Rパッケージで Rustを使うには: extendr入門

Tokyo.R#92

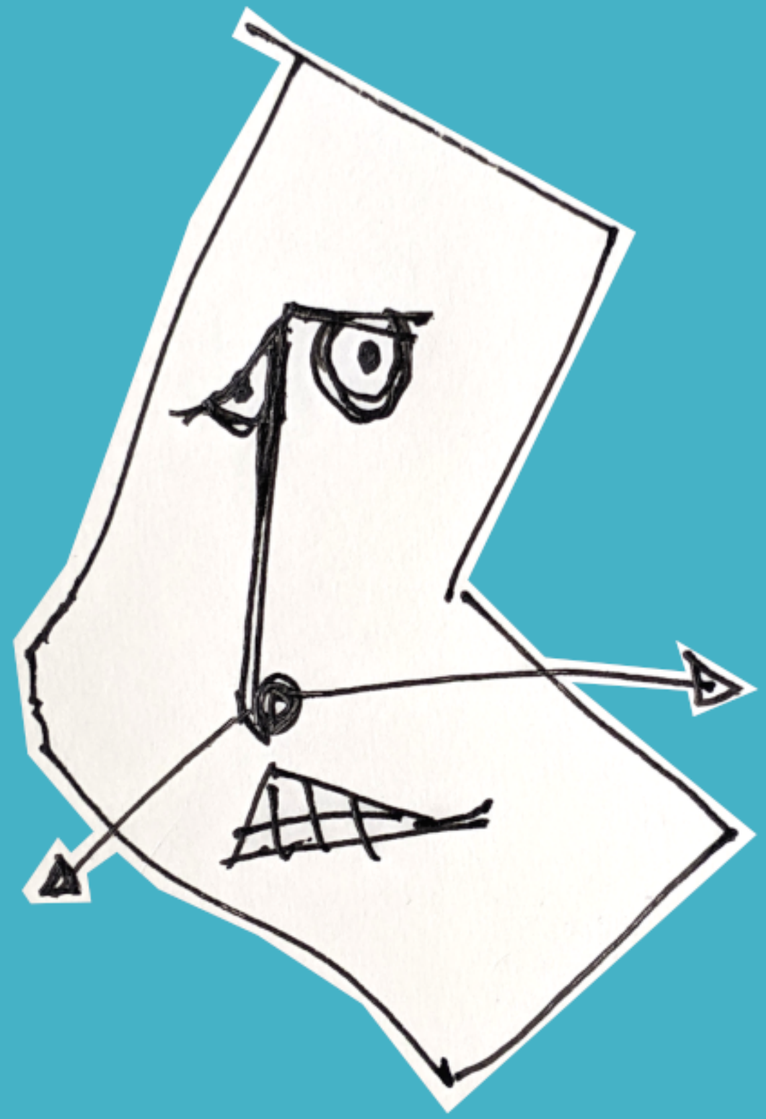
Hiroaki Yutani (@yutannihilation)

ドーモ！

:
@yutannihilation

好きな言語:
R、Rust、忍殺語

最近の趣味:
ガスコンロの電子楽器
をつくってます



Rユーザのための RStudio[実践]入門 第2版！

紙は6月3日、電子は5月31日発売です。

https://gihyo.jp/book/2

4-297-12170-9

extendr

extendrとは？

- RustとRを連携させるためのフレームワーク
- RからRustを使うだけではなく、RustからRを使うこともできる（つまり、Rustの中でggplot2を呼び出してプロットしたり、とかできるらしい）
- なぜか私も中の人です…

なぜRust？

→ そこにRustがあるから！！

（誰か教えてください…）

※今日話さないこと

- Rustの何が素晴らしいのか
- Rust入門
- Rust側からRを操作する方法
- R MarkdownのRust engineとか、パッケージ外でのextendrの使いみち

extendrの愉快な仲間たち

libR-sys（Rust）：
RのC APIにbindgenで生成したバインディング

extendr（Rust）：
libR-sysを使いやすくラップしたフレームワーク

rextendr（Rパッケージ）：
Rからextendrを使うためのユーティリティ
（usethisパッケージのような立ち位置）

準備

Rustのインストール

macOS / Linux:

