Rustr Book

updated on 2016-04-24

Contents

Chapter 1

Introduction

Rust and R Integration

`rustr` is a Rust library that provides a Rust API to work with R.

Write pure Rust code with `rustr`, and then use `rustinr` R package to generate Rust interfaces to R.

This project is now under construction. Issues and Pull requests are welcome!

Chapter 2

Setup

Rustr support Linux, Mac, and Windows.

2.1 Windows

2.1.1 Get R

Get R > 3.3.0 builded by GCC 4.9 and Rtools 3.3

2.1.2 Get Rust with GNU ABI

Checkout https://www.rust-lang.org/downloads.html

One of stable, beta, or nightly version of Rust is OK. You can put Rust installation in `PATH` or set `CARGO_HOME` environment variable to the path of `cargo.exe`.

If you want to build multi-arch R package, make sure you install both 64bit and 32bit Rust standard library, for example `C:\Rust\lib\rustlib\i686-pc-windows-gnu` and `C:\Rust\lib\rustlib\x86_64-pc-windows-gnu`

2.1.3 Get rustinr

Run This in R:

```
install.packages("devtools")
devtools::install_github("rustr/rustinr")
```

And we are ready to play!

Run this in R console.

```
library(rustinr)

rust(code ='
// #[rustr_export]
pub fn say_hi() -> String{
```

