Skip to main content

Stack Overflow

1. About
2. Products
3. For Teams

1. Stack Internal Implement a knowledge platform layer to power your enterprise and AI tools.
2. Stack Data Licensing Get access to top-class technical expertise with trusted & attributed content.
3. Stack Ads Connect your brand to the world’s most trusted technologist communities.
4. Releases Keep up-to-date on features we add to Stack Overflow and Stack Internal.
5. About the company Visit the blog

Loading…

current community

Stack Overflow
help chat
- Meta Stack Overflow

your communities

Sign up or log in to customize your list.

more stack exchange communities

company blog

3. Log in
4. Sign up

AI Assist is now on Stack Overflow. Start a chat to get instant answers from across the network. Sign up to save and share your chats.

Home
Questions
AI Assist
Tags
Challenges
7. Chat
Articles
Users
Companies
Collectives

Communities for your favorite technologies. Explore all Collectives

Stack Internal

Stack Overflow for Teams is now called **Stack Internal**. Bring the best of human thought and AI automation together at your work.

Try for free Learn more

Stack Internal

Bring the best of human thought and AI automation together at your work. Learn more

How to move one field out of a struct that implements Drop trait?

Ask Question

Asked 10 years, 5 months ago

Modified 1 year, 9 months ago

Viewed 17k times

36

Here's an invalid Rust program (Rust version 1.1) with a function that does an HTTP client request and returns only the headers, dropping all other fields in the response.

```
extern crate hyper;

fn just_the_headers() -> Result<hyper::header::Headers, hyper::error::Error> {
   let c = hyper::client::Client::new();
   let result = c.get("http://www.example.com").send();
   match result {
       Err(e) => Err(e),
       Ok(response) => Ok(response.headers),
   }
}

fn main() {
   println!("{:?}", just_the_headers());
}
```

Here are the compiler errors:

```
main.rs:8:28: 8:44 error: cannot move out of type `hyper::client::response::Response`, which defines the `Drop` trait
main.rs:8         Ok(response) => Ok(response.headers),
                                ^~~~~~~~~~~~~~~~
error: aborting due to previous error
```

I understand *why* the borrow checker doesn't accept this program—i.e., that the `drop` function will use the `response` after it has had its `headers` member moved.

My question is: How can I get around this and still have good safe Rust code? I know I can do a *copy*, via `clone()`, like so:

```
Ok(response) => Ok(response.headers.clone()),
```

But, coming from C++, that seems inefficient. Why *copy* when a *move* should suffice? I envision in C++ doing something like the following to force a call to a move constructor, if available:

```
headers_to_return = std::move(response.headers);
```

Is there any way to forgo the *copy* in Rust and instead force a *move*, similar to C++?

- rust

Share

Improve this question

Follow

edited May 25, 2018 at 19:19

Boiethios

43.7k2525 gold badges151151 silver badges198198 bronze badges

asked Jul 9, 2015 at 3:36

Craig M. Brandenburg

3,68466 gold badges2828 silver badges3939 bronze badges

1

This won't technically move the member value, but if you're OK to change your structure a bit, but you can wrap the headers by changing the type of `response.headers` to `Option<Headers>` and `take()` its value. This will reset the value to None, which is useful if you're unable to find a good default value for your type (e.g. a `thread`). This is done in doc.rust-lang.org/stable/book/ch17-03-oo-design-patterns.html and doc.rust-lang.org/stable/book/…

hsandt – hsandt

2019-07-13 19:48:02 +00:00

Commented Jul 13, 2019 at 19:48

Add a comment |

3 Answers

Sorted by: Reset to default

Highest score (default) Trending (recent votes count more) Date modified (newest first) Date created (oldest first)

48

You can use `std::mem::replace()` to swap the field with a new blank value in order to transfer ownership to you:

```
extern crate hyper;

fn just_the_headers() -> Result<hyper::header::Headers, hyper::error::Error> {
   let c = hyper::client::Client::new();
   let result = c.get("http://www.example.com").send();
   match result {
       Err(e) => Err(e),
       Ok(mut response) => Ok(std::mem::replace(&mut response.headers, hyper::header::Headers::new())),
   }
}

fn main() {
   println!("{:?}", just_the_headers());
}
```

Here, we're replacing `response.headers` with a new empty set of headers. `replace()` returns the value that was stored in the field before we replaced it.

