Rust FFI Tour

2017/05/14 meetup

@王依依

我好菜啊

FFI

Foreign Function Interface

FFI HelloWorld

FFI HelloWorld

```c
/* hello.c */
/*
 * gcc -c -o hello.o hello.c
 * ar res libhello.a hello.o
 */

#include <stdio.h>

void say_hello(const char* p) {
    printf("Hello %s\n!", p);
}
```

```rust
// hello.rs

use std::os::raw::c_char;
use std::ffi::CString;

#[link(name="hello")]
extern {
    fn say_hello(p: *const c_char);
}

fn main() {
    let name = CString::new("Rust FFI").unwrap();
    unsafe {
        say_hello(name.as_ptr());
    }
}
```

Foreign Function Interface

- “inter-language calls”
    - Rust calls C
    - Rust calls C++
    - Rust calls Objective-C?
    - Rust calls Assembly?
    - Rust calls Python? Ruby? Java? …
- How about .* calls Rust?

ABI

Application Binary Interface

Application Binary Interface

```rust
pub enum Abi {
    // Single platform ABIs
    Cdecl,
    Stdcall,
    Fastcall,
    Vectorcall,
    Aapcs,
    Win64,
    SysV64,
    PtxKernel,
    Msp430Interrupt,
    X86Interrupt,

    // Multiplatform / generic ABIs
    Rust,
    C,
    System,
    RustIntrinsic,
    RustCall,
    PlatformIntrinsic,
    Unadjusted
}
```

```
extern {}
extern "C" {}
```

The Easiest Way

https://crates.io

Easy Way

- pointer conversion
  - & -> *const
  - &mut -> *mut
- repr(C), repr(u32), repr(packed)
- wrap raw pointer in Rust
  - write field access function
- use bindgen!

Easy Way(cont.)

- C++ demangle
    - e.g. #[link_name = "_ZNK7rocksdb7Options13DumpCFOptionsEPNS_6LoggerE"]
- unsafe std fn/type:
    - CString, CStr
    - Slice::from_raw_parts
    - as_ptr()
    - std::mem / std::ptr
        - is_null() / offset()
    - Cell: UnsafeCell

Easy Way(cont.)

- enum?
  - bitflags! Github: rust-lang-nursery/bitflags
- tagged union?

Union?

```
#[repr(C)]
pub struct __BindgenUnionField<T>(::std::marker::PhantomData<T>);
impl <T> __BindgenUnionField<T> {
    #[inline]
    pub fn new() -> Self { __BindgenUnionField(::std::marker::PhantomData) }
    #[inline]
    pub unsafe fn as_ref(&self) -> &T { ::std::mem::transmute(self) }
    #[inline]
    pub unsafe fn as_mut(&mut self) -> &mut T { ::std::mem::transmute(self) }
}
impl <T> ::std::default::Default for __BindgenUnionField<T> {
    #[inline]
    fn default() -> Self { Self::new() }
}
impl <T> ::std::clone::Clone for __BindgenUnionField<T> {
    #[inline]
    fn clone(&self) -> Self { Self::new() }
}
impl <T> ::std::marker::Copy for __BindgenUnionField<T> { }
```

Union?

```
#[repr(C)]
#[derive(Debug, Copy)]
pub struct wait {
    pub w_status: __BindgenUnionField<::std::os::raw::c_int>,
    pub w_T: __BindgenUnionField<wait__bindgen_ty_1>,
    pub w_S: __BindgenUnionField<wait__bindgen_ty_2>,
    pub bindgen_union_field: u32,
}
```

The Hacker Way

- Box<T> + into_raw()
- mem::forget()
- mem::zerod() / mem::uninitialized()
- mem::transmute()
  - mem::transmute_copy()
- Rep { ptr: *mut T, size: usize }

NonZero?

- core::nonzero::NonZero
- size_of::<Option<&T>>() == size_of::<&T>()
- mem::transmute() rocks! （误

FFI challenge?

- Memory management
  - Rust
    - Box<T> / Vec<T> / Rc<T> / Arc<T> …
    - Drop
  - C
    - void* malloc(size_t) / free(void*)
    - tcmalloc / jemalloc / glibc
  - C++
    - new / delete
    - unique_ptr<T> / shared_ptr<T>
    - Allocator

FFI challenge?

- Cross language reference
    - Callback! (Rust -> C -> Rust)
    - C++ inheritance (A C++ Class in Rust?)
- Performance
    - unnecessary memcpy
- Sync / Send / Copy
- Lifetime

Example - rust-sdl2

```rust
pub struct Surface {
    raw: *ll::SDL_Surface,
}
```

Example - rust-sdl2

```rust
pub struct Surface<'a> {
    raw: *ll::SDL_Surface,
    _marker: PhantomData<'a ()>,
}

// with sdl2::init() -> context<'a>
```

Example - rust-sdl2

```rust
pub struct SurfaceContext<'a> {
    raw: *mut ll::SDL_Surface,
    _marker: PhantomData<&'a ()>
}

impl<'a> Drop for SurfaceContext<'a> {}

pub struct Surface<'a> {
    context: Rc<SurfaceContext<'a>>,
}
```

Send / Sync

- unsafe impl
- refer doc!

Clone / Copy

- Rc<T> !
- do not copy raw pointers!

Drop

- Who allocates, who drops!
- lifetime to help!

missing part

- extern static variables (errno, version string)
- C++ class in Rust
    - Trait Objects + extern “C” fn
- Callbacks
    - Box<Fn> + extern “C” fn
    - Channel<T> for thread-safe
- Objective-C?
    - rust-objc (objc is C with runtime)
- swig / sip?
    - TODO
- write python modules in Rust?
    - #[no_mangle]
- JNI in rust?

gossip

- Rust patterns? - make a 3rd party lib rust-style.
    - Builder pattern
    - Chaining style
    - Vec/slice, String/str, CString/CStr, PathBuf/Path
    - Deref ?
    - Error / Result
    - From/Into
    - Vacuum pattern
    - move

linking & building

- #[link(name=…, kind=…)
- #[cfg] + target_os + target_arch
- #[no_mangle]
- build.rs

3rd party crates

- libc
    - A library for types and bindings to native C functions
- nix
    - friendly bindings to *nix APIs
- winapi
    - Types and constants for WinAPI bindings
- objc
    - Objective-C Runtime bindings and wrapper for Rust
- bindgen
    - Automatically generates Rust FFI bindings to C and C++ libraries.
- gcc
    - invoking the native C compiler to compile native C code into a static archive to be linked into Rust code.
- pkg_config
    - run the pkg-config system tool at build time in order to be used in Cargo build scripts
- cmake
    - running `cmake` to build a native library other rust-cpp
- other: google/rustcxx, cpp, c_vec, …

Thanks