &str;
}

pub struct Direct;

impl Backend for Direct {
    fn detect_platform(&self) -> &str {
        // プラットフォーム判別処理
    }
}
```

Specinfraオブジェクトの外部用定義(Rust)

引数でトレイトを受け取るようにしていると...

```
pub extern "C" fn specinfra_new<'a>(ptr: *const Backend)
    -> *const Specinfra<'a> {
    let b = unsafe { &*ptr };
    let s = Specinfra::new(b); // この処理中の…
    Box::into_raw(Box::new(s))
}
```

セグフォ...(Rust)

```
impl<'a> Specinfra<'a> {
    pub fn new(b: &Backend) -> Specinfra {
        b.detect_platform(); // ここでセグフォする

        Specinfra { backend: b }
    }
```

BackendWrapper structの導入(Rust)

トレイトは直接引数で渡さずBackendWrapperでくるむ

```
pub struct BackendWrapper(pub Box<Backend>);

pub extern "C" fn specinfra_new<'a>(ptr: *const BackendWrapper)
    -> *const Specinfra<'a> {
    let b = unsafe { &*ptr };
    let s = Specinfra::new(&*b.0);
    Box::into_raw(Box::new(s))
}
```

Directオブジェクトの外部用定義(Rust)

```
pub extern "C" fn direct_new() -> *const Direct {
    let d = Direct::new();
    Box::into_raw(Box::new(d))
}
```

を以下のようにする

```
pub extern "C" fn direct_new() -> *const BackendWrapper {
    let d = Direct::new();
    let b = BackendWrapper(Box::new(d));
    Box::into_raw(Box::new(b))
}
```

トレイトを扱うコード(Ruby)

こう書きたい

```
b = Direct::Binding.new
s = Specinfra::Binding.new(b)
f = s.file("/etc/passwd")
printf("%#o\n", f.mode)
```

Directオブジェクトの定義(Ruby)

```
class Direct < FFI::AutoPointer
  def self.release(ptr)
    Binding.free(ptr)
  end

  module Binding
    attach_function :free, :direct_free, [Direct], :void
    attach_function :new,  :direct_new,  [],       Direct
  end
end
```

Specinfraオブジェクトの修正(Ruby)

```
class Specinfra < FFI::AutoPointer
  ...

  module Binding
    ...
    attach_function :new,  :specinfra_new,  [:pointer],  Specinfra
    ...
  end
end
```

トレイトを扱うコード(mruby)

こう書きたい

```
b = Direct.new
s = Specinfra.new(b)
f = s.file("/etc/passwd")
printf("%#o\n", f.mode)
```

トレイト関連定義(mrbgem)

```
typedef int backend;

extern specinfra *specinfra_new(backend *);
extern backend *direct_new(void);

struct mrb_data_type mrb_direct_type = { "Direct", mrb_free };
```

Direct.new(mrbgem)

```
mrb_value direct_new_(mrb_state *mrb, mrb_value self)
{
    backend *b;
    b = direct_new();
    DATA_TYPE(self) = &mrb_direct_type;
    DATA_PTR(self) = b;
    return self;
}
```

Specinfra.newの修正(mrbgem)

```
mrb_value specinfra_new_(mrb_state *mrb, mrb_value self)
{
    mrb_value b;
    mrb_get_args(mrb, "o", &b);
    specinfra *s = specinfra_new(DATA_PTR(b));
    DATA_TYPE(self) = &mrb_specinfra_type;
    DATA_PTR(self) = s;
    return self;
}
```

初期化(mrbgem)

```
void mrb_mruby_trait_gem_init(mrb_state *mrb)
{
    ...
    s = mrb_define_class(mrb, "Specinfra", mrb->object_class);
    mrb_define_method(mrb, s, "initialize", new, MRB_ARGS_REQ(1));

    ...

    struct RClass *d
        = mrb_define_class(mrb, "Direct", mrb->object_class);
    mrb_define_method(mrb, d, "initialize", direct_new_, MRB_ARGS_NONE());
}
```

エラー

エラー処理

- Rust側で発生したエラーを外部言語にどう伝えるのか、を自分なりに考えてみた
- もっといい方法あれば教えてください

エラーを発生させるコード(Rust)

パーミッション取得時にエラーが発生することを想定