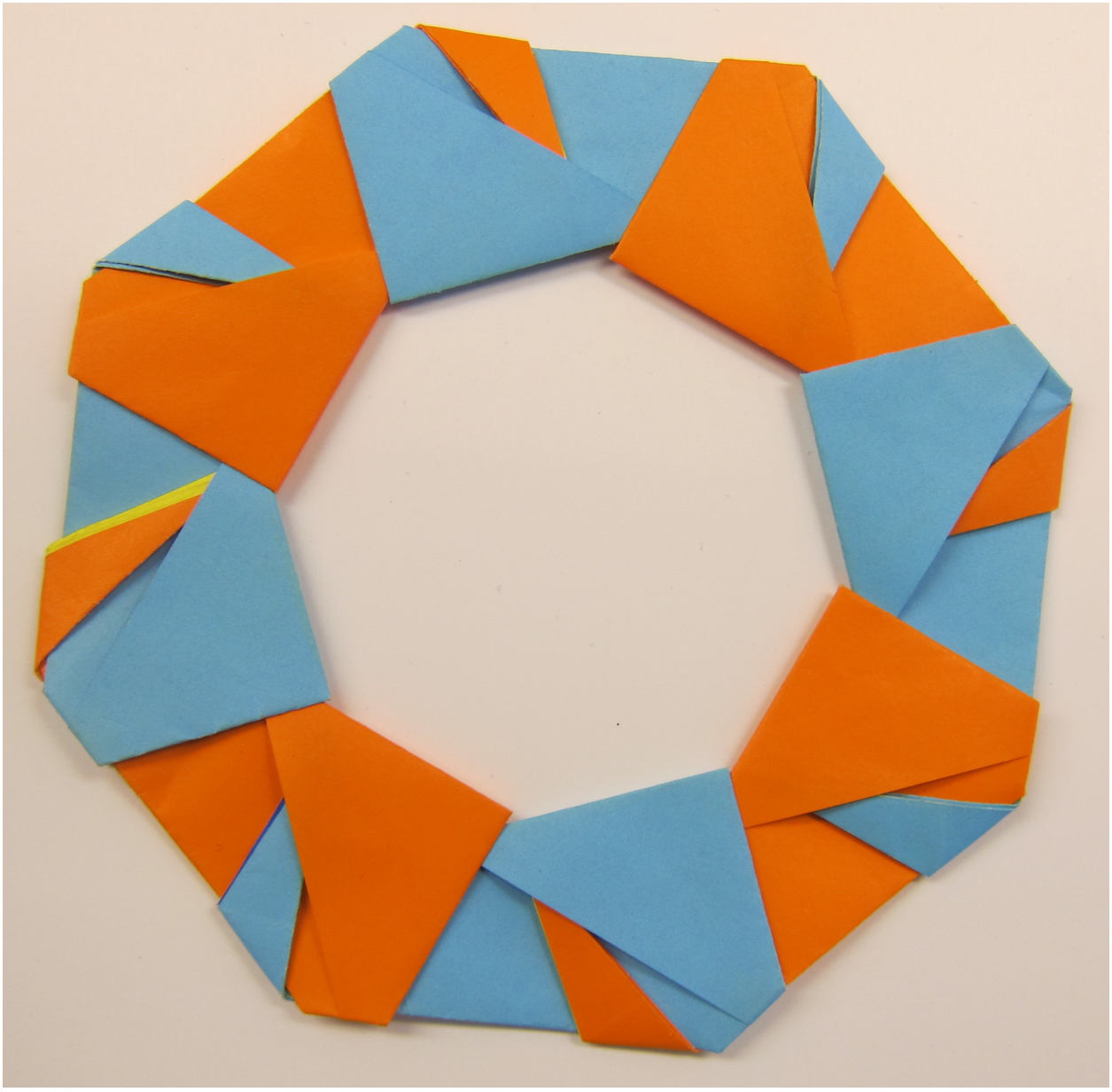

Figure 2.1: Fuse Ring

```
    "Hello World".into()
}
')

say_hi()
#> [1] "Hello World"
```

If some errors show up, run `check_rustr` to get more info:

```
check_rustr(detail = T)
```

2.2 Mac

2.2.1 Get R

Get R > 3.3.0.

2.2.2 Get Rust with GNU ABI

Checkout https://www.rust-lang.org/downloads.html

```
curl -sSf https://static.rust-lang.org/rustup.sh | sh
```

One of Stable, Beta, or Nightly version of Rust is OK.

You can put Rust installation in path or set `CARGO_HOME` environment variable to the path of `cargo`.

2.2.3 Get rustinr

Run This in R:

```
install.packages("devtools")
devtools::install_github("rustr/rustinr")
```

2.2.4 Get Xcode Command Line Tools

Open Terminal, and run `git` or `clang`, It may show a message to get you started.

Or you can check this great guide.

And we are ready to play!

Run this in R console.

```
library(rustinr)

rust(code ='
// #[rustr_export]
pub fn say_hi() -> String{
```

```
    "Hello World".into()
}
')

say_hi()
#> [1] "Hello World"
```

If some errors show up, run `check_rustr` to get more info:

```
check_rustr(detail = T)
```

2.3 Linux

2.3.1 Get R

Get R > 3.3.0.

2.3.2 Get Rust with GNU ABI

```
curl -sSf https://static.rust-lang.org/rustup.sh | sh
```

One of Stable, Beta, or Nightly version of Rust is OK.

You can put Rust installation in path or set `CARGO_HOME` environment variable to the path of `cargo`.

2.3.3 Get rustinr

Run This in R:

```
install.packages("devtools")
devtools::install_github("rustr/rustinr")
```

And we are ready to play!

Run this in R console.

```
library(rustinr)

rust('
// #[rustr_export]
pub fn say_hi() -> String{
    "Hello World".into()
}
')

say_hi()
#> [1] "Hello World"
```

If some errors show up, run `check_rustr` to get more info:

```
check_rustr(detail = T)
```

2.4 Playground with Docker

Visit https://play.rustr.org , and begin to code Rust in R.

This Web App is run in a Docker container. If you want to run this app locally. You can install Docker, and then run:

```
docker pull qinwf/shiny-rust-docker

docker run -p 3838:3838 qinwf/shiny-rust-docker
```

Visit http://127.0.0.1:3838 for your local version of rustr playground.

You can also run this Docker image with an R console that supports Rust:

```
docker run -ti --rm qinwf/shiny-rust-docker R
```

Chapter 3

Run

Now we will begin to code Rust in R.

