Dive into Rust Closure

董泽润

董泽润

十一年 IT 从业者

资深运维、DBA、后端工程师

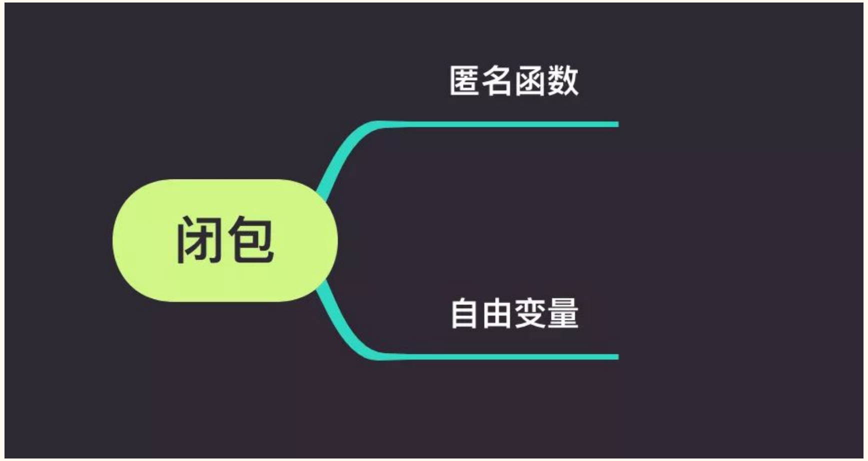
擅长 golang、redis、mysql 及 linux 底层

目前就职于某打车公司

公众号:董泽润的技术笔记

什么是闭包 Closure

In programming languages, a closure, also lexical closure or function closure, is a technique for implementing lexically scoped name binding in a language with first-class functions. Operationally, a closure is a record storing a function[a] together with an environment.[1] The environment is a mapping associating each free variable of the function (variables that are used locally, but defined in an enclosing scope) with the value or reference to which the name was bound when the closure was created.[b]



普通函数与嵌套函数

- 一等公民，可以做为参数、返回值和内置类型地位一样
- 不支持默认参数，用 Option 代替
- 不支持变参，可以用宏
- 不允许返回多值、用结构体或元组
- 嵌套函数隔离 namespace
- 嵌套函数不能像闭包一样引用外部变量

https://github.com/dongzerun/rust_closure/blob/main/function-and-nest.rs

https://github.com/dongzerun/rust_closure/blob/main/differ-fn-closure.rs

Rust 闭包

```
let f = |x: i32| -> i32 { x + 1 };

let f = |x: i32| x + 1; // 省略函数体

let f = |x| x+1; // 自动推导类型 1 可以推出 i32

let f = |x| x; // 类型也可以省略, 第一次使用时绑定, 或是后面有声明类型
```

