E0515: general options for finding way back to an owned type, for objects you don't author?

help

daxodev December 28, 2024, 5:21pm 1

tl;dr I'm hitting a dangling reference case, but I'm curious what the general strategy is here. Is there not one? (eg: I'm just out of luck if my object doesn't provide `clone()`?)

So my specific case (playground) is I have a helper method that's constructing a `std::process::Command` in multiple places, so I want to let the builder's initial setup be done by a private helper, but then have callers own the structure (heap or stack, I don't really care for this tiny object):

```
    /// Sample usage:
   /// ```
   /// self.start_cmd()
   ///     .stdout(Stdio::null())
   ///     .output()
   ///     .expect("failed proc")
   /// ```
   fn start_cmd(&self) -> Command {
       Command::new("cli")
           .arg("sub-cmd")
           .arg("--flag1")
           .arg("--flag2")
           // ... potentially more things...
           .arg("--flag3")
           .current_dir(self.dir.clone())
           // TODO: need something here; .into() doens't work
   }
```

I *believe* the problem is: `Command::new()` is returning an "owned type" like I want but then once I use builders, I've switched into a "reference type" `&mut Command` (which is the the callout below: *"you can never transfer ownership using a reference type"*). I've found the the first and third sentence in redditor (u/Darksonn)'s answer really helpful:

> To return an owned value, you must use an owned type. In this case that would be a `String`. References are always borrowed, with no exceptions, and you can never transfer ownership using a reference type.
>
> As for the missing ampersand, it's somewhere inside the `as_str` method.

So my question: what's the general solution/pattern when I hit cases like this, where my goal is:

1. avoid copy/pasting the same N lines
   - ie: trying to have said lines maintained in a single location (eg: `start_cmd()` method)
2. ownership should be with the callers
3. the object I'm returning just happens not to be an owned type
   - (not general solution) said object *happens* to satisfy `ToOwned`? would that have even helped me?
   - (not general solution) ... similarly if it *happened* to provide `.clone()`?

jofas December 28, 2024, 5:45pm 2

You might be overcomplicating things looking for general solutions/patterns. You can usually construct owned objects inside of a function and return them. That goes for `Command` as well. If I want to return a reference to a local object, I just return the object itself and let the caller (mostly also me) get the reference when needed. If you are truly hell-bend on abstracting something into its own unit that needs to construct owned data first but you really want to return a reference to it, you could use a declarative macro instead of a function to avoid dropping the owned data, binding it to some secret variable in the scope of the caller.

3 Likes

steffahn December 28, 2024, 6:09pm 3

Some slight syntactic relief, compared to the implementation @jofas already provided, can come from helpers such as the Tap extension trait:

```
use std::process::Command;
use tap::Tap;

struct Runner {
   dir: String,
}

impl Runner {
   /// Sample usage:
```

```
    /// ```
    /// self.start_cmd()
    ///     .stdout(Stdio::null())
    ///     .output()
    ///     .expect("failed proc")
    /// ```
    fn start_cmd(&self) -> Command {
        Command::new("cli").tap_mut(|c| {
            c.arg("sub-cmd")
                .arg("--flag1")
                .arg("--flag2")
                // ... potentially more things...
                .arg("--flag3")
                .current_dir(self.dir.clone());
        })
    }
}
```

(playground)

1 Like

kpreid December 28, 2024, 6:15pm 4

Trying to give a different, possibly clearer, explanation:

The point at which your code paints itself into a corner is right at `Command::new("cli").arg("sub-cmd")`. `Command::new()` returns (owned) `Command`, but `arg()` takes `&mut Command` and returns `&mut Command`, so right at this point is where you have lost the ability to return `Command` because the `Command` is living in an unnamed temporary place the compiler creates for the implicit `&mut` borrow, so you can't[1] retrieve it to return it.

The solution, therefore, is to specifically put *the owned `Command`* in a variable:

```
fn start_cmd(&self) -> Command {
    let cmd: Command = Command::new("cli");

    // This makes a temporary &mut borrow but doesn't consume `cmd`.
    // It returns `&mut Command` but we do not use it in this case.
    cmd.arg("sub-cmd");

    // return the Command, *not* &mut Command
    cmd
}
```

As long as you do this, you can call as many builder methods you want in the middle.

Also, note that there are two flavors of builder pattern in Rust. `Command` uses this one where each builder method takes and returns `&mut Self`, but there is also the one where each builder method takes and returns `Self` (or, sometimes, returns a different type). In *that* case, your original code would be correct.

> well, without `std::mem::replace()`, which would be silly here ■■

8 Likes

daxodev December 28, 2024, 7:58pm 5

kpreid:

> The point at which your code paints itself into a corner is right at `Command::new("cli").arg("sub-cmd")`. `Command::new()` returns (owned) `Command`, but `arg()` takes `&mut Command` and returns `&mut Command`, so right at this point [...]
>
> The solution, therefore, is to specifically put *the owned `Command`* in a variable:
>
> ```
> fn start_cmd(&self) -> Command {
>     let cmd: Command = Command::new("cli");
>
>     // This makes a temporary &mut borrow but doesn't consume `cmd`.
>     // It returns `&mut Command` but we do not use it in this case.
>     cmd.arg("sub-cmd");
>
>     // return the Command, *not* &mut Command
>     cmd
> }
> ```
>
> As long as you do this, you can call as many builder methods you want in the middle.

Aaaahh thank you for spelling this out! My goodness I can't believe I didn't see that this was happening. I even recognized the two types, but didn't think to just save my own variable (instead of continuing chaining builder-pattern). And I see now how the other replies are just tackling the same issue too.

Thanks everyone for your answers. My seeking a "pattern" was a bit of xy-problem confusion I think. (the result of my assuming I must still be misunderstanding *something* about what the compiler is doing, so *maybe* there'd be an ownership pattern people use that would show me what I misunderstood).

kpreid December 28, 2024, 9:40pm 6

daxodev:

> My seeking a "pattern" was a bit of xy-problem confusion I think. (the result of my assuming I must still be misunderstanding *something* about what the compiler is doing, so _maybe _there'd be an ownership pattern people use that would show me what I misunderstood).

I think there *is* a general principle worth learning here, it's just a very obvious-once-understood one: if you need ownership, make sure you don't give it away. Or, differently: going from `T` to `&T` is irreversible (in the absence of cloning).

1 Like

system Closed March 28, 2025, 9:41pm 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
How to return the ownership of a std::process::Command variable help	3	1028	July 10, 2020	
Struggling with this basic borrow help	16	2379	January 12, 2023	
Pattern for "RefOrMove" arguments? help	19	536	July 11, 2023	
How to avoid a borrow checker issue without duplicating part of my implementation? help	15	890	December 14, 2020	
The best way to return a value from a builder help	16	2192	January 12, 2023	