Figure 3.1: Dive In

3.1 `rustinr` R package

`rustinr` is a R package to help user generate the basic struture of a Rust-R package and source Rust script in R. Install it with

```
devtools::install_github("rustr/rustinr")
```

3.1.1 Rust Function in R Console

`rust()` is a R function to create Rust function in R console interatively.

For a Rust function:

```
pub fn say_hi() -> String{
    "Hello World".into()
}
```

Just mark it with `// #[rustr_export]`, and put this function in `rust()`

```
library(rustinr)

rust(code = '
// #[rustr_export]
pub fn say_hi() -> String{
    "Hello World".into()
}
')

say_hi()
#> [1] "Hello World"
```

and then you can call `say_hi` in R.

```
say_hi
#> function ()
#> {
#>     .Call("ouNIkssJBKBM_say_hi", PACKAGE = "ouNIkssJBKBM")
#> }
```

Here is another example:

```
rust(code = '
// #[rustr_export]
pub fn fib_rs(x:u64)-> u64{
    if x == 0 { return 0 };
    if x == 1 { return 1 };
    return fib_rs(x - 1) + fib_rs(x - 2);
}')

fib_rs(10L)
#> [1] 55
```

For more example, you can also checkout https://gallery.rustr.org

3.2 Create a Rust-R Package

3.2.1 Init a Package

`rustr_init` will create an R package with Rust support.

Just run:

```
rustr_init("pkgname",".")
#> Creating directories ...
#> Creating DESCRIPTION ...
#> Creating NAMESPACE ...
#> Creating Read-and-delete-me ...
#> Saving functions and data ...
#> Making help files ...
#> Done.
#> Further steps are described in './pkgname/Read-and-delete-me'.
#> added useDynLib to NAMESPACE

list.dirs("pkgname")
#> [1] "pkgname"                "pkgname/man"
#> [3] "pkgname/R"              "pkgname/src"
#> [5] "pkgname/src/rustlib"    "pkgname/src/rustlib/src"
```

`pkgname/src/rustlib` is a Rust library, for more info see http://doc.crates.io/

Rust files are in `pkgname/src/rustlib/src`.

`pkgname/src/rustlib/src/lib.rs` should begin with:

```
# Run This In R
headr()
#> #[macro_use]
#> extern crate rustr;
#> pub mod export;
#> pub use rustr::*;
#>
#> // #[rustr_export]
#> pub fn say_hi()->RResult<String>{
#>     Ok("hello world".into())
#> }
```

or

```
#[macro_use]
extern crate rustr;
pub mod export;
pub use rustr::*;
```

3.2.2 Intro `rustrize()`

Rust function begin with `// #[rustr_export]` will be imported to R by `rustrize()` R function.

```rust
// /src/rustlib/src/lib.rs

#[macro_use]
extern crate rustr;
pub mod export;
pub use rustr::*;

// ' Roxygen Comment Header (Optional)
// '
// ' detail info about function
// ' @param a parameter detail
// ' @export
// #[rustr_export]
pub fn say_hi()-> String{
    "Hello World!".into()
}
```

and then run `rustrize()`.

3.2.2.1 Generated Files

These files are generated by `rustrize()`.

1. `/R/REXPORT.R` - `.Call()` binding in R.
2. `/src/REXPORT.c` - Exported `.Call()` Functions.
3. `/src/rustlib/src/export.rs` - A public module in /src/rustlib/src/lib.rs, this module export Rust function to R.

```r
# /R/REXPORT.R

#' Roxygen Comment Header (Optional)
#'
#' detail info about function
#' @param a parameter detail
#' @export
say_hi = function(){ .Call('SvhCRFYxjsuH_say_hi',PACKAGE = 'SvhCRFYxjsuH')}
```

```c
// /src/REXPORT.c

#include <Rinternals.h>
#include <R.h>

extern SEXP rustr_say_hi();
SEXP SvhCRFYxjsuH_say_hi(){ return(rustr_say_hi());}
```

```rust
// /src/rustlib/src/export.rs

use super::*;

#[no_mangle]
pub extern "C" fn rustr_say_hi()->SEXP{
```

```
 let res  = say_hi();

 let res_sexp : SEXP = unwrapr!(res.intor());

 return res_sexp;
}
```

And your R package are ready to `R CMD build`.