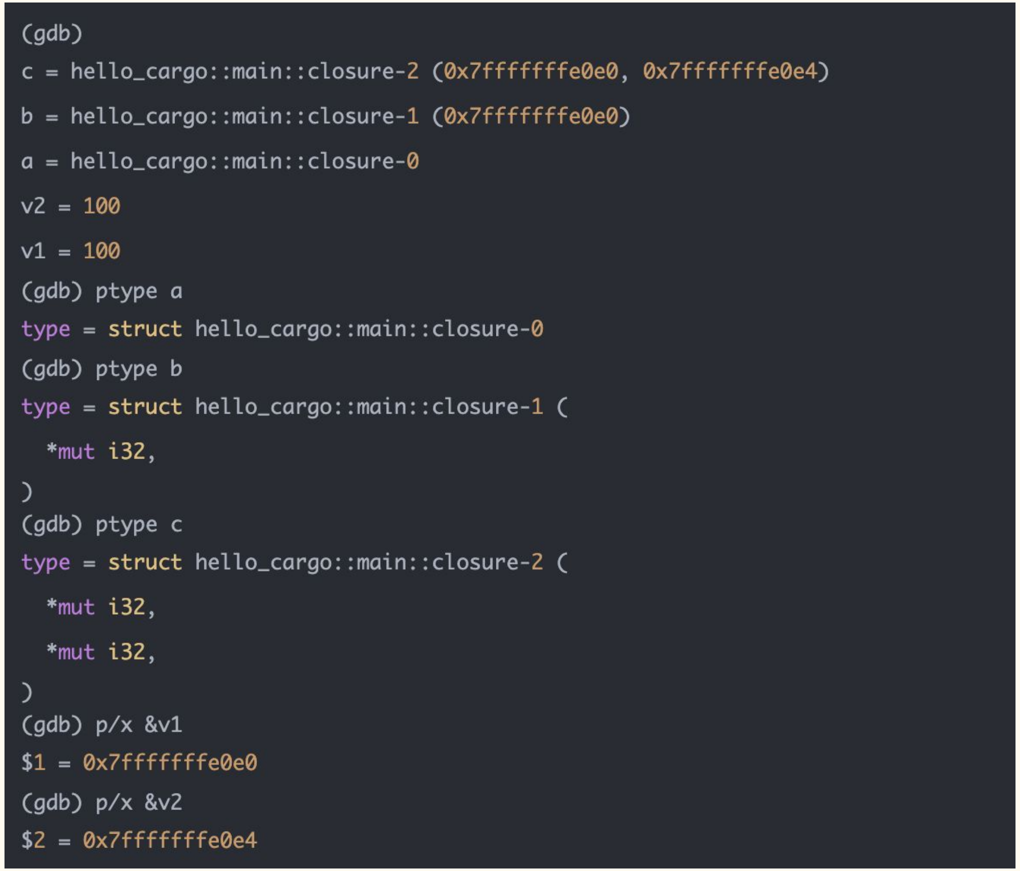
https://github.com/dongzerun/rust_closure/blob/main/closure_syntax.rs

Closure Struct

Closure 是一个结构体

函数指针 + 捕获变量

所有闭包名称唯一



```
(gdb)
c = hello_cargo::main::closure-2 (0x7fffffffe0e0, 0x7fffffffe0e4)
b = hello_cargo::main::closure-1 (0x7fffffffe0e0)
a = hello_cargo::main::closure-0
v2 = 100
v1 = 100
(gdb) ptype a
type = struct hello_cargo::main::closure-0
(gdb) ptype b
type = struct hello_cargo::main::closure-1 (
  *mut i32,
)
(gdb) ptype c
type = struct hello_cargo::main::closure-2 (
  *mut i32,
  *mut i32,
)
(gdb) p/x &v1
$1 = 0x7fffffffe0e0
(gdb) p/x &v2
$2 = 0x7fffffffe0e4
```

