The compiler suggests I add a 'static lifetime because the parameter type may not live long enough, but I don't think that's what I want

Asked 5 years, 2 months ago

Active 3 years, 4 months ago

Viewed 16k times

58

12

I'm trying to implement something that looks like this minimal example:

```
trait Bar<T> {}

struct Foo<T> {
    data: Vec<Box<Bar<T>>>,
}

impl<T> Foo<T> {
    fn add<U: Bar<T>>(&mut self, x: U) {
        self.data.push(Box::new(x));
    }
}
```

Since Rust defaults to (as far as I can tell) pass-by-ownership, my mental model thinks this should work. The `add` method takes ownership of object `x` and is able to move this object into a `Box` because it knows the full type `U` (and not just trait `Bar<T>`). Once moved into a `Box`, the lifetime of the item inside the box should be tied to the actual lifetime of the box (e.g., when `pop()` ed off the vector the object will be destroyed).

Clearly, however, the compiler disagrees (and I'm sure knows a bit more than I...), asking me to consider adding a `'static` lifetime qualifier (E0310). I am 99% sure that's not what I want, but I'm not exactly sure what I'm supposed to do.

To clarify what I'm thinking and help identify misconceptions, my mental model, coming from a C++ background, is:

- `Box<T>` is essentially `std::unique_ptr<T>`
- Without any annotations, variables are passed by value if `Copy` and rvalue-reference otherwise
- With a reference annotation, `&` is roughly `const&` and `&mut` is roughly `&`
- The default lifetime is lexical scope

rust lifetime

Share
Improve this question
Follow

edited Jun 25 '18 at 17:01

Shepmaster
305k ● 59 ● 824 ● 1083

asked Oct 14 '16 at 23:51

Robert Mason
3,759 ● 3 ● 28 ● 42

Add a comment

2 Answers

Active Oldest Votes

73

✓

Check out the entire error:

```
error[E0310]: the parameter type `U` may not live long enough
 --> src/main.rs:9:24
  |
8 |     fn add<U: Bar<T>>(&mut self, x: U) {
  |            -- help: consider adding an explicit lifetime bound `U: 'static`...
9 |         self.data.push(Box::new(x));
  |                        ^^^^^^^^^^^
  |
note: ...so that the type `U` will meet its required lifetime bounds
 --> src/main.rs:9:24
  |
9 |         self.data.push(Box::new(x));
  |                        ^^^^^^^^^^^
  |
```

Specifically, the compiler is letting you know that it's possible that some arbitrary type `U` *might contain a reference*, and that reference could then become invalid:

```
impl<'a, T> Bar<T> for &'a str {}

fn main() {
    let mut foo = Foo { data: vec![] };

    {
        let s = "oh no".to_string();
        foo.add(s.as_ref());
    }
}
```

That would be Bad News.

Whether you want a `'static` lifetime or a parameterized lifetime is up to your needs. The `'static` lifetime is easier to use, but has more restrictions. Because of this, it's the default when you declare a *trait object* in a struct or a type alias:

```
struct Foo<T> {
    data: Vec<Box<dyn Bar<T>>>,
    // same as
    // data: Vec<Box<dyn Bar<T> + 'static>>,
}
```

However, when used as an argument, a trait object uses *lifetime elision* and gets a unique lifetime:

```
fn foo(&self, x: Box<dyn Bar<T>>)
// same as
// fn foo<'a, 'b>(&'a self, x: Box<dyn Bar<T> + 'b>)
```

These two things need to match up.

```
struct Foo<'a, T> {
    data: Vec<Box<dyn Bar<T> + 'a>>,
}

impl<'a, T> Foo<'a, T> {
    fn add<U>(&mut self, x: U)
    where
        U: Bar<T> + 'a,
    {
        self.data.push(Box::new(x));
    }
}
```

or

```
struct Foo<T> {
    data: Vec<Box<dyn Bar<T>>>,
}

impl<T> Foo<T> {
    fn add<U>(&mut self, x: U)
    where
        U: Bar<T> + 'static,
    {
        self.data.push(Box::new(x));
    }
}
```

Share
Improve this answer
Follow

edited Aug 3 '18 at 11:51

Boiethios
28.9k ● 10 ● 104 ● 147

answered Oct 15 '16 at 0:07

Shepmaster
305k ● 59 ● 824 ● 1083

5

I can't express enough how valuable this explanation is. Thank you so much!
– AlexLiesenfeld
Nov 11 '20 at 22:07

Add a comment

30

> asking me to consider adding a 'static lifetime qualifier (E0310). I am 99% sure that's not what I want, but I'm not exactly sure what I'm supposed to do.

Yes it is. The compiler does not want a `&'static` reference, it wants `U: 'static`.

Having `U: 'static` means that `U` contains no references with a lifetime less than `'static`. This is required because you want to put a `U` instance in a structure without lifetimes.

```
trait Bar<T> {}

struct Foo<T> {
    data: Vec<Box<dyn Bar<T>>>,
}

impl<T> Foo<T> {
    fn add<U: Bar<T> + 'static>(&mut self, x: U) {
        self.data.push(Box::new(x));
    }
}
```

Share
Improve this answer
Follow

edited Aug 3 '18 at 11:52

Boiethios
28.9k ● 10 ● 104 ● 147

answered Oct 15 '16 at 0:07

mcarton
22.7k ● 5 ● 63 ● 75

Add a comment

Your Answer

Post Your Answer

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

Visit chat

Linked

6
rust: borrowed value must be valid for the static lifetime

4
What does the 'static lifetime mean in a trait bound in a Rust future?

2
"Expected lifetime parameter" error in minimal futures callback example?

3
Lifetime of returned Boxed value does not live long enough

0
Rust: how to use lifetimes when there are no references?

1
Rust &str convertion to &'static &str

Why won't my closure capture an argument?

0
Why is a static lifetime needed when using move?

How to apply a lifetime to VecDeque<Box<dyn Trait>>?

0
Parameter T might not live long enough

See more linked questions

Related

Parameter type may not live long enough?

Parameter type may not live long enough (with threads)

Parameter type may not live long enough

Value does not live long enough with explicit lifetime, but does live long enough when omitted

Method not compatible with trait with confusing error message

"expected bound lifetime parameter" error when attempting to call a generic function

"The parameter type `C` may not live long enough", when it doesn't need to

The parameter type `T` may not live long enough

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

Hot Network Questions

How to increase white wine shelf life specifically bought for cooking?
`apt-mark showmanual` shows almost all packages. messed up?
What was the plutonium for, that was stolen at the start of The Amazing Spider-Man 2?
Does anyone have a DG Business BASIC manual?
Numbers, Racked Up
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