Rust lifetime mismatch - 'a does not necessarily outlive lifetime as defined here

Ask Question

Asked 2 years, 1 month ago

Modified 22 days ago

Viewed 212 times

3

I'm running into lifetime issues when trying to move borrowed data between structs. At a high level I'd like to do something like:

- read a yml file into a string buffer
- deserialize that into a struct using `from_str`
- create a different struct from that same borrowed data
- write that out to a file

Here's a sample implementation of what I built that gives the lifetime error:

Rust playground link

```
use serde::{Serialize, Deserialize};

#[derive(Deserialize, Debug, Serialize)]
pub struct Foo<'a> {
 name: &'a str
}

impl<'a> Foo<'a> {

   pub fn new(name: &'a str) -> Self {
       Self { name }
   }
}

#[derive(Deserialize, Debug, Serialize)]
pub struct Bar<'a> {
 name: &'a str
}

impl<'a> Bar<'a> {

   pub fn new(name: &'a str) -> Self {
       Self { name }
   }
}


pub trait Visitor {
   type Value;
   fn visit_table_borrowed<'a>(&mut self, t: &'a Foo<'a>) {
       let _ = t;
   }
}

pub trait DataBorrowed {
   fn accept<V: Visitor>(&self, visitor: &mut V);
}

impl<'a> DataBorrowed for Foo<'a> {
   fn accept<V: Visitor>(&self, visitor: &mut V) {
       visitor.visit_table_borrowed(self)
   }
}


impl<'a> Visitor for Bar<'a> {
   type Value = Bar<'a>;
   fn visit_table_borrowed(&mut self, t: &'a Foo<'a>) {
      self.name = t.name;
   }
}
```

```
impl<'a> From<Foo<'a>> for Bar<'a> {
   fn from(dt: Foo<'a>) -> Self {
       let mut table = Bar::new("bar");
       dt.accept(&mut table);
       table
   }
}

fn main() {
   let f = Foo::new("foo");
   let b: Bar = Bar::from(f);

}
```

The error:

```
error[E0308]: method not compatible with trait
 --> src/main.rs:49:5
  |
49 |     fn visit_table_borrowed(&mut self, t: &'a Foo<'a>) {
  |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ lifetime mismatch
  |
  = note: expected signature `fn(&mut Bar<'a>, &'a Foo<'a>)`
             found signature `fn(&mut Bar<'a>, &'a Foo<'a>)`
note: the lifetime `'a` as defined here...
 --> src/main.rs:49:5
  |
49 |     fn visit_table_borrowed(&mut self, t: &'a Foo<'a>) {
  |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
note: ...does not necessarily outlive the lifetime `'a` as defined here
 --> src/main.rs:47:6
  |
47 | impl<'a> Visitor for Bar<'a> {
  |      ^^

For more information about this error, try `rustc --explain E0308`.
```

- rust
- borrow-checker

Share

Improve this question

Follow

edited Nov 13 at 14:55

cafce25

30k55 gold badges4949 silver badges6868 bronze badges

asked Nov 3, 2023 at 23:01

seve

30522 silver badges1212 bronze badges

0

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)

3

The first problem with your code is that your trait definition and the actual implementation don't match in their signature, one method does have a lifetime parameter, the other doesn't, this is what the compiler complains about.
To fix this you can just double check that both methods have the same signature (I personally always copy & paste the signature over from the trait's definition or let the LSP server do that for me).

Then there is the problem that `&'a Foo<'a>` really is a code smell more often than not, it's virtually never the actual lifetime and ok in only a few more cases. Unless you know what you're doing you should always start with a different lifetime per lifetime hole, that way the compiler can tell you exactly what it thinks how those lifetimes relate.

You don't actually want your borrows to be limited by the reference to the `Foo`, after all you want to move the reference from it and use that longer lifetime, but that's impossible if you artificially force `'a` and `'b` of `&'a Foo<'b>` to be the same by reusing `'a` for the inner lifetime.
To fix this replace `&'a Foo<'a>` with `&Foo<'a>` everywhere (the outer lifetime can be elided in all places you used it)

The third problem is that your trait definitions don't allow for your requirements, you can't have a function be generic over a lifetime and at the same time fix that same lifetime to some lifetime from the implementation.
To fix it change `Visitor` and `DataBorrowed` to accept a lifetime parameter:

```
pub trait Visitor<'a> {
   type Value;
   fn visit_table_borrowed(&mut self, t: &Foo<'a>) {
       let _ = t;
   }
}

pub trait DataBorrowed<'a> {
   fn accept<V: Visitor<'a>>(&self, visitor: &mut V);
}
```

