Beginner lifetime question on AsyncFnOnce vs FnOnce -> impl Future

help

ben.g May 17, 2025, 11:48am 1

Currently developping a backend in axum, and ended up creating this function :

```
pub async fn publish_job<T: Send + 'static, E>(
       &self,
       f: impl AsyncFnOnce(Arc<RwLock<dyn JobProducerInterface>>) -> Result<T, E>,
   ) -> Result<T, MyErrorType>
   where
       E: Into<MyErrorType>,
   {
```

Which triggered this compilation error somewhere high up the stack (at the axum router) :

```
> "implementation of `AsyncFnOnce` is not general enough…
> …
> …note: `{async closure@src/biz/account.rs:81:27: 81:52}` must implement `AsyncFnOnce<(Arc<tokio::sync::RwLock<(dyn producer:
>    = note: ...but it actually implements `AsyncFnOnce<(Arc<tokio::sync::RwLock<dyn producer::JobProducerInterface>>,)>`
```

I ended up changing the *signature* of the function to something **i thought** was strictly equivalent :

```
pub async fn publish_job<T: Send + 'static, Fut, E>(
       &self,
       f: impl FnOnce(Arc<RwLock<dyn JobProducerInterface>>) -> Fut,
   ) -> Result<T, MyError>
   where
       Fut: Future<Output = Result<T, E>>,
       E: Into<MyError>,
```

And to my surprise, the compilation error disappeared. Note that the underlying code hasn't changed. It's strictly the same, i only changed the signature.

Does someone has an explanation ?

1 Like

00100011 May 17, 2025, 2:51pm 2

Having a hard time getting your error to show up: an example like this works just fine.

jonh May 17, 2025, 5:39pm 3

Judging by the note try

```
f: impl for<'a> AsyncFnOnce(Arc<RwLock<dyn JobProducerInterface + 'a>>) -> Result<T, E>,
```

Not sure it should be taking nothing other than 'static though.

`AsyncFnOnce` is not the same as traditional closures. It allows the `Future` to access variables that the closure borrowed.

ben.g May 17, 2025, 8:35pm 4

That worked...

Can you explain to me what happened , and which part of the rust book i should read again to understand that ?

I am absolutely clueless.. I initially thought it was due to that bug Tracking issue for incorrect lifetime bound errors in async · Issue #110338 · rust-lang/rust · GitHub but then thought i would still ask here. About AsyncFnOnce : what kind of trait should i use in my case, where i don't expect the future to use anything captured by the closure ?

ben.g May 17, 2025, 8:43pm 5

jonh found the solution. But in case you want to reproduce :
the code only stopped compiliing once i started using that function with a specific closure containing a specific code :

```
pub async fn my_function(&self, …) {
// ... code..creating data_id: i32
let job_request: JobRequest = self
           .publish_job(async move |job_producer| {
```

```
            job_producer
                .read()
                .await
                .publish_special_job(data_id) // data_id is an i32
                .await
        })
        .await?;
```

then that function is itself called in two layers of async functions, with the topmost one being an axum handler.

00100011 May 17, 2025, 10:05pm 6

ben.g:

> a specific closure containing a specific code

Yup, the initial hunch was on point: capture was definitely involved, after all.

ben.g:

> About AsyncFnOnce : what kind of trait should i use in my case, where i don't expect the future to use anything captured by the closure ?

That's not quite your case, though:

```
// (1) ... code..creating data_id: i32;
let job_request: JobRequest = self
// then:
self.publish_job(async move |job_producer| {
   // (2) capture of `data_id`
   .publish_special_job(data_id)
```

The `Future` is just an associated type, it doesn't "use" anything by itself. Depending on the kind of data you put into it, however - some additional bounds may or may not be required.

ben.g:

> I initially thought it was due to that bug Tracking issue for incorrect lifetime bound errors in async · Issue #110338 · rust-lang/rust · GitHub

Might be - since the error simply vanished when you turned your `impl AsyncFnOnce` into an `impl FnOnce` without changing the `Arc<RwLock<dyn JobProducerInterface>>` passed to it, in any way.

ben.g:

> Can you explain to me what happened , and which part of the rust book i should read again to understand that ?

Going back to the error (with a few omissions for the sake of clarity):

```
`{async closure@<...>}` must implement `X<(Y<(dyn Z + '0)>>,)>`, for any lifetime `'0`...
... but it actually implements `X<(Y<dyn Z>>,)>`
```

That phrasing alone is (usually) quite a clear indicator that HRTBs need to enter the picture. In your case, the compiler has inferred your closure is only able to handle the `X<(Y<dyn Z>>,)>` *specific* to the place/scope in which you've defined it. Meanwhile, the `{async closure}` is required to be able to tackle any `X<(Y<dyn Z + 'a>>,)>` for *any* `'a`.

Thus, the need for a `for <'a>` beforehand [1].

The (beginner's) book only has a chapter on the difference between functions and closures, which isn't going to be of much help here. Take a look at @quinedot's blog instead: [1] and [2].

> as @jonh pointed out, though - it might not be wise to let your closure "loose" over any, arbitrarily *short* lifetime; the explicit `'static` bound makes a lot more sense. ■■

1 Like

system Closed August 15, 2025, 10:06pm 7

This topic was automatically closed 90 days after the last reply. We invite you to open a new topic if you have further questions or comments.

Related topics

Topic	Replies	Views	Activity

Is it possible to achieve these lifetimes in this generic FnOnce -> Future? help	8	575	September 9, 2024
Accessing internal lifetimes in async code (`FnOnce` is not general enough) help	3	358	November 11, 2022
Issue with FnOnce and async function with a &reference type argument help	4	1555	April 21, 2021
Lifetime may not live long enough in async closure? help	5	1493	September 24, 2023
Trait bounds for `Fn` returning a future that forwards the lifetime of the `Fn`'s arguments help	9	1700	November 5, 2021