3.3 Summary

3.3.1 rust()

```
rust(path = "Rust file", code = "Rust code", depend = "Optional Cargo TOML dependency")
```

Source Rust file or code in R console.

3.3.2 rustr_init()

```
rustr_init(name = "R package name", path = "Pacakage path")
```

Init a Rust-R package struture. It is similar to `package.skeleton()`.

3.3.3 headr()

```
headr()
```

Give you the header of the `lib.rs` for Rust Library to get started with R.

3.3.4 rustrize()

```
rustrize(path = ".")
```

Generate R function for Rust function in a R package. It is similar to `compileAttributes()` in `Rcpp`.

Chapter 4

Rusty Error Handling

4.0.1 RResult

Do you want to `Throw Throw Throw`?? Sorry You can not!

But you can `try!()` `try!()` or `rtry!()`.

Make sure you learn enough Rust and read the Rust Book

This is `Monadic Error Handling`. If you do not know what it means, just forget it. It is not that kind of bad _ss thing.

If a Rust function may cause error, and need to throw a exception in R, you can return RResult as your function return type.

For example,

```
pub fn rinteger(x : SEXP)->RResult<usize>{
    let res :RResult<usize> = usize::rnew(x);
    res
}
```

Trait `RNew` will convert `SEXP` R Object Pointer to Rust type. It may fail, so it return `RResult<some_type>`.

```
rust(code = '
// #[rustr_export]
pub fn rinteger(x : SEXP)->RResult<usize>{
    let res :RResult<usize> = usize::rnew(x);
    res
}
')

rinteger(1)
#> Error in eval(expr, envir, enclos): NotCompatible: expecting a integer

rinteger(1L)
#> [1] 1

e = tryCatch(rinteger(1), error = function(e) e)
e
#> <NotCompatible in eval(expr, envir, enclos): NotCompatible: expecting a integer>
```

```
str(e)
#> List of 2
#>  $ message: chr "NotCompatible: expecting a integer"
#>  $ call   : language eval(expr, envir, enclos)
#>  - attr(*, "class")= chr [1:4] "NotCompatible" "RustError" "error" "condition"
```

Chapter 5

R Console Print in Rust

There are several ways to print to R console.

5.0.1 `r_printf()`

Print messages.

```
r_printf(&format!("{}, {}, {}", "1", 2, 3))
```

5.0.2 `r_message()`

Print messages, which can be suppressed.

```
r_message("bing!")
```

5.0.3 `r_warn()`

Print warnings.

```
r_warn("warning!")
```

5.0.4 `rprint()`

Print R Objects.

```
let res : NumVec = try!(NumVec::rnew(x))
rprint(res)
```

Chapter 6

R Type <-> Rust Type

Let's look at the code generated by rustrize().

6.0.1 Origin:

```
pub fn say_hi(a:String)-> String{
    a
}
```

6.0.2 Generated:

```
#[no_mangle]
pub extern "C" fn rustr_say_hi(a : SEXP)->SEXP{

 let a_ : String = unwrapr!( String::rnew(a) );
 let res  = say_hi(a_);

 let res_sexp : SEXP = unwrapr!(res.intor());

 return res_sexp;
}
```

`rustr_say_hi` takes a SEXP R pointer and then uses `String::rnew` to convert SEXP to Rust String.

6.0.3 RNew

`rnew` is from trait `RNew`:

```
pub trait RNew where Self: Sized
{
    fn rnew(x: SEXP) -> RResult<Self>;
}
```

`String` impl `RNew`, so we can call `String::rnew(SEXP)`.

```
impl RNew for String {
    fn rnew(x: SEXP) -> RResult<String> {
        .....
    }
}
```

And that is all. If you want to use you **own** awesome Rust type as input for R function. impl `RNew` for it.

