Плагины на Rust для распределённой СУБД — технологический вызов

Мошкин Георгий

Георгий “Егор” Мошкин

- ВМК МГУ
- github.com/gmoshkin
- picodata.io

О чём доклад?

- Что такое плагины
- Как это работает на низком уровне
- Как с этим работать на rust
- Примеры

Что такое плагины?

Пользовательский код внутри вашей программы

- Расширение функционала
- Недостающие фичи
- Ускорение разработки
- Кастомизация

Кому нужны плагины?

- Media (photoshop, blender, …)
- Text editors & IDE (neovim, vs code, …)
- Gaming (modding)
- Web (java, flash, javascript)
- DBMS (postgres extensions, picodata plugins, …)
- Compilers (gcc, clang, rust)
- OS (drivers, eBPF)

Какие бывают плагины?

- Интерпретируемые
  - lua, python, lisp, …
- Байт-код виртуальные машины
  - wasm, jvm, .NET
- Нативные (компилируемые)
  - C/C++, rust, …
- Networking/IPC
  - LSP

Как сделать выбор?

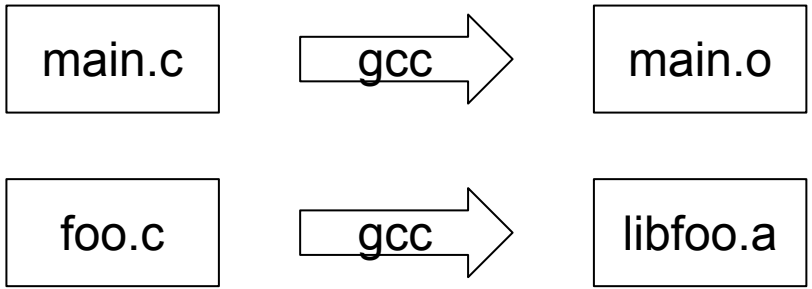
- Кто будет писать плагины?
- Какой язык программирования?
  - язык хост программы
  - язык плагинов
- Какие требования к производительности?

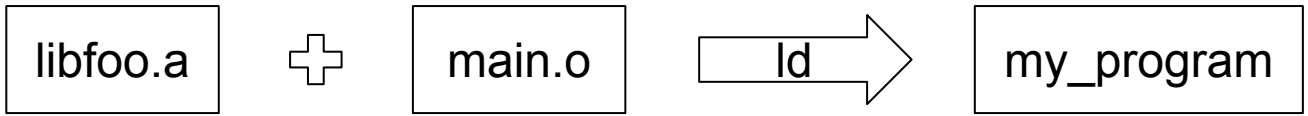
Нативные плагины

- dynamically loaded library (dll)
  - aka shared object (so)
- mmap(…, PROT_EXEC, …)

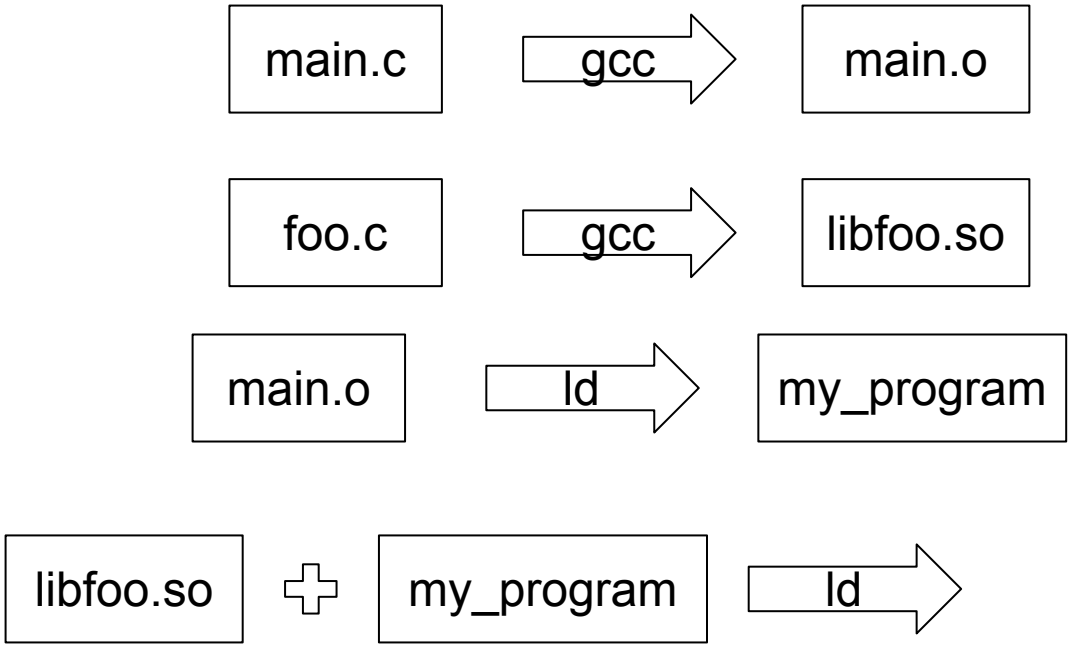
Что такое линкер?