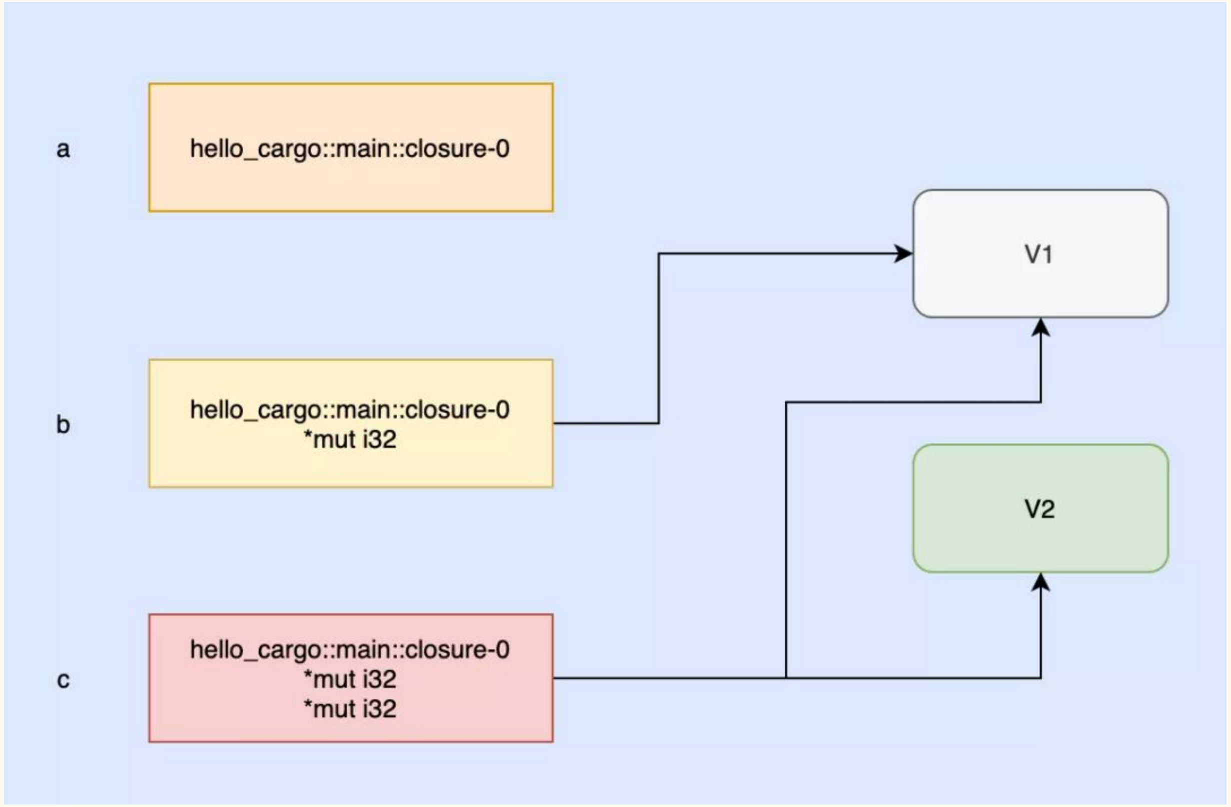
https://github.com/dongzerun/rust_closure/blob/main/closure_sizeof.rs

Closure Struct

Closure 是一个结构体

函数指针 + 捕获变量

所有闭包名称唯一


a
hello_cargo::main::closure-0
b
hello_cargo::main::closure-0
*mut i32
c
hello_cargo::main::closure-0
*mut i32
*mut i32
V1
V2