- ふつうにRustをインストール（ググる）

Windows

- MSVCのtoolchainに加えて、64bit/32bit GNU用のtargetを追加する必要がある

```
rustup default stable-msvc
rustup target add x86_64-pc-windows-gnu
rustup target add i686-pc-windows-gnu
```

rextendrパッケージのインストール

- GitHubからインストール

```
devtools::install_github("extendr/rextendr")
```

パッケージのセットアップ

RStudioからパッケージ作成

Roxygenを使うように設定変更

NAMESPACEを上書き

```
usethis::use_namespace()
```

Build optionsを設定

Build > Configure Build Tools... > Generate documentation with Roxygen に☑を入れる

不要なファイルを削除

- `R/hello.R`
- `man/hello.Rd`

extendrのデフォルト設定を生成

```
rextendr::use_extendr()
```

```
✓ Creating src/rust/src.
✓ Writing 'src/entrypoint.c'
✓ Writing 'src/Makevars'
✓ Writing 'src/Makevars.win'
✓ Writing 'src/.gitignore'
✓ Writing src/rust/Cargo.toml.
✓ Writing 'src/rust/src/lib.rs'
✓ Writing 'R/extendr-wrappers.R'
✓ Finished configuring extendr for package myextendr.
• Please update the system requirement in DESCRIPTION file.
• Please run `rextendr::document()` for changes to take effect
```

生成されたファイル

```
.
├── R
│   └── extendr-wrappers.R
...
└── src
    ├── Makevars
    ├── Makevars.win
    ├── entrypoint.c
    └── rust
        ├── Cargo.toml
        └── src
            └── lib.rs
```

いじるファイル

src/rust:

extendrを使ったRustのcrate。開発のメインはここ。

基本いじらないファイル

Makevars, Makevars.win:

パッケージインストール時に`cargo build`が走るようにする設定。

entrypoint.c:

コンパイラにシンボルを勝手に消されないためのおまじない。

R/extendr-wrappers.R:

Rustの関数から自動生成されたRの関数。

src/rust/Cargo.toml

```
[package]
name = 'myextendr'
version = '0.1.0'
edition = '2018'

[lib]
crate-type = [ 'staticlib' ]

[dependencies]
extendr-api = '*'
```

src/rust/src/lib.rs（一部省略）

```rust
use extendr_api::prelude::*;

/// Return string `"Hello world!"` to R.
/// @export
#[extendr]
fn hello_world() -> &'static str {
    "Hello world!"
}

extendr_module! {
    mod myextendr;
    fn hello_world;
}
```

src/rust/src/lib.rs

- よく使う関数をまとめて読み込み

```
use extendr_api::prelude::*;
```

- `///`（3つ）のコメントはそのままRoxygenのコメントになる

```
/// Return string `"Hello world!"` to R.
/// @export
```

- これをつけるとRの関数が自動生成！

```
#[extendr]
```

src/rust/src/lib.rs

- 関数をエクスポートしてRが認識できるように登録（routine registration）してくれるマクロ。 新しく関数を追加したらここに入れる必要がある。

```
extendr_module! {
    mod myextendr;
    fn hello_world;
}
```

開発の流れ

開発の流れ

1. Rustのコードを編集
2. `rextendr::document()`でRのコードを自動生成（Rustのコードのコンパイルもこれがやってくれる）
3. （必要あれば）生成されたコードをRの側でいい感じにラップする
4. `devtools::load_all()`（やテスト）で動作確認

`rextendr::document()`

```
> rextendr::document()
✓ Saving changes in the open files.
ℹ Generating extendr wrapper functions for package: myextendr.
! No library found at src/myextendr.so, recompilation is requi
Re-compiling myextendr
─  installing *source* package ‘myextendr’ ... (382ms)
   ** using staged installation
   ** libs
   rm -Rf myextendr.so ./rust/target/release/libmyextendr.a en
   gcc -std=gnu99 -I"/usr/share/R/include" -DNDEBUG      -fpic
   cargo build --lib --release --manifest-path=./rust/Cargo.to
       Updating crates.io index
```