Статическая линковка





Динамическая линковка



Статическая линковка

foo/src/main.rs

```
fn bar() → i32 {
    42
}

fn foo() → i32 {
    27 + bar()
}
```


```
006b20 <bar>:
    mov     $42,%eax
    ret

006b30 <foo>:
    push    %rax
    call    6b20 <bar>
    add     $27,%eax
    ret
```

Статическая линковка

bar/src/lib.rs

```
pub fn bar() → i32 {
    42
}
```

foo/src/main.rs

```
fn foo() → i32 {
    27 + bar::bar()
}
```


```
006b20 <bar>:
    mov     $42,%eax
    ret

006b30 <foo>:
    push    %rax
    call    *0×dffd1(%rip)
    add     $27,%eax
    ret
```

```
$ nm target/debug/foo | grep bar
0000000000006b20 T _ZN3bar3bar17h2a…
```

Динамическая линковка

bar/src/lib.rs

```
#[no_mangle]
pub extern “C”
fn bar() → i32 {
    42
}
```

foo/src/main.rs

```
fn foo() → i32 {
    27 +
    unsafe { bar() }
}
extern “C” {
    fn bar() → i32;
}
```


```
006b20 <bar>:
    mov     $42,%eax
    ret
```

```
006b30 <foo>:
    push    %rax
    call    *0×dffd1(%rip)
    add     $27,%eax
    ret
```

```
$ nm target/debug/foo | grep bar
                 U bar
```

ELF

- Executable and Linkable Format
- `nm`, `readelf`, `objdump`
- Плоское пространство имён

Name mangling

test.c

```
#include <stdio.h>

int my_func(int n) {
    return 27 + n;
}

int main() {
    int i = my_func(42);
    printf(“%d\n”, i);

    return 0;
}
```

```
$ gcc test.c -o test-c

$ nm test-c | grep my_func
0000000000001149 T my_func
```

Name mangling

test.cpp

```
#include <stdio.h>

int my_func(int n) {
    return 27 + n;
}

int main() {
    int i = my_func(42);
    printf(“%d\n”, i);

    return 0;
}
```

```
$ g++ test.cpp -o test-cpp

$ nm test-cpp | grep my_func
0000000000001149 T _Z7my_funci
```

Name mangling

test.cpp

```
int my_func(int n) {
    return 27 + n;
}

int my_func(char *c) {
    return strlen(c);
}
```

```
$ g++ test.cpp -o test-cpp

$ nm test-cpp | grep my_func
0000000000001169 T _Z7my_funci
000000000000117c T _Z7my_funcPc
```

Name mangling

test.rs

```
fn my_func(n: i32) → i32 {
    27 + n
}

fn main() {
    let i = my_func(42);
    println!(“{i}”);
}
```

```
$ rustc test.rs -o test-rust

$ nm test-rust | grep my_func
… _ZN4test7my_func17h39758bdd57c88daeE
```

Name mangling


```
$ rustc test.rs

                ⇒ _ZN4test7my_func17h39758bdd57c88daeE


$ rustc test.rs -C metadata=XXX

                ⇒ _ZN4test7my_func17he28f0d7cc46e3475E
```

Name mangling

- version sensitive
- unstable! v2
- dynamic linking?

Name mangling

bar/src/lib.rs

```
#[no_mangle]
pub extern “C” fn bar() → i32 {
    42
}
```

Name mangling

bar/src/lib.rs

```
#[export_name = “bar”]
pub extern “C” fn bar() → i32 {
    42
}
```

ABI

bar/src/lib.rs

```
#[no_mangle]
pub extern "C" fn bar() → i32 {
    42
}
```

ABI

Application Binary Interface

https://doc.rust-lang.org/reference/items/external-blocks.html#abi:

- `extern "Rust"` - default (unstable)
- `extern "C"` - same as C
- `extern "win64"` - Windows
- `extern "aapcs"` - ARM
- `extern "efiapi"` - UEFI
- …

ABI

test.rs

```
fn add(a:i32,b:i32)→i32 {
    a + b
}

fn main() {
    let i = add(13,37)-42;
    println!(“{i}”);
}
```

```
$ rustc test.rs
$ objdump -d test

013370 <add>:
    add    %esi,%edi
    mov    %edi,%eax
    ret

069420 <main>:
    mov    $0×d,%edi
    mov    $0×25,%esi
    call   013370 <add>
    sub    $0×2a,%eax
     ...
```

ABI

test.rs

```
fn add(a:i32,b:i32)→i32 {
    a + b
}

fn main() {
    let i = add(13,37)-42;
    println!(“{i}”);
}
```

```
> rustc.exe test.rs
> dumpbin.exe /disasm test.exe

add:
    add     ecx,edx
    mov     eax,ecx
    ret

main:
    mov     ecx,0×d
    mov     edx,0×25
    call    add
    sub     eax,0×2a
     ...
```

FFI

- Foreign function interface
- FFI = C-ABI = lingua franca
- extern “C”
- #[no_mangle]

Динамическая линковка

bar/src/lib.rs

```
#[no_mangle]
pub extern “C” fn bar() → i32 {
    42
}
```

foo/src/main.rs

```
fn foo() → i32 {
    27 + unsafe { bar() }
}
extern “C” {
    fn bar() → i32;
}
```

Пример: Динамическая линковка


```
$ tree dynamic-library-example
dynamic-library-example
├── application
│   ├── Cargo.toml
│   └── src
│       └── main.rs
├── Cargo.toml
└── library
    ├── Cargo.toml
    └── src
        └── lib.rs
```

