Passing an async function and then running it on a async thread

help

AmakeSasha January 15, 2025, 5:38pm 1

I have the code in the library:

```
pub async fn launch_server<F>(
   listener: TcpListener,
   function: impl Fn(&mut TcpStream) -> F
) where F: Future<Output = ()> + std::marker::Send + 'static,
{    loop {
       let (mut socket, _) = listener.accept().await.unwrap();
       tokio::spawn(async move {
           function(&mut socket).await;
       });
   }
}
```

But when trying to pass a function to it (outside the library):

```
async fn work(stream: &mut TcpStream) -> () { }
```

Call code:

```
launch_server(listener, work).await
```

I get this error:

```
error: higher-ranked lifetime error
--> src\main.rs:9:5
 |
9 |     launch_server(listener, work).await;
 |     ^^^^^^^^^^^^^
```

kornel January 15, 2025, 6:49pm 2

Are you using the latest version of Rust? Do you get equally unhelpful error in nightly?

I've tried to reproduce this problem, and I've got several errors, but not this one.

Your `function` is moved in a loop. Moves take *exclusive* ownership of the object, and the scope the object has been moved from is not allowed to use it any more. This exclusivity is in conflict with loops, since the loop needs to keep the object for itself and can't move it anywhere, if it wants to have it for the next iteration. You need to make your `function` implement `Copy` or `Clone` and then you need to `clone()` it for every iteration of the loop.

The `function` is used inside `Send` + `'static` future, so the `function` must also implement `Send + 'static`:

```
function: impl Fn(&mut TcpStream) -> F + Copy + Send + 'static
```

jofas January 15, 2025, 7:32pm 3

Even when you don't use `stream` in the body of your async function, it is still captured in the opaque future returned from `work`. This makes the returned future not `'static`, but have the lifetime of `stream`. I.e. `work` desugars to:

```
fn work<'a>(stream: &'a mut TcpStream) -> Future<Output=()> + 'a {
   async move {}
}
```

(Edit: see @quinedot's answer below for how it really desugars)

The async closures RFC explains the weird error message far better than I could:

> This happens because the type for the `Fut` generic parameter is chosen by the caller, in `main`, and it cannot reference the higher-ranked lifetime `for<'c>` in the `FnMut` trait bound, but the anonymous future produced by calling `do_something` does capture a generic lifetime parameter, and *must* capture it in order to use the `&str` argument.

My suggestion would be to change your implementation such that your callback takes `TcpStream` by value and not by mutable reference while the `AsyncFn` traits are still unstable:

```
use std::future::Future;
use tokio::net::{TcpListener, TcpStream};

pub async fn launch_server<F>(
   listener: TcpListener,
   function: impl Fn(TcpStream) -> F + Send + Copy + 'static,
) where F: Future<Output = ()> + Send,
{    loop {
        let (socket, _) = listener.accept().await.unwrap();
        tokio::spawn(async move {
            function(socket).await;
        });
   }
}

async fn work(_stream: TcpStream) -> () { }

#[tokio::main]
async fn main() {
   let listener = TcpListener::bind("localhost:8080").await.unwrap();
   launch_server(listener, work).await;
}
```

Playground.

1 Like

quinedot January 15, 2025, 7:40pm 4

jofas:

> I.e. `work` desugars to:

Nits:

```
// Don't apply an outlives bound, but do capture all generics
// (and you need the `impl`)
fn work<'a>(stream: &'a mut TcpStream) -> impl Future<Output=()> + use<'a> {
   async move {
       // Unconditionally capture all arguments
       let stream = stream;
       // (original body here)
   }
}
```

3 Likes

AmakeSasha January 15, 2025, 11:52pm 5

Although I couldn't get this code to compile , it explains pretty well how it actually works.

AmakeSasha January 16, 2025, 12:10am 6

jofas:

> My suggestion would be to change your implementation such that your callback takes `TcpStream` by value and not by mutable reference while the `AsyncFn` traits are still unstable:

What to do if it is necessary to pass & or `&mut`?

jofas January 16, 2025, 9:21am 7

You could still pass `TcpStream` by value and return it back from the future when it is done with it. Example.

You can also work with a concrete `dyn Future` type instead of an opaque future. This allows you to get rid of the problematic F bound on `launch_server`. Example.

1 Like

AmakeSasha January 16, 2025, 4:33pm 8

Thank you, I will use option 1, because it suits me the most. And I have never had such ideas before xD

1 Like

system Closed April 16, 2025, 4:33pm 9

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
How to pass async function which accepts parameter with lifetime to other function? help	4	690	October 31, 2023
Help! One type is more general than the other help	7	163	May 24, 2025
Understanding rust's opaque types help	4	1218	February 2, 2024
Async function as an argument - one type is more general than the other	10	422	June 9, 2024
How to pass a closure that returns a Future, as function argument? help	7	7645	December 5, 2021