Skip to main content

Stack Overflow

1. About
2. Products
3. For Teams

1. Stack Internal Implement a knowledge platform layer to power your enterprise and AI tools.
2. Stack Data Licensing Get access to top-class technical expertise with trusted & attributed content.
3. Stack Ads Connect your brand to the world’s most trusted technologist communities.
4. Releases Keep up-to-date on features we add to Stack Overflow and Stack Internal.
5. About the company Visit the blog

Loading…

current community

Stack Overflow
help chat
- Meta Stack Overflow

your communities

Sign up or log in to customize your list.

more stack exchange communities

company blog

3. Log in
4. Sign up

AI Assist is now on Stack Overflow. Start a chat to get instant answers from across the network. Sign up to save and share your chats.

Home
Questions
AI Assist
Tags
Challenges
7. Chat
Articles
Users
Companies
Collectives

Communities for your favorite technologies. Explore all Collectives

Stack Internal

Stack Overflow for Teams is now called **Stack Internal**. Bring the best of human thought and AI automation together at your work.

Try for free Learn more

Stack Internal

Bring the best of human thought and AI automation together at your work. Learn more

Should I add a lifetime for variable or pass it between functions Rust

Ask Question

Asked 3 years, 4 months ago

Modified 3 years, 4 months ago

Viewed 120 times

3

I have an LdapConn that I am passing around to multiple functions. Currently I'm passing the ldap variable to a function and then returning it. Inside of the function I'm not doing any dangerous modification of the ldapConn, I'm just changing the search result part. Passing it around works, but what's the best way to make a variable last the length of my program?

```
//main.rs
   let mut ldap: LdapConn = LdapConn::with_settings(
       LdapConnSettings::new()
           .set_no_tls_verify(true)
           .set_starttls(true),
       "ldaps://ldap.example.com:636",
   )
   .unwrap();

//other_file.rs
pub fn get_group_members(group: &str, mut conn: LdapConn) -> (LdapConn, Vec<String>) {
   let (s_filter, ou) = split_dn(group);
   let search_result = conn
       .search(
           &ou,
           Scope::Subtree,
           &format!("(&(objectClass=group)({}))", s_filter),
           vec!["member"],
       )
       .unwrap();
   let resp: Vec<
       std::collections::HashMap<std::string::String, std::vec::Vec<std::string::String>>,
   > = search_result
       .0
       .iter()
       .map(|x| SearchEntry::construct(x.clone()).attrs)
       .collect();

   (conn, trim_users(resp[0].get("member").unwrap().to_vec()))
}

//main.rs
   let (ldap, users) = get_group_members(group, ldap);
```

PS: LdapConn is not cloneable
https://docs.rs/ldap3/latest/ldap3/struct.LdapConn.html

> The API is virtually identical to the asynchronous one. The chief difference is that LdapConn is not cloneable: if you need another handle, you must open a new connection.

- rust

Share

Improve this question

Follow

edited Aug 2, 2022 at 15:45

Silvio Mayolo

70.9k66 gold badges9292 silver badges140140 bronze badges

asked Aug 2, 2022 at 15:35

Brandon Kauffman

1,93311 gold badge1717 silver badges4343 bronze badges

Add a comment |

1 Answer

Sorted by: Reset to default

Highest score (default) Trending (recent votes count more) Date modified (newest first) Date created (oldest first)

2

You should always take the least intrusive thing you can. That means: If you can take a `&`, you should do that. If not, you should take a `&mut` if you can. And only if you have no other option should you take ownership of a value. In your case, you're going to need unique access to the connection, so `&` is out of the question. `&mut`, however, is fair game.

```
pub fn get_group_members(group: &str, conn: &mut LdapConn) -> Vec<String> {
  ...
}

// main.rs
let mut ldap = ...;
let users = get_group_members(group, &mut ldap);
```

What you've stumbled upon (namely, that you can pass ownership and take it back) is called linear typing. It's the origin of Rust's type system, on which the borrow checker is built. There's nothing inherently *wrong* (mathematically) with always passing ownership and taking it back, except that it gets very tedious to constantly be accepting ownership back of things you just gave away. That's exactly why Rust allows borrowing, to prevent that sort of situation from getting out of hand.