Пример: Динамическая линковка


```
$ cat Cargo.toml
[workspace]
members =
["application","library"]
```

```
$ cat application/Cargo.toml
[package]
name = "application"
version = "0.1.0"
edition = "2021"
```

```
$ cat library/Cargo.toml
[package]
name = "library"
version = "0.1.0"
edition = "2021"

[lib]
crate-type = ["cdylib"]
```

Пример: Динамическая линковка


```
$ cat library/src/lib.rs
#[no_mangle]
pub extern "C" fn add(a: u64, b: u64) → u64 {
    a + b
}
```

```
$ cat application/src/main.rs
fn main() {
    let result = unsafe { add(1, 1) };
    println!("1 + 1 = {result}");
}

extern "C" {
    fn add(a: u64, b: u64) → u64;
}
```

Пример: Динамическая линковка


```
$ cargo build
 ...
error: linking with `cc` failed: exit status: 1
   |
   = note: /usr/bin/ld: target/d…a.rcgu.o: in function `application::main':
           …/application/src/main.rs:2: undefined reference to `add'
           collect2: error: ld returned 1 exit status


   …
   = note: use the `cargo:rustc-link-lib` directive to specify the native
libraries to link with Cargo (see
https://doc.rust-lang.org/cargo/reference/build-scripts.html#rustc-link-lib)
```

rustc-link-lib


```
$ tree dynamic-library-example
dynamic-library-example
├── application
│   ├── Cargo.toml
│   ├── build.rs
│   └── src
│       └── main.rs
└── ...
```

```
$ cat application/build.rs
fn main() {
    println!("cargo:rustc-link-lib=dylib=library");
}
```

rustc-link-lib


```
$ cargo build
 ...
error: linking with `cc` failed: exit status: 1
  |
  = note: /usr/bin/ld:
          cannot find -llibrary: No such file or directory
             collect2: error: ld returned 1 exit status
```

rustc-link-search


```
$ cat application/build.rs
fn main() {
    let out_dir = std::env::var("OUT_DIR").unwrap();
    let lib_dir = format!(“{out_dir}/../../..”);
    println!("cargo:rustc-link-search=native={lib_dir}");
    println!("cargo:rustc-link-lib=dylib=library");
}
```

```
$ cargo build --all
   Compiling application v0.1.0 (.../application)
   Compiling library v0.1.0 (.../library)
    Finished `dev` profile [debug] target(s) in 0.47s
```

```
$ ./target/debug/application
./target/debug/application:
    error while loading shared libraries: liblibrary.so:
    cannot open shared object file: No such file or directory


$ ldd ./target/debug/application
        linux-vdso.so.1 ( ... )
        liblibrary.so ⇒ not found
        libgcc_s.so.1 ⇒ /lib/…/libgcc_s.so.1 ( ... )
        libc.so.6 ⇒ /lib/x86_64-linux-gnu/libc.so.6 ( ... )
        /lib64/ld-linux-x86-64.so.2 ( ... )
```

LD_LIBRARY_PATH


```
$ LD_LIBRARY_PATH=./target/debug ./target/debug/application
1 + 1 = 2

$ LD_LIBRARY_PATH=./target/debug ldd ./target/debug/application
        ...
        liblibrary.so ⇒ ./target/debug/liblibrary.so ( ... )
        ...
```

-rpath


```
$ cat application/build.rs
fn main() {
    println!("cargo:rustc-link-arg=-Wl,-rpath=$ORIGIN");
    println!("cargo:rustc-link-lib=dylib=library");
}
```

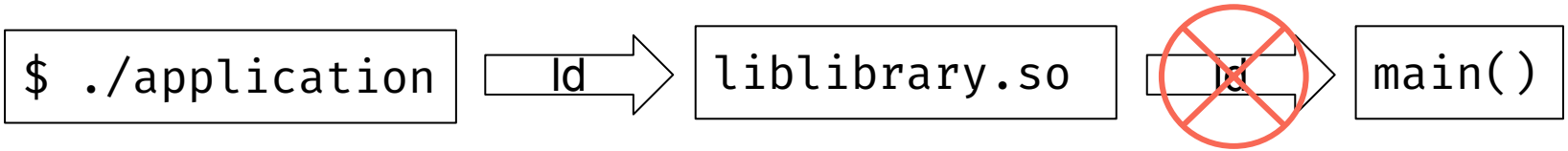
```
$ cargo build
$ objdump -x ./target/debug/application | grep RUNPATH
  RUNPATH                  $ORIGIN

$ ./target/debug/application
1 + 1 = 2

$ ldd ./target/debug/application | grep liblibrary
    liblibrary.so ⇒ /home/…/./target/debug/liblibrary.so
```

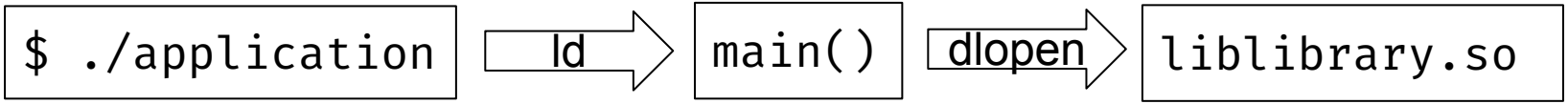
Динамическая линковка

- rustc-link-lib (-llibrary)
- rustc-link-search (-Lpath/to/library)
- LD_LIBRARY_PATH/-rpath
- неявные зависимости!



