Rust ownership in loops

Asked 1 year, 3 months ago
Active 1 year, 3 months ago
Viewed 582 times

1

I'm trying to implement rabbitmq send/listen functionality in Rust and I have the following code:

```
struct RabbitMQ {
  connection: Connection,
}

impl RabbitMQ {
  fn connect() -> Self {
    RabbitMQ {
      connection: the created connection
    }
  }
}

impl MessageBroker for RabbitMQ {
  async fn publish(&self, topic: &Topic) -> Result<PublisherConfirm, Error> {
    let channel = self.connection.create_channel().await.unwrap();

    RabbitMQ::create_exchange(&channel, &topic.exchange).await;

    let payload = topic.message.as_bytes();

    let res = channel.basic_publish(
      topic.exchange.name.as_str(),
      topic.queue.routing_key.as_str(),
      topic.exchange.publish_options,
      payload.to_vec(),
      BasicProperties::default(),
    );

    res.await
  }
}
```

So far so good!

Now I want to publish many messages in a for loop without waiting for the confirmation from the server, the problem is that when I spawn tokio async task I need to move my broker value and this makes it invalid for the next iteration of the loop:

```
let broker = RabbitMQ::connect(&connection_details).await;

for x in 1..10 {
  tokio::spawn(async move {
    let confirm = broker.publish(&my_topic).await.unwrap();
  }).await.unwrap();
}
```

The above code won't compile with the following error:

```
 error[E0382]: use of moved value: `broker`
  --> src/main.rs:47:33
   |
21 |     let broker = RabbitMQ::connect(&connection_details).await;
   |         ------ move occurs because `broker` has type `message_broker::RabbitMQ`, which >does not implement the `Copy` trait
...
47 |         tokio::spawn(async move {
   | _______________________________________^
48 | |           let confirm = &broker.publish(&enable_cdn).await.unwrap();
   | |                         ------ use occurs due to use in generator
49 | |       }).await.unwrap();
   | |_______^ value moved here, in previous iteration of loop
```

I can't implement the **Copy** trait as **Connection** isn't primitive and it seems that I can't use reference "&" to the broker.

My question is how can I accomplish this without writing **n** publish calls?

rust | async-await | ownership | rust-tokio

Share
Improve this question
Follow
edited Sep 25 '20 at 7:37
asked Sep 24 '20 at 12:55
Martin Petkov
13 • 4

does broker expose a clone or try_clone method? If yes try cloning the connection (broker) inside for loop to a variable and pass it in the tokio::spawn.
– Saud Qureshi
Sep 24 '20 at 13:23

unfortunately no
– Martin Petkov
Sep 25 '20 at 6:42

Add a comment

1 Answer

Active Oldest Votes

3

You're using an `async move` block, which means any *name* which is used in the block is moved into the future, regardless of the operations being performed. So writing

```
&broker.publish
```

inside the block makes no difference: first `broker` is moved, and the future (when polled with `.await`) takes an internal reference to it. So what you need to do is borrow outside the block then move that borrow inside:

```
let broker = RabbitMQ::connect(&connection_details).await;

for x in 1..10 {
    let broker = &broker;
    tokio::spawn(async move {
        let confirm = broker.publish(&enable_cdn).await.unwrap();
    }).await.unwrap();
}
```

but I *think* that's not going to work either: `tokio::spawn` is not scoped, so even though you're await-ing it, the compiler has no idea that it will not outlive `broker`. As far as it's concerned a tokio task can live as long as it wants. This means you're now probably going to get a lifetime error (the compiler will assume the borrow *can* outlive the enclosing function, and thus its origin).

An easy solution to that would be to put the Connection behind an `Arc` or something.

Alternatively, restructure your system to work better with the requirements of rabbitmq: no idea which you're using but amiquip states that connections are thread-safe, and *channels* while not thread-safe can be sent to other threads.

So rather than publish-ing to an implicit connection, in each iteration of the loop create a channel and move *that* into the task in order to actually perform the publication.

Also,

> Now I want to publish many messages in a for loop without waiting for the confirmation from the server

aren't you still doing that since you're awaiting the result of tokio::spawn?

Share
Improve this answer
Follow
edited Sep 25 '20 at 12:05

trent formerly cl
20.1k 7 42 72

answered Sep 24 '20 at 14:00

Masklinn
20.6k 1 17 34

1

Couldn't the OP solve the second problem by using `Arc`? I.e. change `let broker = RabbitMQ::connect(&connection_details).await` to `let broker = Arc::new(RabbitMQ::connect(&connection_details).await)` and, in the `for` loop, add `let broker = Arc::clone(&broker)`.
– user4815162342
Sep 24 '20 at 15:42

@user4815162342 I originally wrote that, but the amiquip documentation (don't know if that's what OP uses but the API looks very similar) states that creating a channel requires exclusive access to the connection, hence the short paragraph about the exclusive lock.
– Masklinn
Sep 25 '20 at 5:56

1

The need for a lock is an orthogonal concern, but I think `Arc` will still be needed for the lifetime. I.e. `Arc::new(...)` or `Arc::Mutex::new(...))`.
– user4815162342
Sep 25 '20 at 6:06

Thanks for the pinpoints guys! Sorry, I've forgot to mention, I'm using the lapin crate and I'm trying to implement interface which can be used by other brokers, that is why my approach with the publish function and the connection is such. The use of Arc helped and my code compiled! I'm further planing to collect the futures in vec or something and then awaiting them after the loop or something like that :) As for the approach to the broker implementation itself I'll see what will happen further with this but I as well thing that passing only the rabbitmq channels around is more proper way.
– Martin Petkov

Sep 25 '20 at 7:34

Add a comment

Your Answer

Post Your Answer

By clicking "Post Your Answer", you agree to our terms of service, privacy policy and cookie policy

Not the answer you're looking for? Browse other questions tagged rust async-await ownership rust-tokio or ask your own question.

Related

Why doesn't println! work in Rust unit tests?

1
How to borrow/avoid a move of a socket in tokio::spawn(async

Rust Multithread Asynchronous Websocket Server

How to send asynchronous messages into a sink through a futures channel?

Hot Network Questions

which one of these paths has the priority: /usr or /usr/local

What's the social meaning of "He was a student of..."?

Why did the JWST solar array deploy early?

How to plot molecules with angles and bond lengths

Is it possible to convert a taproot address into a native segwit address?

more hot questions

Question feed