Why is adding a lifetime to a trait with the plus operator (Iterator<Item = &Foo> + 'a) needed?

Asked 4 years, 10 months ago
Active 2 years, 3 months ago
Viewed 2k times

17

4

I'm applying a closure on the iterator and I want to use stable, so I want to return a boxed `Iterator`. The obvious way to do so is the following:

```
struct Foo;

fn into_iterator(myvec: &Vec<Foo>) -> Box<dyn Iterator<Item = &Foo>> {
    Box::new(myvec.iter())
}
```

This fails because the borrow checker cannot infer the appropriate lifetimes.

After some research, I've found [What is the correct way to return an Iterator (or any other trait)?](), which brought me to adding `+ 'a`:

```
fn into_iterator<'a>(myvec: &'a Vec<Foo>) -> Box<dyn Iterator<Item = &'a Foo> + 'a> {
    Box::new(myvec.iter())
}
```

But I don't understand

- What this does
- And why it is needed here

rust lifetime

Share
Improve this question
Follow

edited Sep 1 '19 at 21:11
Lukas Kalbertodt
61.1k ● 18 ● 189 ● 248

asked Feb 3 '17 at 16:13
torkleyy
1,019 ● 8 ● 24

Add a comment

1 Answer

Active Oldest Votes

25

✓

There is one thing that is easily overlooked: if you have a trait `Bar` and you want to have a boxed trait object `Box<dyn Bar>`, the compiler automatically adds a `'static` lifetime bound (as specified in [RFC 599]()). This means that `Box<dyn Bar>` and `Box<dyn Bar + 'static>` are equivalent!

In your case, the compiler automatically adds the static bound such that this ...

```
fn into_iterator(myvec: &Vec<Foo>) -> Box<dyn Iterator<Item = &Foo>>
```

... is equivalent to that:

```
fn into_iterator(myvec: &Vec<Foo>) -> Box<dyn Iterator<Item = &Foo> + 'static>
```

Now lifetime elision rules kick in and "connect" the two lifetime-slots, such that the above code is equivalent to:

```
fn into_iterator<'a>(myvec: &'a Vec<Foo>) -> Box<dyn Iterator<Item = &'a Foo> + 'static>
```

But the type `Iter<'a, Foo>` (the specific iterator type for `Vec<Foo>`) obviously does not satisfy the bound `'static` (because it is borrowing the `Vec<Foo>`)! So we have to tell the compiler that we don't want the default `'static` bound by specifying our own lifetime bound:

```
fn into_iterator<'a>(myvec: &'a Vec<Foo>) -> Box<dyn Iterator<Item = &'a Foo> + 'a>
```

Now the compiler knows that the trait object is only valid for the lifetime `'a`. Note that we don't explicitly need to annotate the lifetime of the associated `Item` type! Lifetime elision rules take care of that.

Share
Improve this answer
Follow
edited Sep 1 '19 at 21:10

answered Feb 3 '17 at 16:31

Lukas Kalbertodt
61.1k ● 18 ● 189 ● 248

Oh of course! I completely forgot that it isn't about Foo but the Iterator itself; I first thought the lifetime bound was on a struct... Thanks for clarifying this!
– torkleyy
Feb 3 '17 at 16:46

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 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

3
Lifetime of returned Boxed value does not live long enough

0
How can I satisfy the 'static lifetime requirement that the Rust compiler is requiring on an owned trait object?

0
Why am I getting a lifetime conflict in this code?

0
How to iterate over struct list with lifetime argument in Rust?

0
How to solve lifetime conflict in closure?

1
Rust fails to infer lifetimes for Box<dyn T<'a>> in some cases

1
Confused by lifetime of Box<Trait> in struct

0
Cannot create iterator and bind to a local variable

1
Returning a boxed iterator over a value in a structure

What is the correct way to return an Iterator (or any other trait)?

See more linked questions

Related

Cannot infer a lifetime for a closure returning a boxed trait that contains a reference

How to add lifetime argument to closure not returning a reference

lifetime with closure captures in rust

How can I return an impl Iterator that has multiple lifetimes?

2
Returning a type in Rust

How can I fix "cannot infer an appropriate lifetime for autoref" when implementing an iterator that returns mutable references?

Declaring Associated Type of Trait Object in Async Function Parameter

Lifetimes in lambda-based iterators

Hot Network Questions

Circuit analysis homework - LTspice results are different from calculations

'Cloth shop' and 'Clothes shop'

Tax implications of large gift

Best star for a Dyson sphere?

what is the official procedure if you answer "yes" to any of the US visa (DS 160) "are you a terrorist/slaver/bodysnatcher" questions?

more hot questions

Question feed

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