dlopen

- явный вызов
- выбор путей
- выбор символов



Пример: dlopen


```
$ tree plugin-example
plugin-example
├── application
│   ├── Cargo.toml
│   └── src
│       └── main.rs
├── Cargo.toml
└── plugin
    ├── Cargo.toml
    └── src
        └── lib.rs
```

Пример: dlopen


```
$ cat Cargo.toml
[workspace]
members = ["application","plugin"]
```

```
$ cat application/Cargo.toml
[package]
name = "application"
version = "0.1.0"
edition = "2021"

[dependencies]
libc = "*"
```

```
$ cat plugin/Cargo.toml
[package]
name = "plugin"
version = "0.1.0"
edition = "2021"

[lib]
crate-type = ["cdylib"]
```

Пример: dlopen


```
$ cat plugin/src/lib.rs
#[no_mangle]
pub extern "C" fn add(a: u64, b: u64) → u64 {
    a + b
}
```

Пример: dlopen

application/src/main.rs

```
fn main() {
    let module = unsafe { load_library("libplugin.so") };

    type fn_add_signature = extern "C" fn(u64, u64) → u64;
    let add: fn_add_signature = unsafe {
        load_symbol(module, "add")
    };

    let result = (add)(1, 1);
    println!("1 + 1 = {result}");
}
…
```

Пример: dlopen


```
unsafe fn load_library(name: &str) → *mut libc::c_void {
    let exe_path = std::env::current_exe().unwrap();
    let exe_dir = exe_path.parent().unwrap();

    let plugin_path = exe_dir.join(name);
    let plugin_path = plugin_path.to_str().unwrap();
    let plugin_path = CString::new(plugin_path).unwrap();

    let module = unsafe {
        libc::dlopen(plugin_path.as_ptr(),
                     libc::RTLD_LOCAL | libc::RTLD_NOW)
    };
    assert!(!module.is_null());
    module
}
```

Пример: dlopen


```
unsafe fn load_symbol<T>(
    module: *mut libc::c_void, name: &str
) → T {
    let name = std::ffi::CString::new(name).unwrap();

    let symbol: *mut libc::c_void
         = libc::dlsym(module, name.as_ptr());
    assert!(!symbol.is_null());

    std::mem::transmute_copy(&symbol)
}
```

Пример: dlopen


```
$ cargo build
    Finished `dev` profile [debug] target(s) in 0.07s

$ ldd ./target/debug/application | grep plugin

$ ./target/debug/application
1 + 1 = 2
```

Пример: dlopen


```
$ rm ./target/debug/libplugin.so

$ ./target/debug/application
thread 'main' panicked at application/src/main.rs:20:5:
assertion failed: !module.is_null()
stack backtrace:
    ...
   3: application::load_library
             at ./application/src/main.rs:20:5
   4: application::main
             at ./application/src/main.rs:2:18
    ...
```

Пример: dlopen

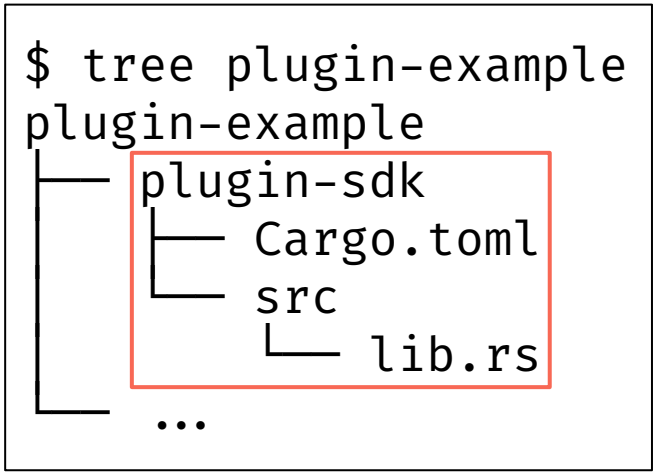
application/src/main.rs

```
fn main() {
    let module = unsafe { load_library("libplugin.so") };

    type fn_add_signature = extern "C" fn(u64, u64) → u64;
    let add: fn_add_signature = unsafe {
        load_symbol(module, "add")
    };

    let result = (add)(1, 1);
    println!("1 + 1 = {result}");
}
…
```

plugin-sdk



plugin-sdk


```
$ cat Cargo.toml
[workspace]
members = ["application","plugin",”plugin-sdk”]
```

```
$ cat application/Cargo.toml
 …
[dependencies]
plugin-sdk.path = “../plugin-sdk”
```

```
$ cat plugin/Cargo.toml
 …
[dependencies]
plugin-sdk.path = “../plugin-sdk”
```

plugin-sdk

plugin-sdk/src/lib.rs

```
pub type type_fn_add = extern "C" fn(a: u64, b: u64) → u64;
```

plugin/src/lib.rs

```
#[no_mangle]
pub extern "C" fn add(a: u64, b: u64) → u64 {…}

const TYPE_CHECK: plugin_sdk::type_fn_add = add;
```