https://github.com/dongzerun/rust_closure/blob/main/closure_sizeof.rs

闭包和所有权

https://github.com/dongzerun/rust_closure/blob/main/closure_ownership-1.rs

https://github.com/dongzerun/rust_closure/blob/main/closure_ownership-2.rs

https://github.com/dongzerun/rust_closure/blob/main/closure_ownership-3.rs

https://github.com/dongzerun/rust_closure/blob/main/closure_ownership-4.rs

Disjoint capture in closures

对于闭包 || a.x + 1, 2018 的实现是捕获整个结构体 a, 但是现在只捕获所需要用的 x

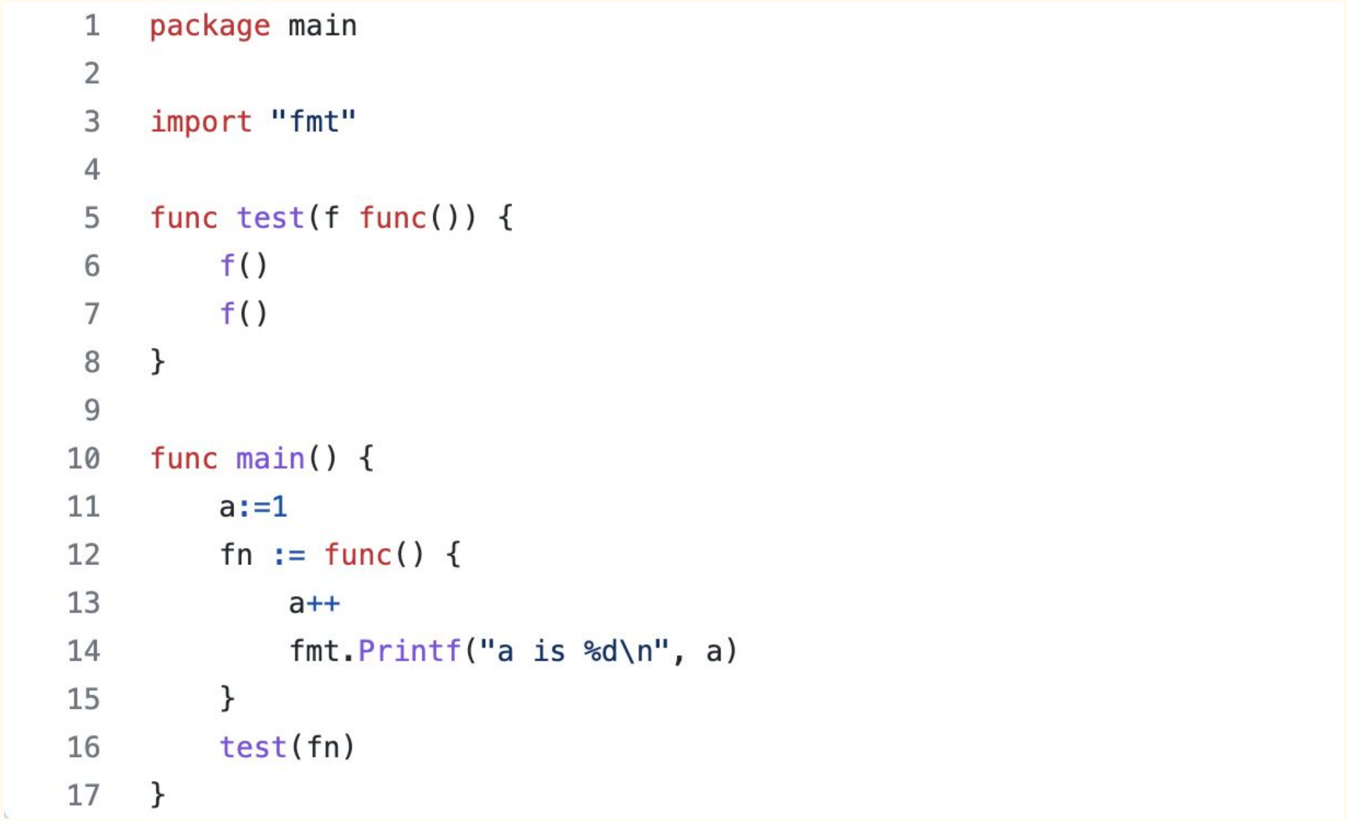
这个特性会导致一些对像在不同时间点被释放 dropped, 或是影响了闭包是否实现 Send 或 Clone trait, 所以 cargo 会插入语句 let _ = &a 引用完整结构体来修复这个问题。这个变动其实很大, 细节可以参考官方文档

https://doc.rust-lang.org/nightly/edition-guide/rust-2021/disjoint-capture-in-closures.html