```
pub struct File<'a> {
    name: &'a str,
    error: &'a str,
}

impl<'a> File<'a> {
    pub fn mode(self) -> Result<i32, &'a str> {
        // パーミッションを取得して返す処理を入れる

    }
}
```

パーミッション取得関数の外部定義を修正(Rust)

```
pub extern "C" fn file_mode(ptr: *mut File) -> int32_t {
    let f = unsafe { &mut *ptr };
    match f.mode() {
        Ok(mode) => mode,
        Err(e) => {
            f.error = e;
            -1
        }
    }
}
```

エラーメッセージ取得のための外部用関数定義(Rust)

```
pub extern "C" fn file_error(ptr: *const File)
    -> *const c_char {
    let f = unsafe { &*ptr };
    CString::new(f.error).unwrap().into_raw()
}
```

エラー処理(Ruby)

こう処理したい

```
b = Direct::Binding.new
s = Specinfra::Binding.new(b)
f = s.file("/etc/passwd")
begin
  printf("%#o\n", f.mode)
rescue => e
  puts e.message
end
```

File#modeを修正(Ruby)

```
class File < FFI::AutoPointer
  ...
  def mode()
    mode = file_mode(self)
    if mode == -1
      raise file_error(self)
    end
    mode
  end
  ...
  attach_function :file_error, [File], :string
end
```

エラー処理(mruby)

こう処理したい

```
b = Direct.new
s = Specinfra.new(b)
f = s.file("/etc/passwd")
begin
  printf("%#o\n", f.mode)
rescue => e
  puts e.message
end
```

File#modeの修正(mrbgem)

```
mrb_value mode(mrb_state *mrb, mrb_value self)
{
    file *f = DATA_PTR(self);
    int m = file_mode(f);
    if (m < 0) {
        mrb_raise(mrb, E_RUNTIME_ERROR, file_error(f));
    } else {
        return mrb_fixnum_value(m);
    }
}
```

実践Rust + Ruby/mrubyまとめ

- オブジェクト、トレイト、エラーを扱う方法について解説した
- この辺りあまり情報がなく、試行錯誤して辿り着いたやり方なので、間違いがあったり、もっといいやり方があるかもしれません

libspcinfraとは

libspecinfraとは

- SpecinfraのRustによる再実装プロジェクト

Specinfraとは

- Serverspecから派生したライブラリ
- OS/ディストリビューション毎のコマンドの違いの抽象化
- 直接、SSH経由、Docker API経由など実行形式の抽象化
- これらの抽象化によってシステム管理系ツールの開発を支援

Specinfraを用いたプロダクトのコード例

Serverspecによるテストコード

```
set :backend, :exec

describe package('nginx') do
  it { should be_installed }
end

describe service('nginx') do
  it { should be_enabled }
  it { should be_running }
end
```

Specinfraからlibspecinfraへ

- Specinfraはrubygemなので他の言語からは使えない
- libpseicnraは以下の形で多言語対応を目指す
  - 共有ライブラリを提供
  - 各種言語バインディングも提供

libspecinfra参考資料

- libspecinfra プロジェクトの概要と今後について | Advanced Technology Lab
  - http://atl.recruit-tech.co.jp/blog/4339/
- libspecinfra 開発者向けチュートリアル | Advanced Technology Lab
  - http://atl.recruit-tech.co.jp/blog/4349/
- libspecinfraの概要と現状と今後
  - https://speakerdeck.com/mizzy/overview-of-libspecinfra-project

なぜRust?

- 共有ライブラリを提供という目的ならCやC++がある
- でも新規でプロジェクトはじめるならRustかな、と
- C言語との親和性（≒他言語との親和性）
  - mrubyやlibffi等と繋ぎやすい
- Rustを覚えてみたかった

他言語連携を考慮したRust製共有ライブラリの開発方針

1. 初期段階では他言語連携は一切考慮せず、Rust単体で使うライブラリとして使いやすいコードにする
2. 1のフェーズで書いたコードを外部から呼び出すためのラッパー関数を書く
    - この時、1のフェーズで書いたコードは基本的に触らない
    - ラッパー関数では、プリミティブな値とポインタだけを扱う
3. 言語バインディングを書く
    - Rustでexportしてる関数そのままだと各言語の慣習に合わないので、慣習との差違をバインディングで吸収し使いやすくする

参考資料

- The Rust FFI Omnibus
  - http://jakegoulding.com/rust-ffi-omnibus/