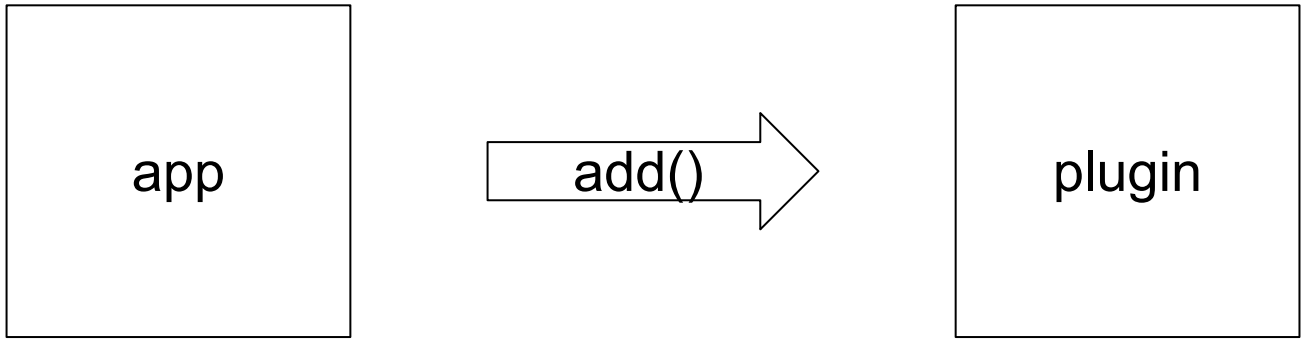
application/src/main.rs

```
fn main() {
    let f: plugin_sdk::type_fn_add = unsafe {
        load_symbol(m,"add")
    };
}
```

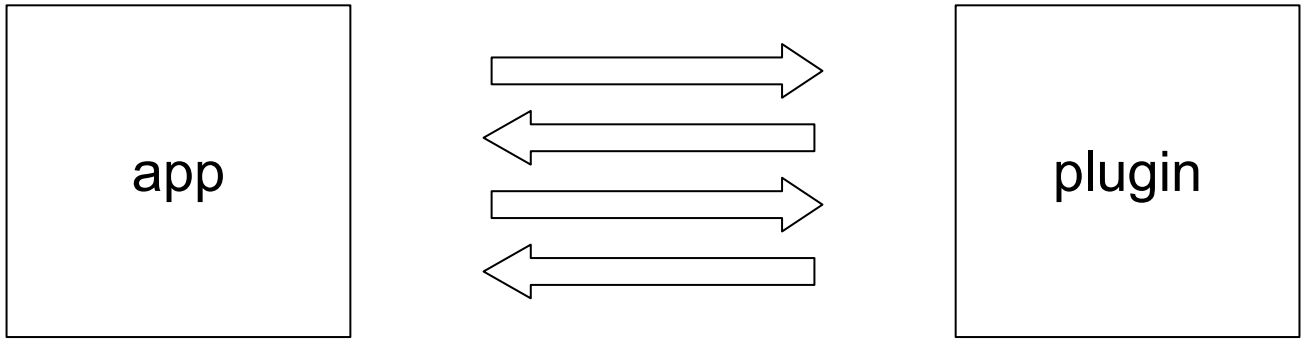
Safety

- macro
- crates.io/crates/libloading - тонкая обёртка над dlopen
- crates.io/crates/stabby - type-safe ABI
  - `#[stabby::export]`
  - `stabby::libloading::StabbyLibrary`

plugin-sdk



plugin-sdk



plugin-sdk

plugin-sdk/src/lib.rs

```
/// Plugin exports symbols with this signature
pub type plugin_fn = extern "C" fn( ... ) →  ... ;

/// Application exports symbols with this signature
extern “C” {
    fn host_fn( ... ) →  ... ;
}
```

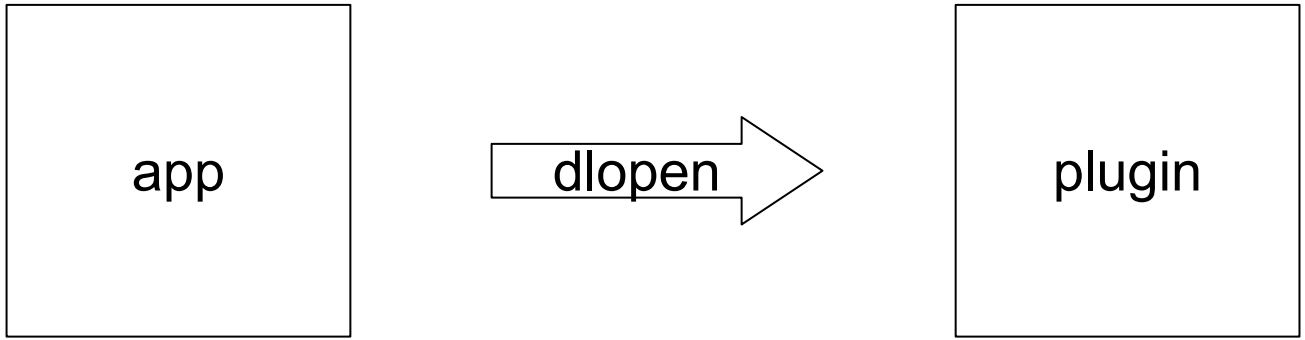
plugin-sdk

application/build.rs

```
fn main() {
    ...
    println!("cargo:rustc-link-arg=-rdynamic");
    ...
}
```