6.0.4 IntoR

```
#[no_mangle]
pub extern "C" fn rustr_say_hi(a : SEXP)->SEXP{

 let a_ : String = unwrapr!( String::rnew(a) );
 let res  = say_hi(a_);

 let res_sexp : SEXP = unwrapr!(res.intor());

 return res_sexp;
}
```

From `say_hi(a_)` we get a `String`, and then we run `res.intor()`. `String` impl `IntoR`, so we can call `res.intor()`.

```
pub trait IntoR {
    fn intor(&self) -> RResult<SEXP>;
}
```

`RNew` - R type to a new Rust type

`IntoR` - Rust type to R type

Hope this is simple enough.

If you want to use you **own** awesome Rust type as **output**. impl `IntoR` for it.

6.0.5 Orphan Rule

There is an orphan rule to take care. Only your **own** Rust type can impl traits from `rustr` crates.

If you want to impl `RNew` for `Vec<String>`, you may not get lucky. But you can create a struct to wrap Vec. Zero-cost abstraction feature of Rust often helps you eliminate the overhead.

6.0.6 `RNew IntoR` impl Added to `rustr` Crate?

Create a pull request!

We will consider support some rust crates with feature tag. For example, `chrono` crate for time.

Chapter 7

Type Table

There are many types!

Figure 7.1: Bridge

7.1 R and Rust Type

7.1.1 R type

SEXP pointer in R has these types:

```
typedef enum {
    NILSXP  = 0,    /* nil = NULL */
    SYMSXP  = 1,    /* symbols */
    LISTSXP = 2,    /* lists of dotted pairs */
    CLOSXP  = 3,    /* closures */
    ENVSXP  = 4,    /* environments */
    PROMSXP = 5,    /* promises: [un]evaluated closure arguments */
    LANGSXP = 6,    /* language constructs (special lists) */
    SPECIALSXP  = 7,    /* special forms */
    BUILTINSXP  = 8,    /* builtin non-special forms */
    CHARSXP = 9,    /* "scalar" string type (internal only)*/
    LGLSXP  = 10,   /* logical vectors */
    INTSXP  = 13,   /* integer vectors */
    REALSXP = 14,   /* real variables */
    CPLXSXP = 15,   /* complex variables */
    STRSXP  = 16,   /* string vectors */
    DOTSXP  = 17,   /* dot-dot-dot object */
    ANYSXP  = 18,   /* make "any" args work */
    VECSXP  = 19,   /* generic vectors */
    EXPRSXP = 20,   /* expressions vectors */
    BCODESXP    = 21,   /* byte code */
    EXTPTRSXP   = 22,   /* external pointer */
    WEAKREFSXP  = 23,   /* weak reference */
    RAWSXP  = 24,   /* raw bytes */
    S4SXP   = 25,   /* S4 non-vector */

    NEWSXP      = 30,   /* fresh node creaed in new page */
    FREESXP     = 31,   /* node released by GC */

    FUNSXP  = 99    /* Closure or Builtin */
} SEXPTYPE;
```

From R source code, see r-source

7.1.2 Rust type

Rust standard container, such as `Vec` `LinkedList` are all supported. And there are some other types are supported with feature tags.

For example, DMat of `nalgebra` can be used with `ty_nalgebra` feature tag.