Share

Improve this answer

Follow

answered Jul 9, 2015 at 4:44

Francis Gagné

66.7k1313 gold badges201201 silver badges172172 bronze badges

Sign up to request clarification or add additional context in comments.

6 Comments

Add a comment

DK.

DK. Over a year ago

It might be worth nothing that `std::mem::replace` in Rust is *more or less* what `std::move` is in C++. Because the source and destination must be valid to destruct both before *and after* a move, C++ doesn't really move, it swaps.

2015-07-09T05:02:24.723Z+00:00

6

Reply

- Copy link

Francis Gagné

Francis Gagné Over a year ago

Indeed, with the difference that in C++, it is the class that decides how to implement the move (in the move constructor or move assignment operator), whereas `std::mem::replace` requires the caller to provide a suitable value. In fact, `std::mem::replace` is implemented in terms of `std::mem::swap`.

2015-07-09T05:05:35.12Z+00:00

4

Reply

- Copy link

Matthieu M.

Matthieu M. Over a year ago

You might wish to note that Rust also places on emphasis on extremely efficient "default constructs", for example neither `String::new()` nor `Vec::new()` allocate memory, which is what makes this replace as efficient as the C++ move on top of being safer.

2015-07-09T08:04:24.277Z+00:00

5

Reply

- Copy link

Craig M. Brandenburg

Craig M. Brandenburg Over a year ago

Thanks! Is using `std::mem::replace` for this use case idiomatic? (Is what I'm trying to do — to force a *move* instead of a *copy* — idiomatic?) I ask because the call to `std::mem::replace` seems like a lot of typing to do something that could be a common use case.

2015-07-09T12:27:49.81Z+00:00

1

Reply

- Copy link

Francis Gagné

Francis Gagné Over a year ago

`std::mem::replace` seems to be the most suitable tool to use to take ownership of a value you can't take ownership of with Rust's standard ownership rules. Don't forget you can use `use` declarations to make names shorter, e.g. `use std::mem;`, then `mem::replace`, or `use std::mem::replace`, then `replace`. The preferred style is to use functions qualified on the module (`mem::replace`), but types unqualified (`Headers`).

2015-07-09T23:42:32.84Z+00:00

2

Reply

- Copy link

Add a comment | Show 1 more comment

2

As of Rust 1.40.0 (released in December 2019), `std::mem::take` should be used. Rust docs mention:

```
pub fn take<T>(dest: &mut T) -> T
where
   T: Default,


Replaces dest with the default value of T, returning the previous dest value.
```

This is how it will be used in your example:

```
fn just_the_headers() -> Result<hyper::header::Headers, hyper::error::Error> {
   let c = hyper::client::Client::new();
   let result = c.get("http://www.example.com").send();
   match result {
       Err(e) => Err(e),
       Ok(response) => Ok(std::mem::take(&mut response.headers)),
   }
}

fn main() {
   println!("{:?}", just_the_headers());
}
```

Share

Improve this answer

Follow

answered Feb 18, 2024 at 18:30

Arpit Saxena

6733 silver badges77 bronze badges

Comments

Add a comment

1

It's worth noting that the `hyper` crate now has a different API which supports taking the headers by value. Here is the solution for hyper 14.2 (Rust edition 2021, version 1.69):

```
extern crate http;
extern crate hyper;

async fn just_the_headers() -> Result<http::header::HeaderMap, hyper::Error> {
   let c = hyper::client::Client::new();
   let result = c
       .get(hyper::Uri::from_static("http://www.example.com"))
       .await;
   match result {
       Err(e) => Err(e),
       Ok(response) => Ok(response.into_parts().0.headers),
   }
}
```