Go Closure

换成 rust 就不行了

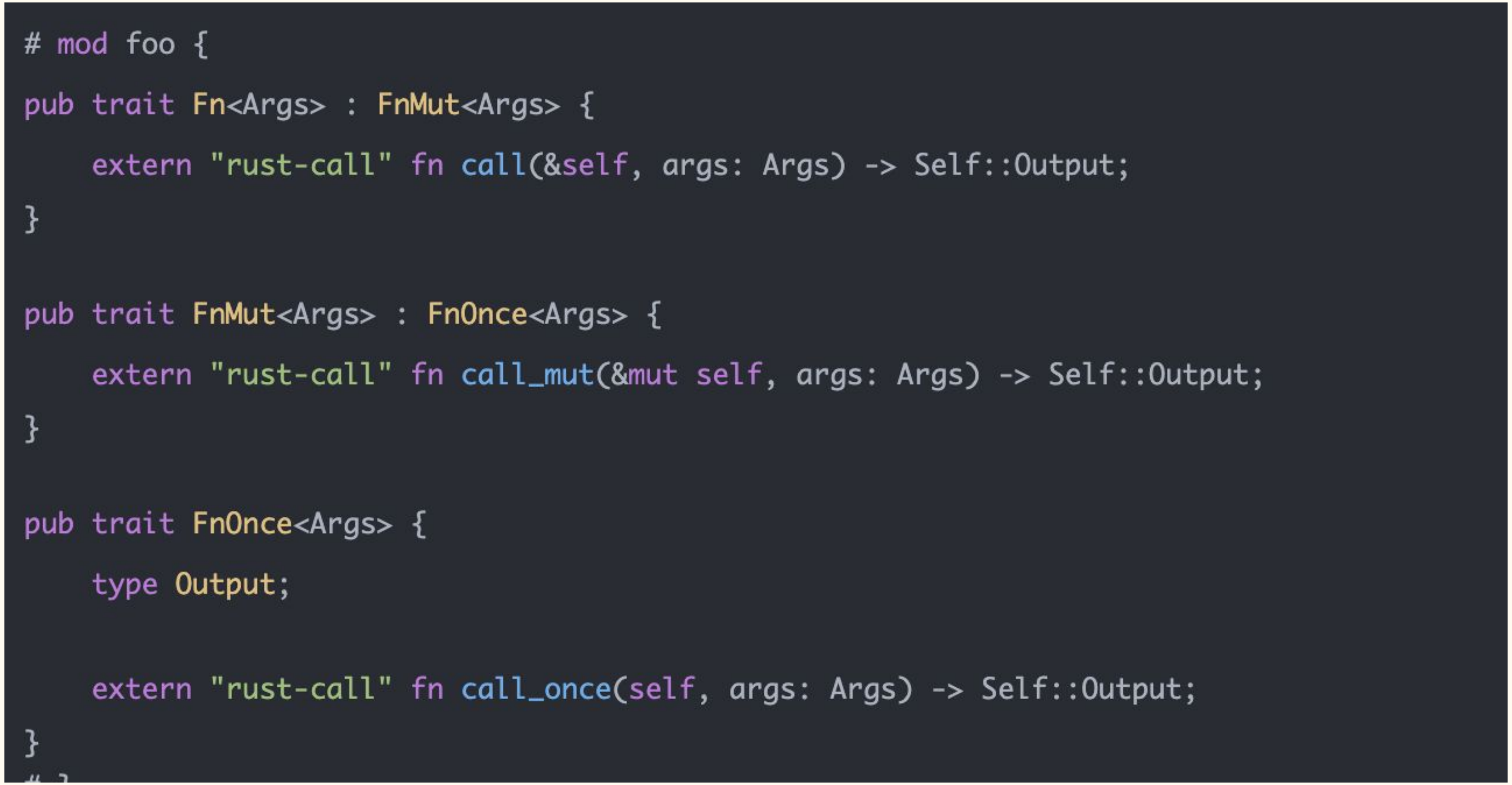
Rust 闭包是匿名的



```go
package main

import "fmt"

func test(f func()) {
	f()
	f()
}

func main() {
	a:=1
	fn := func() {
		a++
		fmt.Printf("a is %d\n", a)
	}
	test(fn)
}
```

https://github.com/dongzerun/rust_closure/blob/main/go-closure.go

Fn FnOnce FnMut



```rust
# mod foo {
pub trait Fn<Args> : FnMut<Args> {
    extern "rust-call" fn call(&self, args: Args) -> Self::Output;
}

pub trait FnMut<Args> : FnOnce<Args> {
    extern "rust-call" fn call_mut(&mut self, args: Args) -> Self::Output;
}

pub trait FnOnce<Args> {
    type Output;

    extern "rust-call" fn call_once(self, args: Args) -> Self::Output;
}
# }
```

Fn FnOnce FnMut

- FnOnce consumes the variables it captures from its enclosing scope, known as the closure’s environment. To consume the captured variables, the closure must take ownership of these variables and move them into the closure when it is defined. The Once part of the name represents the fact that the closure can’t take ownership of the same variables more than once, so it can be called only once.
- FnMut can change the environment because it mutably borrows values.
- Fn borrows values from the environment immutably.

