



Products

Rust - Lifetime of struct member depends on another struct member [duplicate]

[Log in](#) [Sign up](#)

[Ask Question](#)

Asked 2 years, 5 months ago

Active 2 years, 5 months ago

Viewed 1k times



1



This question already has answers here:

[Why can't I store a value and a reference to that value in the same struct?](#) (2 answers)

Closed 2 years ago.

I'm trying to write a Rust struct. The struct owns a Reference counted pointer to a string and also owns a vector of string slices to the same string.

Furthermore I'm trying to write a function to generate this struct. I'm unsure how to proceed.

```
struct MyStruct<a> {
    rc_string: Rc<String>,
    vec: Vec<&'a str>
}

fn build_my_struct<a>(s: &Rc<String>) -> MyStruct<a> {
    let rc_string = s.clone();
    let mut vec = Vec::new();
    vec.push(&rc_string[0..2]);

    MyStruct {
        rc_string: rc_string,
        vec: vec
    }
}
```

```
error[E0515]: cannot return value referencing local variable `rc_string`
--> src/main.rs:13:5
|
|   vec.push(&rc_string[0..2]);
|   ----- `rc_string` is borrowed here
12|
13| // MyStruct {
14| |   rc_string: rc_string,
15| |   vec: vec
16| | }
| |____^ returns a value referencing data owned by the current function
```

I understand that the `vec` variable has borrowed the `rc_string`. The compiler doesn't like returning `vec` because it has the borrow to the local variable `rc_string`.

However `rc_string` is being returned as well? The string slices are valid for the duration of the life of `MyStruct.rc_string`?

[rust](#) [lifetime](#)

Share

[Improve this question](#)

[Follow](#)

edited Jul 4 '19 at 18:10

asked Jul 4 '19 at 17:47



James Welchman

55 • 6

[Add a comment](#)

1 Answer

[Active](#) [Oldest](#) [Votes](#)



2



You need to borrow `Rc` for life time `'a` as well. Compiler needs to know that slice from a `String` is living in `'a` or not. In this case we need to borrow `Rc` for `'a` and compiler will know inner of `Rc` will also live in `'a`.

If you clone `s` and assign it to `rc_string`:

By clicking "Accept all cookies", you agree Stack Exchange can store cookies on your device and disclose information in accordance with our [Cookie Policy](#).

- ☒ `s` will stay in the function's scope as borrowed `Rc` for lifetime `'a`

- ☐ `rc_string` will be the owner of the `Rc` **pointer**

[Accept all cookies](#) [Customize settings](#)

and compiler won't be able to know slice of a `rc_string` is living for `'a` or not.

Using slice from a `s` will work :

```
fn build_my_struct<a>(s: &'a Rc<String>) -> MyStruct<'a> {  
    let mut vec = Vec::new();  
    let rc_string = s.clone();  
  
    vec.push(&s[0..2]);  
  
    MyStruct { rc_string, vec }  
}
```

[Playground](#)

[Share](#)

[Improve this answer](#)

[Follow](#)

edited Jul 4 '19 at 18:30

answered Jul 4 '19 at 18:09



[Ömer Erden](#)

5,861 ● 5 ● 25 ● 39

@JamesWelchman The code in your second question will work with the same way, please check [this code](#). You can't return slice from a moved content unless it lives in static lifetime.

– [Ömer Erden](#)

Jul 4 '19 at 18:38

In my original code I use the `Rc` pointer *because* the lifetime of `MyStruct` extends beyond the lifetime of the `Rc<String>` in the calling function. Just to be clear - I can't move *both* a `String` and a slice of the string into a return value?

– [James Welchman](#)

Jul 4 '19 at 18:50

Basically [this](#) is always impossible.

– [James Welchman](#)

Jul 4 '19 at 19:00

@JamesWelchman borrow checker's duty is preventing you from facing with dangling pointers, you can create a struct both `String` and it's slice but there is a possibility at runtime you can move(or even drop) `String` value from a field of struct without moving or dropping slices from that string. With this you can have dangling pointers

– [Ömer Erden](#)

Jul 4 '19 at 19:08

1

@JamesWelchman [this](#) is not possible because according to our definition `&s` needs to live in `my_struct`'s lifetime argument. It is a bit confusing because you are not dropping the real string but borrow checker is not able to know the string is trapped inside a `Rc` pointer.

– [Ömer Erden](#)

Jul 4 '19 at 19:15

[Add a comment](#)

Not the answer you're looking for? Browse other questions tagged [rust](#) [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

[Why can't I store a value and a reference to that value in the same struct?](#)

Related

[How do I transform &str to ~str in Rust?](#)

[Rust rustc::middle::graph::Graph with string node indices](#)

[Unable to coerce &String to &str](#)






[Encapsulating sequentially initialized state with self-references in Rust struct](#)

1
[What is the better way to wrap a FFI struct that owns or borrows data?](#)

[Problems with Tuple's lifetime in rust.](#)

[How to use delayed initialization in rust code and pass compiler "possibly-uninitialized variable" rule?](#)

Hot Network Questions

-  [Why are nerves blocked even though potassium channels are not blocked?](#)
-  [Alternatives to replace tandem single pole MWBC breakers?](#)
-  [Why is JWST parked in sunlight, rather than using a nuclear battery?](#)
-  [Question on connections in general relativity and particle physics](#)
-  [What's the largest REG_SZ value that Regedit can edit?](#)

[more hot questions](#)

STACK OVERFLOW

[Questions](#)
[Jobs](#)
[Developer Jobs Directory](#)
[Salary Calculator](#)
[Help](#)
[Mobile](#)

PRODUCTS

[Teams](#)
[Talent](#)
[Advertising](#)
[Enterprise](#)

COMPANY

[About](#)
[Press](#)
[Work Here](#)
[Legal](#)
[Privacy Policy](#)
[Terms of Service](#)
[Contact Us](#)
[Cookie Settings](#)
[Cookie Policy](#)

STACK EXCHANGE NETWORK

[Technology](#)
[Culture & recreation](#)
[Life & arts](#)
[Science](#)
[Professional](#)
[Business](#)
[API](#)
[Data](#)

[Blog](#)
[Facebook](#)
[Twitter](#)
[LinkedIn](#)
[Instagram](#)

site design / logo © 2021 Stack Exchange Inc; user contributions licensed under [cc by-sa](#). rev 2021.12.22.41046