7.2 The Type Table

You can also see Rust Docs and csv files on GitHub

Version	Traits	From	To	Feature	Remove.Version
0.1.5	RNew URNew	LGLSXP	bool	default	NA
0.1.5	RNew URNew	INTSXP	u64	default	NA
0.1.5	RNew URNew	INTSXP	u32	default	NA
0.1.5	RNew URNew	INTSXP	u16	default	NA
0.1.5	RNew URNew	INTSXP	i64	default	NA
0.1.5	RNew URNew	INTSXP	i32	default	NA
0.1.5	RNew URNew	INTSXP	i16	default	NA
0.1.5	RNew URNew	INTSXP	i8	default	NA
0.1.5	RNew URNew	INTSXP	usize	default	NA
0.1.5	RNew URNew	INTSXP	isize	default	NA
0.1.5	RNew URNew	REALSXP	f64	default	NA
0.1.5	RNew URNew	REALSXP	f32	default	NA
0.1.5	RNew URNew	INTSXP RAWSXP	u8	default	NA
0.1.5	RNew URNew	INTSXP	Vec<u64>	default	NA
0.1.5	RNew URNew	INTSXP	Vec<u32>	default	NA
0.1.5	RNew URNew	INTSXP	Vec<u16>	default	NA
0.1.5	RNew URNew	INTSXP	Vec<i64>	default	NA
0.1.5	RNew URNew	INTSXP	Vec<i32>	default	NA
0.1.5	RNew URNew	INTSXP	Vec<i16>	default	NA
0.1.5	RNew URNew	INTSXP	Vec<i8>	default	NA

Chapter 8

R Object

More in Rust Docs.

Figure 8.1: Sweet

8.1 Vector

TODO

https://docs.rustr.org/rustr/vector/index.html

8.1.1 NumVec

https://docs.rustr.org/rustr/vector/numvec/struct.NumVecM.html

8.1.2 IntVec

https://docs.rustr.org/rustr/vector/intvec/struct.IntVecM.html

8.1.3 RawVec

https://docs.rustr.org/rustr/vector/rawvec/struct.RawVecM.html

8.1.4 BoolVec

https://docs.rustr.org/rustr/vector/boolvec/struct.BoolVecM.html

8.1.5 CplVec

https://docs.rustr.org/rustr/vector/cplvec/struct.CplVecM.html

8.2 VectorX - Special Vector

8.2.1 CharVec

https://docs.rustr.org/rustr/vectorx/charvec/struct.CharVecM.html

8.2.2 List

https://docs.rustr.org/rustr/vectorx/list/struct.RListM.html

8.2.3 ExprVec

https://docs.rustr.org/rustr/vectorx/exprvec/struct.ExprVecM.html

8.3 Rcomplex

https://docs.rustr.org/rustr/rdll/win64/struct.Struct_Unnamed9.html

8.4 Envir - Enviroment

https://docs.rustr.org/rustr/environment/struct.EnvirM.html

8.5 RFun - Function

https://docs.rustr.org/rustr/rfunction/struct.RFunM.html

8.6 RPtr - External Pointer

https://docs.rustr.org/rustr/rptr/struct.RPtrM.html

8.7 Fml - Formula

https://docs.rustr.org/rustr/formula/struct.RFmlM.html

8.8 RError - Error

https://docs.rustr.org/rustr/error/struct.RError.html

8.9 Symbol - Symbol

https://docs.rustr.org/rustr/symbol/struct.SymbolM.html

8.10 Promise - Promise

https://docs.rustr.org/rustr/promise/struct.PromiseM.html

8.11 S4 - S4

https://docs.rustr.org/rustr/s4/struct.S4M.html

8.12 Reference - RC

https://docs.rustr.org/rustr/reference/struct.ReferenceM.html

8.13 RObj - Object

https://docs.rustr.org/rustr/robject/struct.RObjM.html

8.14 RWeak - Weak Reference

https://docs.rustr.org/rustr/rweak/struct.RWeakM.html

8.15 RRand - Random Source

https://docs.rustr.org/rustr/feature/random/struct.RRand.html

8.16 RLang - Language

https://docs.rustr.org/rustr/rlang/struct.RLangM.html

Chapter 9

R Raw FFI with `unsafe` Rust

If you are interested in C API, you can read the docs about raw R-C API.
https://docs.rustr.org/rustr/rdll/index.html and https://docs.rustr.org/rustr/rdll/win64/index.html
These APIs are generated by rust-bindgen

Chapter 10

Cheat Sheet

Bibliography