Lifetime issues with async code inside a trait implementation

help

ar3s3ru December 19, 2019, 7:24am 1

Hello peeps,

I'm working on my personal project and I recently stumbled upon the real force of why `async fn` is hard.

I have this nice trait with an async function in it.

The result type of the function, however, is expressed as an associated type, since we don't have GAT yet and `impl Trait` can't be used.

```
pub trait Handler {
   type Command;
   type Aggregate: Aggregate;
   type Error;
   type Result: Future<Output = Result<Vec<EventOf<Self::Aggregate>>, Self::Error>>;


   fn handle(&self, state: &StateOf<Self::Aggregate>, command: Self::Command) -> Self::Result;
}
```

Now, my intention is to write a decorator over this trait.
I already have some code in place, check it out here:

```
impl<H: Handler> Handler for AsHandler<H> {
   type Command = H::Command;
   // Decorated Aggregate type
   type Aggregate = AsAggregate<H::Aggregate>;
   type Error = H::Error;
   type Result = H::Result;


   fn handle(&self, state: &StateOf<Self::Aggregate>, command: Self::Command) -> Self::Result {
       self.0.handle(state, command)
   }
}
```

As you can see, the `type Result` is basically taken as-is from the generic `Handler` implementation we're decorating.

However, I'd like to return a different result here.

Since we have no GAT, I need to box and pin the async block to await the decorated `Handler.handle` method execution and then mapping the result, so it would be something like this:

```
type Result =
   Pin<Box<dyn Future<Output = Result<Vec<EventOf<Self::Aggregate>>, Self::Error>>>>;

fn handle(
   &self,
   state: &StateOf<Self::Aggregate>,
   command: Self::Command,
) -> Self::Result {
   Box::pin(async move {
       let version = state.version();

       self.0.handle(state, command).await.map(|events| {
           events
               .into_iter()
               .map(|event| Versioned::with_version(event, version + 1))
               .collect::<Vec<EventOf<Self::Aggregate>>>()
       })
   })
}
```

However, this is not going to work due to `type Result` assuming `'static` lifetime for `dyn Future`.

Is there any way I can do this?

For complete reference, this is the link to the actual source code if you want to take a look: eventually-rs/versioned.rs at 5beeb2f94260a40bfccf8a28eea9d32b8e3ea796 · get-eventually/eventually-rs · GitHub

Many many thanks to whoever puts some time into this!

alice December 19, 2019, 10:44am 2

Sometimes you can build the traits like this:

```
trait BorrowIter<'a> {
   type Item: 'a;
   fn next(&'a mut self) -> Option<Self::Item>;
}
```

However actually using this kind of trait is a major lifetime pain, e.g. try making this compile.

Generally if you're going to use `Pin<Box<...>>` as the associated type anyway, just put it directly in the trait, because you can specify the lifetime directly as wanted:

```
pub trait Handler {
   type Command;
   type Aggregate: Aggregate;
   type Error;

   fn handle<'a>(&'a self, state: &StateOf<Self::Aggregate>, command: Self::Command)
       -> Pin<Box<dyn Future<Output = Result<Vec<EventOf<Self::Aggregate>>, Self::Error>> + 'a>>;
}
```

ar3s3ru December 19, 2019, 12:33pm 3

Hey @alice thank you for your answer!

> Generally if you're going to use `Pin<Box<...>>` as the associated type anyway, just put it directly in the trait, because you can specify the lifetime directly as wanted:

Yeah, but that's the problem: I generally wouldn't want to limit consumers of the trait to use `Pin<Box<Future>>` explicitly.

On the other hand, however, I can see how the only kind of async code you can write in a trait method as of now (with no GAT support) is necessarily `Pin<Box<Future>>` (unless very particular cases where you could use something like `futures::future::Ready`, but that's more an exception than the norm).

So maybe there's no way out of this with said design as of now, and I just need to sit tight for GAT and use `futures::BoxFuture` everywhere (or `async-trait`).

1 Like

system Closed March 18, 2020, 12:44pm 4

This topic was automatically closed 90 days after the last reply. New replies are no longer allowed.

Related topics

Topic		Replies	Views	Activity
How can i await inside a trait help	2	467	June 17, 2020	
How do you make a trait function return a future with a lifetime? help	5	3278	December 30, 2019	
[Solved] Is it possible to run async code in a trait method (with StdFuture/async/await)? help	9	3982	January 12, 2023	
Newbie: Understanding traits and associated types help	5	610	November 3, 2021	
Async fn in trait with lifetime GAT help	4	473	May 15, 2023	

- Guidelines
- Terms of Service