生成されるファイル

```
.
...
├── NAMESPACE                          ← @exportが反映される
├── R
│   └── extendr-wrappers.R             ← Rustの関数から自動生成
└── src
    ├── myextendr.so                   ← libmyextendr.aを使ってビルドされた
    │                                    共有オブジェクト
    └── rust
        └── target                     ← Rustの生成物が入るディレクトリ
            └── release
                ├── libmyextendr.a     ← Rustのコードの静的ライブラリ
                │                        （OSによって拡張子は違う）
                ...
```

自動生成されたRの関数

Rust

```rust
/// Return string `"Hello world!"` to R.
/// @export
#[extendr]
fn hello_world() -> &'static str {
    "Hello world!"
}
```

R

```r
#' Return string `"Hello world!"` to R.
#' @export
hello_world <- function() .Call(wrap__hello_world)
```

実行結果

```
devtools::load_all(".")

hello_world()
```

```
#> [1] "Hello world!"
```

例1) i32 (integer)を引数に取る関数

自動生成されたRの関数

Rust

```rust
/// @export
#[extendr]
fn add(x: i32, y: i32) -> i32 {
    x + y
}
```

R

```r
#' @export
add <- function(x, y) .Call(wrap__add, x, y)
```

実行結果

```
devtools::load_all(".")


# 引数の型は i32 だけど実数も渡せる
add(1, 2)
```

```
#> [1] 3
```

```
# 長さ1以上だとエラーになる
add(1:2, 2:3)
```

```
#> Error in add(1:2, 2:3) :
#>   Input must be of length 1. Vector of length >1 given.
```

例2) Vec<i32>を引数に取る関数

自動生成されたRの関数

Rust

```rust
#[extendr]
fn mult(x: Vec<i32>, y: i32) -> Vec<i32> {
    x
      .iter()
      .map(|n| n * y)
      .collect::<Vec<_>>()
}
```

R

```r
#' @export
mult <- function(x, y) .Call(wrap__mult, x, y)
```

実行結果

```
devtools::load_all(".")

mult(1:5, 10)
```

```
#> [1] 10 20 30 40 50
```

例3) struct

struct...?

- 環境としてエクスポートされるので状態を持つことができる
- たまに便利（正規表現のキャッシュを持たせている例: rr4r）

自動生成されたRの関数

Rust

```
struct Counter {
  i: i32,
}
```

```
/// @export
#[extendr]
impl Counter {
  fn new() -> Self {
    Self { i: 0 }
  }

  fn count(&mut self) -> i32 {
    self.i = self.i + 1;
    self.i
  }
}
```

自動生成されたRの関数

R

```r
#' @export
Counter <- new.env(parent = emptyenv())

Counter$new <- function() .Call(wrap__Counter__new)

Counter$count <- function() .Call(wrap__Counter__count, self)

#' @rdname Counter
#' @usage NULL
#' @export
`$.Counter` <- function (self, name) { func <- Counter[[name]]
```

実行結果

```
devtools::load_all(".")

cnt ← Counter$new()
cnt$count()
```

```
#> [1] 1
```

```
cnt$count()
```

```
#> [1] 2
```

その他こまごました話
（時間があれば）

その他

- 文字列関連はlifetimeを意識しないと使えないので初心者にはハードモード。数値計算系からはじめるのがおすすめ。
- Windowsのセットアップはやや面倒だけど、GitHub Actionsでビルド済みのバイナリを配布したりできるはず（調査中）
- CRANにはすでにextendrを使っているパッケージも存在する

まとめ

まとめ

- extendrを使うとマクロの魔術でRustの関数からRの関数を生成してくれる。
- 関数の引数の対応づけはRcppやcpp11と同じノリ。慣れている人はわりとすぐに使えるはず。
  - ちなみに、今回はすべてRustのデータ型に変換するタイプだったが、SEXPのまま扱うこともできる（ここはよく理解できていない）
- フィードバックお待ちしています！

r-wakalangにrustチャンネルをつくりました

References

- extendr: https://github.com/extendr/extendr
- extendrの口ゴはCC-BY-SA 4.0ライセンスで配布されています: https://github.com/extendr/artwork.