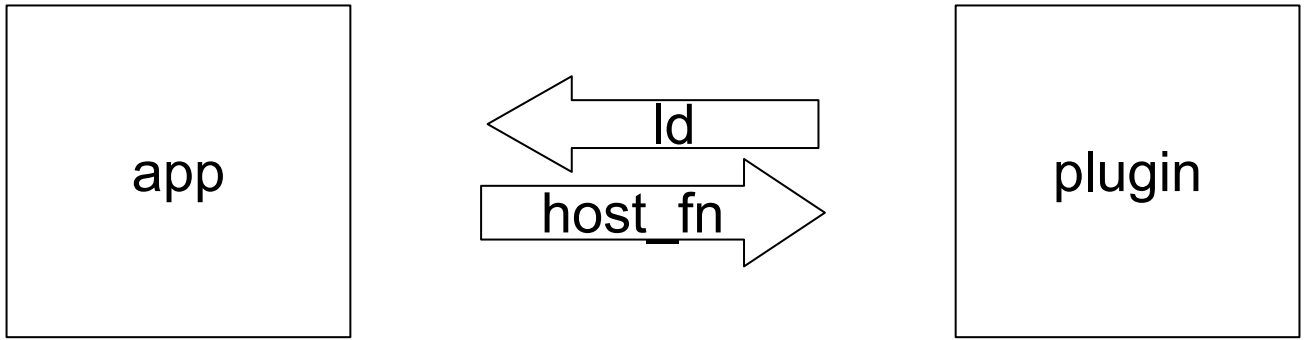
plugin-sdk


```
dlopen( ... ,  ...  | RTLD_NOW)
```



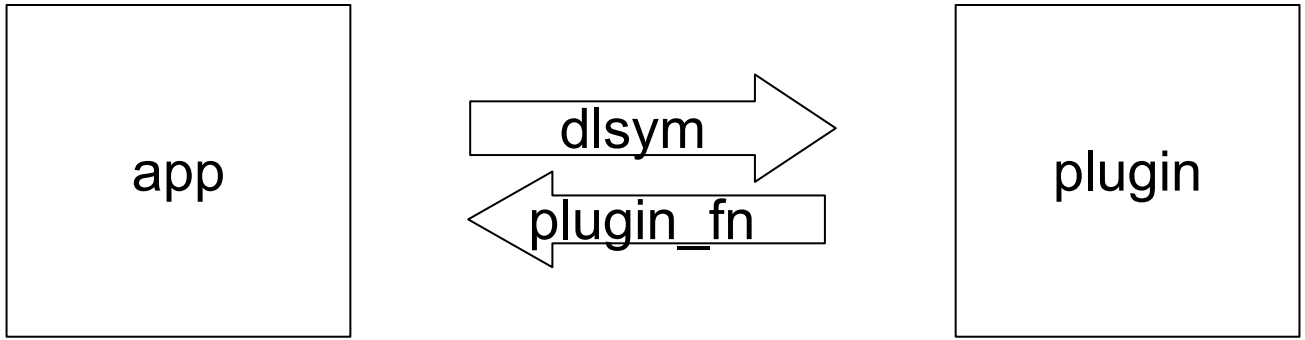
plugin-sdk


```
extern “C” {
    fn host_fn( … ) →  … ;
}
```



plugin-sdk


```
dlsym(m, “plugin_fn”)
```



FFI

- extern “C”
- #[no_mangle]
- crate-type = [“cdylib”]
- -rdynamic
- #[repr(C)]

repr(C)

test.c

```
struct my_struct {
    char a;
    long long b;
    char c;
};

int foo(struct my_struct *s) {
    return s->a + s->c;
}
```

```
<foo>:
    movsbl (%rdi),%eax
    movsbl 0x10(%rdi),%edx
    add    %edx,%eax
    ret
```

repr(C)

test.rs

```
struct MyStruct {
    a: u8,
    b: u64,
    c: u8,
};

fn foo(s: &MyStruct) → i32 {
    (s.a + s.c) as _
}
```

```
<foo>:
    movzbl 0×8(%rdi),%eax
    movzbl 0×9(%rdi),%edx
    add    %edx,%eax
    ret
```

repr(C)


```
#[repr(Rust)]
struct Rust {
    a: u8, b: i64, c: u8
}
```

size_of::<Rust>() == 16

b	b	b	b	b	b	b	b
a	c	-	-	-	-	-	-

```
#[repr(C)]
struct C {
    a: u8, b: i64, c: u8
}
```

size_of::<C>() == 24

a	-	-	-	-	-	-	-
b	b	b	b	b	b	b	b
c	-	-	-	-	-	-	-

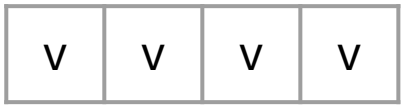
repr(C)


```
#[repr(Rust)]
enum Rust {
    A, B, C
}
```

```
size_of::<Rust>() == 1
```


```
#[repr(C)]
enum C {
    A, B, C
}
```

```
size_of::<C>() == 4
```



