Is there a way to have a Rust closure that moves only some variables into it?

Asked 2 years, 2 months ago

Active 1 year, 8 months ago

Viewed 4k times

16

3

I have a general `struct` with settings and an extra variable setting that I want to tune and play around with.

For all possible values in an integer range, I want to start a (scoped) thread with this variable set to that value. Depending on this value, they do slightly different work.

Each of these threads should be able to read the general settings struct.

```
use crossbeam; // 0.7.3

struct Settings {
    // ... many fields
}

const MAX_FEASIBLE_SCORE: u8 = 10;

fn example(settings: Settings) {
    crossbeam::scope(|scope| {
        for score in 0..MAX_FEASIBLE_SCORE {
            scope.spawn(|_| {
                let work_result = do_cool_computation(&settings, score);
                println!("{:?}", work_result);
            });
        }
    })
    .unwrap();
}

fn do_cool_computation(_: &Settings, _: u8) {}
```

This does not compile:

```
error[E0373]: closure may outlive the current function, but it borrows `score`, which is owned by the current function
  --> src/lib.rs:12:25
   |
10 |     crossbeam::scope(|scope| {
   |                       ----- has type `&crossbeam_utils::thread::Scope<'1>`
11 |         for score in 0..MAX_FEASIBLE_SCORE {
12 |             scope.spawn(|_| {
   |                         ^^^ may outlive borrowed value `score`
13 |                 let work_result = do_cool_computation(&settings, score);
   |                                                       ----- `score` is borrowed here
   |
note: function requires argument type to outlive `'1`
  --> src/lib.rs:12:13
   |
12 | /             scope.spawn(|_| {
13 | |                 let work_result = do_cool_computation(&settings, score);
14 | |                 println!("{:?}", work_result);
15 | |             });
   | |______________^
help: to force the closure to take ownership of `score` (and any other referenced variables), use the `move` keyword
   |
12 |             scope.spawn(move |_| {
   |                         ^^^^^^^
```

This would invalidate `&settings` since the first loop iteration will take ownership of `settings` in a `move` closure.

The only easy ways to make it work were:

- copy the `Settings` struct into each thread (which in my real application is rather expensive)
- introduce an `Arc` around `settings`, which also feels a bit unfortunate.

Is there a way that we can circumvent reference counting here? Is there a way we can move `score` into the inner closure while still being allowed to reference `settings`?

multithreading rust closures move-semantics lifetime

edited Apr 14 '20 at 15:50

Shepmaster

305k ● 59 ● 824 ● 1083

asked Oct 19 '19 at 0:52

Qqwy

4,906 ● 4 ● 33 ● 69

Add a comment

2 Answers

Active Oldest Votes

11

Yes, it is possible to move only one or some variables into a closure (rather than all or none).

Yes, this can be used to "circumvent" reference counting.

I found an answer in the documentation of rayon::scope that turns out to be exactly about this problem: 'Accessing the stack data [from within a scoped threads scope]'. That page also has an example that is clearer than the pseudocode in this question.

It turns out that you can either:

- Use a `move` closure but refer to variables in the outer scope by shadowing them with a reference, therefore capturing them by reference rather than by value, using `let settings = &settings;`:

```
crossbeam::scope(|scope| {
    let settings = &settings; // refer to outer variable by reference
    for score in 0..MAX_FEASIBLE_SCORE {
        scope.spawn(move |_| {
            let work_result = do_cool_computation(settings, score);
            println!("{:?}", work_result);
        });
    }
})
.unwrap();
```

- Use a normal closure, and only move the required variables by shadowing them inside the closure using `let score = score;`:

```
crossbeam::scope(|scope| {
    for score in 0..MAX_FEASIBLE_SCORE {
        scope.spawn(|_| {
            let score = score; // capture only score
            let work_result = do_cool_computation(&settings, score);
            println!("{:?}", work_result);
        });
    }
})
.unwrap();
```

Share
Improve this answer
Follow

edited Apr 14 '20 at 15:54

Shepmaster
305k ● 59 ● 824 ● 1083

answered Oct 19 '19 at 1:27

Qqwy
4,906 ● 4 ● 33 ● 69

2

Super minor nitpick here, but you're not really "circumventing" anything, you're just *not* reference counting.
– trent formerly cl
Oct 19 '19 at 2:09

@trentcl thanks! You are of course correct. 'Circumvention' is maybe a bit of sloppy language here. I have slightly altered the way it is written in the answer, but if you have better suggestions to make it more clear, feel free to propose one (and/or edit the answer yourself directly)
– Qqwy
Oct 19 '19 at 21:04

6

Your second answer does not work.
– Shepmaster
Apr 14 '20 at 15:54

Add a comment

5

The closure! macro gives the ability to selectively reference, move, or clone variables into a closure.

Example taken from the docs:

```
use closure::closure;

let string = "move".to_string();
let x = 10;
let mut y = 20;
let rc = Rc::new(5);

let closure = closure!(move string, ref x, ref mut y, clone rc, |arg: i32| {
    ...
});
```

Variables that are captured but not listed default to being moved.

Share
Improve this answer
Follow

answered Apr 27 '20 at 16:14

Tobu
23.5k ● 3 ● 88 ● 97

Add a comment

Your Answer

Post Your Answer

By clicking "Post Your Answer", you agree to our terms of service, privacy policy and cookie policy

Not the answer you're looking for? Browse other questions tagged multithreading rust closures move-semantics lifetime or ask your own question.

The Overflow Blog

- Sequencing your DNA with a USB dongle and open source code
- Don't push that button: Exploring the software that flies SpaceX rockets and...

Featured on Meta

- Providing a JavaScript API for userscripts
- Congratulations to the 59 sites that just left Beta

Linked

Does move on a closure copy the reference "pointer" or the actual object referenced?

How do I solve "cannot return value referencing local data" when using threads and async/await?

0
How can I use a field of a struct instance in an async move closure passed to a method of the instance?

Related

Cannot move out of borrowed content when borrowing a generic type

How do I return a reference to something inside a RefCell without breaking encapsulation?

What are move semantics in Rust?

How do I return a boxed closure from a method that has a reference to the struct?

Why does the closure take ownership of the vector here?

How to tell Rust to let me modify a shared variable hidden behind an RwLock?

Rust: allow multiple threads to modify an image (wrapper of a vector)?

Creating callback function with closure on a parameter, without "may outlive borrowed value" or "this closure implements `FnOnce`, not `Fn`"

0

Rust nested closure moves and multiple owners

Hot Network Questions

- What is this large long-legged orange and black insect?
- What does this entry on the Rocinante's pilot quick-menu mean?
- How to install a package via `apt-get` without flagging it as manually installed
- What does "Graecōs Argōs" in this sentence mean? (LLpsI)
- Bit Rot within LUKS Encryption

more hot questions

Question feed