Fn:表示捕获方式为通过引用(&T)的闭包

FnMut:表示捕获方式为通过可变引用(&mut T)的闭包

FnOnce:表示捕获方式为通过值(T)的闭包

捕获变量规则

- 通过引用:&T
- 通过可变引用:&mut T
- 通过值:T

Fn FnOnce FnMut

https://github.com/dongzerun/rust_closure/blob/main/closure_fn-1.rs

https://github.com/dongzerun/rust_closure/blob/main/closure_fn-2.rs

https://github.com/dongzerun/rust_closure/blob/main/closure_fn-3.rs

https://github.com/dongzerun/rust_closure/blob/main/closure_fn-4.rs

https://github.com/dongzerun/rust_closure/blob/main/closure_fn-5.rs

Fn FnOnce FnMut

本质上 Rust 为了内存安全，才引入这么麻烦的处理

平时写 go 程序，谁会在乎对象是何时释放，对象是否存在读写冲突呢？

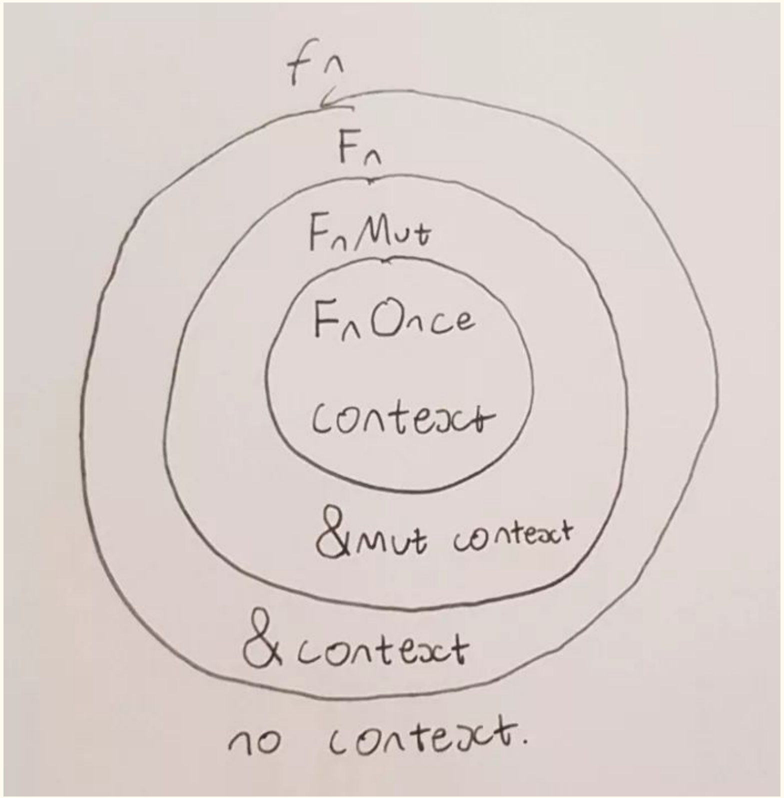
总得有人来做这个事情，Rust 选择在编译期做检查

Fn FnOnce FnMut

可以携带参数

Fn(i32) -> i32

FnMut() -> i32



https://github.com/dongzerun/rust_closure/blob/main/closure_fn-6.rs

闭包返回值

https://github.com/dongzerun/rust_closure/blob/main/return-closure-1.rs

https://github.com/dongzerun/rust_closure/blob/main/return-closure-2.rs

https://github.com/dongzerun/rust_closure/blob/main/return-closure-ownership.rs

参考文章

https://rustwiki.org/zh-CN/rust-by-example/fn.html

https://doc.rust-lang.org/book/ch19-05-advanced-functions-and-closures.html

Thanks Everyone !!!