repr(C)


```
#[repr(Rust)]
enum Rust {
    A,
    B(u8),
}
```

size_of::<Rust>() == 2


```
#[repr(C)]
enum C {
    A,
    B(u8),
}
```

size_of::<C>() == 8

repr(C)


```
#[repr(Rust)]
enum Rust {
    A,
    B(std::num::NonZeroU8),
}
```

size_of::<Rust>() == 1


```
#[repr(C)]
enum C {
    A,
    B(std::num::NonZeroU8),
}
```

size_of::<C>() == 8

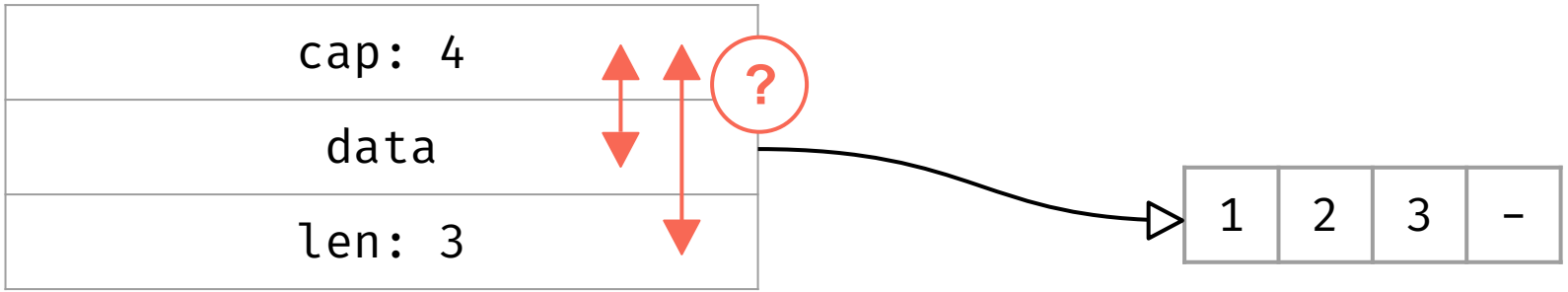
FFI

- repr(Rust) > repr(C)
- unstable!
- &str, Vec, HashMap, (_, _, _) — repr(Rust)

repr(C)


```
let bytes = vec![1,2,3];
```

```
struct Vec<u8> {
    cap: usize,
    data: *mut u8,
    len: usize,
}
```

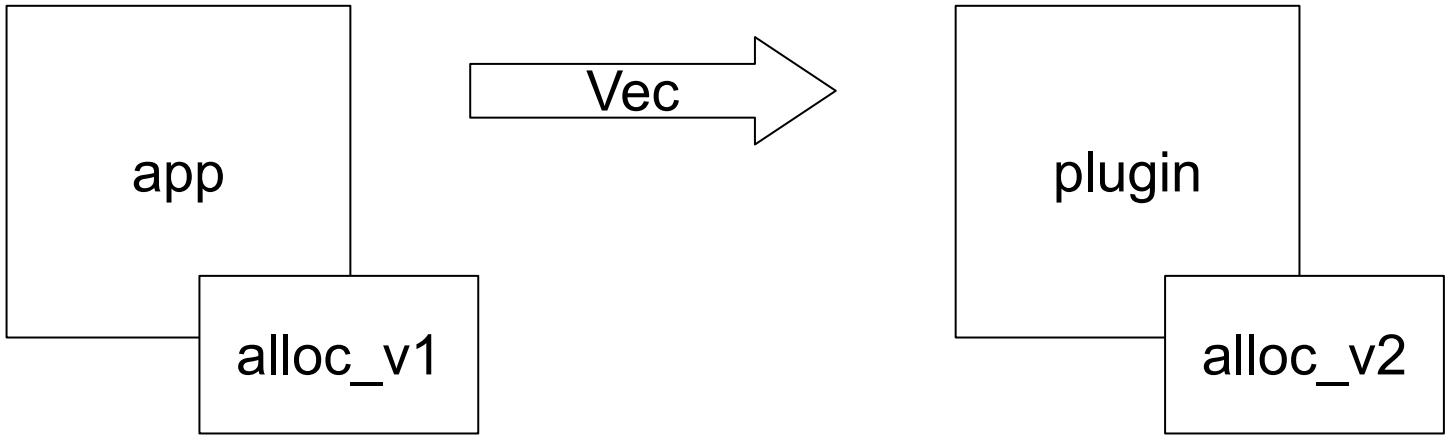


repr(C)

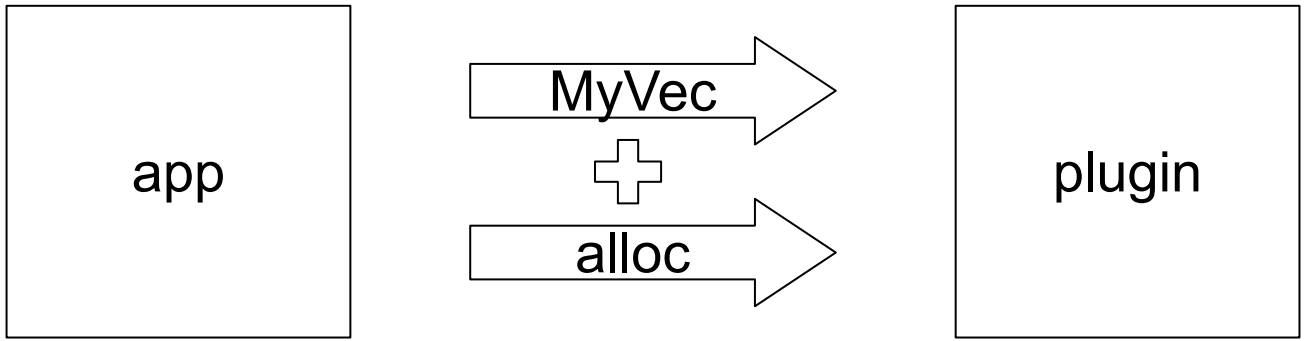

```
let bytes = Vec {
    data: std::alloc(layout),
    len: 3,
    cap: 3,
};
bytes[0] = 1;
bytes[1] = 2;
bytes[2] = 3;

// drop
std::dealloc(bytes.data, layout);
```