Share

Improve this answer

Follow

answered Aug 2, 2022 at 15:41

Silvio Mayolo

70.9k66 gold badges9292 silver badges140140 bronze badges

Sign up to request clarification or add additional context in comments.

3 Comments

Add a comment

Brandon Kauffman

Brandon Kauffman Over a year ago

Awesome. This makes a lot of sense. I was previously try `fn(mut conn: &LdapConn)` which lead me down a wrong path. I now understand the difference of what I am borrowing.

2022-08-02T15:46:13.617Z+00:00

0

Reply

- Copy link

Silvio Mayolo

Silvio Mayolo Over a year ago

`mut conn: &LdapConn` is a mutable variable containing an immutable reference, so in that case you would be permitted to reassign the variable to point to something else, but you can't mutate what's inside of it. `conn: &mut LdapConn` is a mutable reference and allows you to deeply change what's inside.

2022-08-02T15:58:14.653Z+00:00

1

Reply

- Copy link

Sven Marnach

Sven Marnach Over a year ago

Sometimes you take ownership even if you don't have to to improve ergonomics, e.g. when using the builder pattern. For the case the OP asked about I can't see any advantage of passing ownership, though.

2022-08-02T19:37:28.147Z+00:00

1

Reply

- Copy link

Add a comment

Your Answer

Post as a guest

Name

Email

Required, but never shown

Related

3
How can I remember better when to use which lifetime syntax?

65

What is the correct way to use lifetimes with a struct in Rust?

26

How to initialize a variable with a lifetime?

13

When do I need to specify explicit lifetimes in Rust?

1

What is the advantage of using the same lifetime for multiple arguments?

1

Passing variables to function and lifetime

1

What do I follow Which function lifetime 'a or just scope lifetime?

5

Lifetime on a single argument function

1

Dealing with lifetimes in global variables

2

Lifetime relationship between function arguments and return value

Hot Network Questions

- Would it be plausible to have a flat eye with a moving pupil?
- Correct website for Thailand ETA application form
- Unity HDRP custom pass stencil buffer not working in build mode
- What does Felix mean by telling Bond to pick a contact point while "standing up" in Diamonds Are Forever?
- How to handle Expeditious Retreat with the optional chase rules
- What is the difference between an ultraviolet limit and ultraviolet stability?
- Swords and sandals movie where the ancient Greek hero fights in space
- Ping sweep of IP subnets
- Can a 10*10 square be paved with 1*4 rectangular stone plates?
- The state of icy worlds during differentiation and accretion, and how they survive with their water
- variable name change causes segfault
- Storm Sphere and Obstruction
- When should I use tied notes vs. the whole note duration?
- How does supporting "One China" work without also supporting Taiwan-China reunification?
- How many alien worlds could exist within 100 ly without us noticing?
- Is "We will review application starting from..." a deadline?
- Is there a website where we can find the average stipend/salary for the PhD and Postdoc for each country?
- Critiques of the real vs. nominal definition distinction
- Documentation for DUSHIN software
- Preservation of universal sentences under substructures in higher-order logic
- Is there a term for forming emotional or relational bonds with AI chatbots? If not, is “cyberpomorphic” a valid neologism?
- Best Welding Method to Fix a Damaged Steel Surly Frame
- Why does Mtt 13:34 say that Jesus always spoke in parables?
- TikZ word search diagram v2

more hot questions

Question feed

Subscribe to RSS

Question feed

To subscribe to this RSS feed, copy and paste this URL into your RSS reader.

Stack Overflow

- Questions
- Help
- Chat