Rust Community Stuttgart

Workshop

2019-05-15

Ownership, Borrowing, and Lifetimes

By Romeo Disca

RULES — OWNERSHIP & BORROWING

03.05.19

A reference has a copy trait.

A mutable reference doesn't have a copy trait

Stacks:

Arrows (refs) always go up. ⇧✓
Never down. ~~⇩~~ ↯

Heap:

Data on the heap never gets relocated.

1. All assignments are moves!
2. Return values are always moved.

temporary value / rhs value:

3. All assignments are first moved to the temporary memory. Then moved back to the stack

Stacks:

4. Stack values are always dropped in reverse order.

The dot operator:

- no ref → no magic → method call
- ref → dereferences "*" until no ref type found

RULES — LIFETIMES

03.05.19

(#1): fn f<'a, 'b>(x: &'a i32, y: &'b i32) {}
1 2 1 2

(#2): fn f<'a> (x: &'a i32) → &'a i32 {}

(#3): fn f<'ω> (&'self, x: &'_ i32, y: &'_ i32) → & T {...}

type in inference is based on the function signature.

ALL functions implement Fn, FnMut, FnOnce, Copy, Clone, Send, and Sync.

---

function item type

```
let f: fn()→() = fn(){}
```

An f variable.

function pointer type

```
fn f () {}
```

An f declaration.

closure trait

```
fn f<F:FnOnce()→()>(g:F){
f(|| {})
```

A closure.

RULES – LIFETIMES OF STRUCTS

03.05.19

```
struct A<'a, T> {
    x: &'a T,
}
```

older as

"x must be older as A"

```
impl<'a, T> A<'a, T> {
    fn xxx(&self) -> &'a T {
        self.x
    }
}
```

younger as

"output values are always younger as struct A"