Here is a complete version with all three problems fixed:

```
use serde::{Deserialize, Serialize};

#[derive(Deserialize, Debug, Serialize)]
pub struct Foo<'a> {
   name: &'a str,
}

impl<'a> Foo<'a> {
   pub fn new(name: &'a str) -> Self {
       Self { name }
   }
}

#[derive(Deserialize, Debug, Serialize)]
pub struct Bar<'a> {
   name: &'a str,
}

impl<'a> Bar<'a> {
   pub fn new(name: &'a str) -> Self {
       Self { name }
   }
}

pub trait Visitor<'a> {
   type Value;
   fn visit_table_borrowed(&mut self, t: &Foo<'a>) {
       let _ = t;
   }
}

pub trait DataBorrowed<'a> {
   fn accept<V: Visitor<'a>>(&self, visitor: &mut V);
}

impl<'a> DataBorrowed<'a> for Foo<'a> {
   fn accept<V: Visitor<'a>>(&self, visitor: &mut V) {
       visitor.visit_table_borrowed(self)
   }
}

impl<'a> Visitor<'a> for Bar<'a> {
   type Value = Bar<'a>;
   fn visit_table_borrowed(&mut self, t: &Foo<'a>) {
       self.name = t.name;
   }
}

impl<'a> From<Foo<'a>> for Bar<'a> {
```

```
   fn from(dt: Foo<'a>) -> Self {
       let mut table = Bar::new("bar");
       dt.accept(&mut table);
       table
   }
}

fn main() {
   let f = Foo::new("foo");
   let b: Bar = Bar::from(f);
}
```

Share

Improve this answer

Follow

edited Nov 4, 2023 at 14:06

answered Nov 3, 2023 at 23:44

cafce25

30k55 gold badges4949 silver badges6868 bronze badges

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

1 Comment

Add a comment

seve

seve Over a year ago

Thank you for the clear explanation! I actually had no idea about the `&'a Foo<'a>` issue so you saved me several hours of compiler issues in addition to solving the problem I immediately had.

2023-11-04T03:42:24.657Z+00:00

0

Reply

- Copy link

Your Answer

Thanks for contributing an answer to Stack Overflow!

- Please be sure to *answer the question*. Provide details and share your research!

But *avoid* …

- Asking for help, clarification, or responding to other answers.
- Making statements based on opinion; back them up with references or personal experience.

To learn more, see our tips on writing great answers.

Sign up or log in

Sign up using Google

Sign up using Email and Password

Submit

Post as a guest

Name

Email

Required, but never shown

Post as a guest

Name

Email

Required, but never shown

Post Your Answer Discard

By clicking “Post Your Answer”, you agree to our terms of service and acknowledge you have read our privacy policy.

Start asking to get answers

Find the answer to your question by asking.

Ask question

Explore related questions

- rust
- borrow-checker

See similar questions with these tags.

- The Overflow Blog
  Introducing Stack Overflow AI Assist—a tool for the modern developer
  Treating your agents like microservices
- Featured on Meta
  Chat room owners can now establish room guidelines
  AI Assist is now available on Stack Overflow
  Policy: Generative AI (e.g., ChatGPT) is banned

Related

1
the trait `embedded_hal_digital_InputPin` is not implemented for `PE2<Output<OpenDrain>>`
5
expected trait object `dyn Responsability`, found type parameter `T`
1
BorshDeserialize gives try_from_slice error
422
Why does the Rust compiler not optimize code assuming that two mutable references cannot alias?
4
Factory method: instance does not live long enough
4
Store data that implements a trait in a vector
3
"does not necessarily outlive the lifetime" issue
2
Rust JSON serialization/deserialization with serde/serde_json whilst using generics and lifetimes?

Hot Network Questions

- How to protect 100-year-old wooden floors for the next 100 years?
- Misunderstanding the lorentz transformation

- Curve has weird bumps when triangulated
- Is there any point in sharing a scientific discovery as a "nobody"?
- Download is performed unsandboxed as root - Why am I getting this message, and how do I fix it?
- I can't add a loop cut on one face
- Is there a term for forming emotional or relational bonds with AI chatbots? If not, is “cyberpomorphic” a valid neologism?
- Recent time traveling TV show. The lead might have been Asian(?), set in San Francisco(?)
- Proportional odds logistic regression for ordered category outcome - how to convert odds ratios to probabilities in this case?
- How to reduce GAM sensitivity to sparse data
- Correct website for Thailand ETA application form
- The first Pontrjagin class of Milnor's example of exotic 7-sphere
- What does Felix mean by telling Bond to pick a contact point while "standing up" in Diamonds Are Forever?
- Could a world tree have different biomes?
- Storm Sphere and Obstruction
- Zero Inflated Beta Regression (or not)?
- How to handle Expeditious Retreat with the optional chase rules
- Explain this seeming contradiction in Euclid Book 1 Proposition 16
- Does short range teleporting end floating disk?
- Why does Blender import PLY vertex colors incorrectly (89 → 0.1 instead of 0.349)?
- 2 ways of resolution of vectors
- Finding real variable where a 2×2 complex matrix is singular in Mathematica
- Is the phrase "murder will out" from Chaucer?
- Is there any way to restore hearts without consuming anything, without hot springs, and without shrines?

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.

- Professional
- Business
- API
- Data
- Blog
- Facebook
- Twitter
- LinkedIn
- Instagram