Products

Changing a vector inside a closure gives "borrow of moved value" error

Log in Sign up

Ask Question

Asked 1 year, 11 months ago

Active 1 year, 11 months ago

Viewed 995 times

1

I am trying to change the elements of a vector inside a closure:

```
pub struct Foo<'a, T> {
    cb: Box<dyn FnMut(Vec<T>) + 'a>,
}

impl<'a, T> Foo<'a, T> {
    pub fn new<F: FnMut(Vec<T>) + 'a>(cb: F) -> Self {
        Self { cb: Box::new(cb) }
    }

    pub fn fun(&mut self, v: T) {
        let vector = vec![v];
        (self.cb)(vector);
    }
}

fn main() {
    let mut a = Vec::new();
    let mut foo = Foo::new(move |v| {
        for i in v {
            a.push(i);
        }
    });
    foo.fun(1);
    println!("{:?}", a);
}
```

Playground

I'm getting an error:

```
error[E0382]: borrow of moved value: `a`
  --> src/main.rs:24:22
   |
17 |     let mut a = Vec::new();
   |         ----- move occurs because `a` has type `std::vec::Vec<i32>`, which does not implement the `Copy` trait
18 |     let mut foo = Foo::new(move |v| {
   |                            -------- value moved into closure here
19 |         for i in v {
20 |             a.push(i);
   |             - variable moved due to use in closure
...
24 |     println!("{:?}", a);
   |                      ^ value borrowed here after move
```

I understand that Rust can't copy the value of `a` in the closure because `Vec` does not implement the trait `Copy`, so it has to move it, and moving `a` as mutable makes it unusable by `println!` later.

Am I storing the closure correctly? Is the use of the lifetime `'a` correct here? Should I wrap the vector in something like `Box` or `Cell`?

rust

Share
Improve this question
Follow

edited Jan 21 '20 at 14:58

Shepmaster
305k ● 59 ● 824 ● 1083

asked Jan 21 '20 at 14:56

Alex Covizzi
671 ● 7 ● 8

1

Why have you chosen to require ownership of the `Vec` in the closure argument (`FnMut(Vec<T>)`)?
– Shepmaster
Jan 21 '20 at 15:00

Your privacy

By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

1

You can solve your problem by removing the `move` and ensuring that the mutable borrow ends before the immutable borrow.
– Shepmaster
Jan 21 '20 at 15:04

Accept all cookies Customize settings

What if i need to use a before foo is dropped?
– Alex Covizzi
Jan 21 '20 at 15:09

1

Then it's a duplicate of [Can't borrow mutably within two different closures in the same scope]
– Shepmaster
Jan 21 '20 at 15:10

Add a comment

1 Answer

Active Oldest Votes

1

Here's the solution ([playground]):

```
pub struct Foo<'a, T> {
    cb: Box<dyn FnMut(Vec<T>) + 'a>,
}

impl<'a, T> Foo<'a, T> {

    pub fn new<F: FnMut(Vec<T>) + 'a>(cb: F) -> Self {
        Self {
            cb: Box::new(cb),
        }
    }

    pub fn fun(&mut self, v: T) {
        let vector = vec![v];
        (self.cb)(vector);
    }
}

fn main() {
    let mut a = Vec::new();

    // new scope to make sure that `foo` isn't alive when `a` is borrowed later
    {
        // no `move` to prevent moving `a` into the closure
        let mut foo = Foo::new(|v| {
            a = v.clone();
        });
        foo.fun(1);
    }
    println!("{:?}", a);
}
```

Share
Improve this answer
Follow
answered Jan 21 '20 at 19:22

Aloso
4,380 ● 4 ● 21 ● 36

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 rust 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

Can't borrow mutably within two different closures in the same scope

Related

Borrow errors for multiple borrows

Reference to unwrapped property fails: use of partially moved value: `self`

Why does the closure take ownership of the vector here?

How to compute arithmetic operations on a borrowed vector of elements that lack Copy in Rust?

1
the trait `_embedded_hal_digital_InputPin` is not implemented for `PE2<Output<OpenDrain>>`

Awaiting a Number of Futures Unknown at Compile Time

Declaring Associated Type of Trait Object in Async Function Parameter

Hot Network Questions

which one of these paths has the priority: /usr or /usr/local

What is this large long-legged orange and black insect?

Are they already planning a successor to the JWST?

How much of the English history in this Decameron story has any basis in fact?

Company kept previous personal phone number

more hot questions

Question feed