repr(C)



repr(C)



#[global_allocator]


```
struct DynGlobAlloc {};
impl GlobalAlloc for DynGlobAlloc {
    unsafe fn alloc( ... ) → *mut u8 {
        my_alloc( ... ) // extern “C”
    }
    unsafe fn dealloc( ... ) {
        my_dealloc( ... ); // extern “C”
    }
}
#[global_allocator]
static ALLOCATOR: DynGlobAlloc = DynGlobAlloc {};
```

arena allocation


```
extern “C” {
    fn temp_alloc(layout: Layout) → *mut u8;
    fn temp_alloc_mark() → usize;
    fn temp_alloc_truncate(mark: usize);
}

let mark = temp_alloc_mark();
let data_in_temp_allocator = plugin_function();
let data = copy_to_heap(data_in_temp_allocator);
temp_alloc_truncate(mark);
```

virtual table


```
#[repr(C)]
struct MyVec<T> {
    cap: usize,
    data: *mut u8,
    len: usize,
    vtable: MyVecVTable,
}

#[repr(C)]
struct MyVecVTable {
    alloc: extern “C” fn ( ... ),
    realloc: extern “C” fn ( ... ),
    dealloc: extern “C” fn ( ... ),
    drop_element: Option<extern “C” fn (*mut ())>,
}
```

virtual table


```
#[repr(C)]
struct MyVec<T> {
    cap: usize,
    data: *mut u8,
    len: usize,
    vtable: MyVecVTable,
}

#[repr(C)]
struct MyVecVTable {
    allocator_fn: extern “C” fn (
        old_data: *mut u8, old_len: usize, new_len: usize,
    ) → *mut u8,
    drop_element: Option<extern “C” fn (*mut ())>,
}
```

FFI

- repr(Rust) > repr(C)
- unstable!
- &str, Vec, HashMap, (_, _, _) — repr(Rust)
- crates.io/crates/stabby - enum optimizations + more
- `Fn(T) → R ???`

dyn Fn

plugin-sdk/src/lib.rs

```
pub fn register_command(
    name: &str,
    callback: impl Fn(&[&str]) → String,
);
```

plugin/src/lib.rs

```
let context = PluginStruct::new();
register_command(
    “foo”,
    move |args| {
        do_something_fun(args, &context)
    }
);
```

trampoline


```
#[repr(C)]
struct PluginCommandHandler {
    data: *mut (),

    callback: extern “C” fn(
        data: *const (),
        args: FfiSafeSlice<FfiSafeStr>,
    ) → FfiSafeString,

    drop_cb: extern “C” fn(data: *mut ()),
}
```

trampoline


```
pub fn register_command(name: &str, callback: F)
where
    F: Fn(&[&str]) → String,
{
    let closure = Box::new(callback);
    let handler = PluginCommandHandler {
        data: Box::into_raw(closure),
        callback: trampoline::<F>,
        drop: drop_cb::<F>(),
    };
    unsafe {
        ffi_register_command(FfiSafeStr::from(name),
                             handler);
    }
}
```

trampoline


```
extern “C” fn trampoline<F>(
    data: *const (),
    args: FfiSafeSlice<FfiSafeStr>,
) → FfiSafeString,
where
    F: Fn(&[&str]) → String,
{
    let args = args.to_vec_of_str();
    let closure = unsafe { &*(data as *const F) };
    let result = (closure)(&args);
    FfiSafeString::from(result)
}
```

trampoline


```
extern “C” fn drop<F>(data: *mut ()) {
    _ = unsafe {
        Box::from_raw(data as *mut F)
    }
}

impl Drop for PluginCommandHandler {
    fn drop(&mut self) {
        unsafe { self.drop_cb(self.data) }
    }
}
```

trampoline


```
pub fn handler_call(
    handler: &PluginCommandHandler,
    args: &[&str],
) → String
{
    let args = FfiSafeSlice::from_str_slice(args);
    let result = unsafe {
        (handler.callback)(handler.data, args)
    };
    result.into_string()
}
```

FFI

- внутри `repr(C)` и `extern “C”`
- динамическую память передаём через
  - vtable
  - кастомный аллокатор локальный/глобальный
- снаружи юзабельный и безопасный API
- crates.io/crates/stabby

О чём ещё стоит знать

- extern “C-unwind”, catch_unwind
- static constructors, .init_array, static_init
- дизайн API
- версионирование API
- wasm

Итог

- plugins in Rust = dynamic libraries
- extern “C”, repr(C), cdylib
- libloading/crabby