A lot has changed since Rust 1.1. async/await is now a first-class part of the language (stabilized in Rust 1.39). We also have the new ? operator, stabilized in Rust 1.13. Using this operator, the code becomes

```
extern crate http;
extern crate hyper;

async fn just_the_headers() -> Result<http::header::HeaderMap, hyper::Error> {
   Ok(hyper::client::Client::new()
       .get(hyper::Uri::from_static("http://www.example.com"))
       .await?
       .into_parts()
       .0
       .headers)
}
```

Share

Improve this answer

Follow

answered May 22, 2023 at 23:13

Mark Saving

1,81799 silver badges1212 bronze badges

Comments

Add a comment

Your Answer

Name

Email

Required, but never shown

Post Your Answer Discard

Start asking to get answers

Find the answer to your question by asking.

Ask question

Explore related questions

- rust

See similar questions with these tags.

- The Overflow Blog
  Introducing Stack Overflow AI Assist—a tool for the modern developer
  Treating your agents like microservices
- Featured on Meta
  Chat room owners can now establish room guidelines
  AI Assist is now available on Stack Overflow
  Policy: Generative AI (e.g., ChatGPT) is banned

Linked

16
How can I move a value out of the argument to Drop::drop()?
8
How do I destructure an object without dropping it?
4
How to move a field whose type does not implement Default from a struct which implements Drop?
0
Borrow checker problems for parser that can delegate
0
method that mutates `self` and consumes one of the `struct` field (err: move behind a mutable reference)

Related

6
Can not move out of type which defines the `Drop` trait [E0509]
2
Maybe move field inside struct method
5
Easy way to change one field in struct?
16
How can I move a value out of the argument to Drop::drop()?
4
How to take away (move out of) a struct field that contains a big datastructure type?
4
How to move a field whose type does not implement Default from a struct which implements Drop?
1
Is it available to drop a variable holding a primitive value in Rust?
2
How to remove a struct's field?
1
how to implement "drop" on a generic struct where T itself implements a type

1
Do I need to specify 'drop(item)' if i write a custom drop implementation for a struct? Rust

Hot Network Questions

- The first Pontrjagin class of Milnor's example of exotic 7-sphere
- Is there a website where we can find the average stipend/salary for the PhD and Postdoc for each country?
- Polygons (and some stars)
- Can four points in a unit square have mutual distances all larger than 1?
- How to check if two tables are identical?
- Curve has weird bumps when triangulated
- EEPROM Decoupling Capacitor
- Download is performed unsandboxed as root - Why am I getting this message, and how do I fix it?
- Generating Irregular, Separated Polygonal Cells along a Curve in Geometry Nodes
- What do the terms postponement, slip and scrub mean in the context of the space shuttle program?
- How do Buddhists interpret the Buddha’s explanation of earthquakes in AN 8.70?
- Finding real variable where a 2×2 complex matrix is singular in Mathematica
- Can enemies damage each other in Breath of the Wild?
- Is the phrase "murder will out" from Chaucer?
- How many alien worlds could exist within 100 ly without us noticing?
- Obscure method of supression of transient voltage spike in CLC-circuit
- I can't add a loop cut on one face
- How can I determine whether a complex rational equation has real solutions?
- How to protect 100-year-old wooden floors for the next 100 years?
- "Us" in Hegel's "Greater Logic"
- Is my homebrew Species "Attuned" balanced for D&D 2024?
- How can I secure a loose kitchen cabinet?
- How to add blur inside subtracted element in Figma?
- Why doesn’t sleep mode (S3) turn on in the BIOS on my Lecoo Pro 14 laptop (N155A, Lenovo sub-brand) with Windows 11?

more hot questions

Question feed

Subscribe to RSS

Question feed

To subscribe to this RSS feed, copy and paste this URL into your RSS reader.