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

Collectives™ on Stack Overflow

Find centralized, trusted content and collaborate around the technologies you use most.

Learn more about Collectives

Stack Internal

Knowledge at work

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

Explore Stack Internal

How to create an object that can build many other objects all of which have a mutable reference back to the builder object?

Ask Question

Asked 3 years, 5 months ago

Modified 3 years, 5 months ago

Viewed 120 times

1

I have a bit of a difficulty formulating my question concisely enough for a title. Here is the mockup of what I am trying to ask.

I have a `Thing` struct, a `ThingMaker` and a `ThingProcessor`. A `ThingMaker` must be able to create multiple `Things` that are then passed to a `ThingProcessor` to be used.

The catch is that a `ThingProcessor` needs to have mutable access to the `ThingMaker` that made the `Thing` so that the `ThingMaker` can update its internal state using feedback from the `ThingProcessor`. While in code provided there is only one `ThingMaker`, in my real project there are multiple, so the `Things` need to include information on what object produced them.

The naive approach of storing the mutable reference in the `Thing` does not work due to the fact that you cannot have more than one mutable reference to an object at a time, and a `ThingMaker` makes more than one object requiring it.

I also found this question that seems very close to mine, but I cannot seem to adapt it to my case. I could change the definition of `Thing` so it keeps a reference back to the `ThingMaker` through a `RefCell`, but then how is `ThingMaker` supposed to make `Things`?

- rust

Share

Improve this question

Follow

asked Jul 5, 2022 at 14:15

Dizzar

7744 bronze badges

7

`Thing` can own a refcounted smart-pointer back to its maker, e.g. `Rc<RefCell<ThingMaker>>`. See playground.

eggyal – eggyal

2022-07-05 14:33:50 +00:00

Commented Jul 5, 2022 at 14:33

1

Another option is to have `ThingMaker` use interior mutability via `RefCell` (or other tool), so that you only need an immutable reference, `&ThingMaker`, in order to mutate it.

kmdreko – kmdreko

2022-07-05 14:41:25 +00:00

Commented Jul 5, 2022 at 14:41

Right, so. @ChayimFriedman I just opened the playground from a google search and started writing code. It seems it defaults to 2018. The structures I described should not be self-referrential, that is, `Things` point to their makers, but not the other way around, as @kmdreko said.

Dizzar – Dizzar

2022-07-05 14:49:59 +00:00

Commented Jul 5, 2022 at 14:49

As it stands, I had thought of both proposed approaches. Making the type use interior mutability would be clunky (imo at least. dont like using cells and refcells), so I hope to avoid that. Using a smart pointer may be a good solution, I just had a little bit of trouble figuring out how to construct the `Things`, thaks @eggyal for providing an example.

Dizzar – Dizzar

2022-07-05 14:57:12 +00:00

Commented Jul 5, 2022 at 14:57

2

@Dizzar you'll need a `Cell`/`RefCell`/`Mutex` or something with interior mutability to allow for shared mutability, whether its outside the type, `RefCell<ThingMaker>`, or inside, `ThingMaker { inner: RefCell }`, is up to you. The only other option that wouldn't is if the `Things` only had an *identifier* for what `ThingMaker` made it, and have the `ThingProcessor` do the lookup-by-id itself using some other structure.

kmdreko – kmdreko

2022-07-05 15:37:07 +00:00

Commented Jul 5, 2022 at 15:37

| Show **2** more comments

1 Answer

Sorted by: Reset to default

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

0

This can be achieved in two ways. Using smart-pointers or introducing internal mutability to the `ThingMaker`.

Smart-pointers

Using this approach we wrap our reference to `ThingMaker` in `Rc<RefCell>` so that we can have multiple mutable references to one object. There is a limitation to this method (correct me if I'm wrong) - you cannot easily make new `Things` from inside the `ThingMakers` methods. That is because we cannot easily obtain the `Rc` reference to an object from inside said object. Note that neither cells nor `Rc` are thread safe, so if you have a multi-threaded environment use `Arc` instead of `Rc` and `Mutex` instead of cells. Here is an example using `Rc<RefCell>` smart-pointer. (thanks @eggyal)

In this expample, we rewrote `Thing` to store a smart-pointer instead of a direct reference:

```
struct Thing {
   pub parent: Rc<RefCell<ThingMaker>>,
}
```

we then have to wrap our maker in a `Rc<RefCell>` smart-pointer, like this:

```
let maker = Rc::new(RefCell::new(ThingMaker { data: 16 }));
```

Which we then give to creation function to make `Things`:

```
ThingMaker::make_thing(&maker);
```

Later, we can access and mutate `ThingMaker` using a `Things` reference like so:

```
let mut parent = thing.parent.borrow_mut();
parent.data += 1;
```

`Rc` smart-pointer allows shared ownership of `ThingMaker` between `Things`, and the `RefCell` allows us to reintroduce mutability, seeing as `Rc` is an immutable pointer.

Interior mutability

This method makes it so that we do not *need* a mutable reference to mutate our object, only an immutable one. As we can have as many immutable referenes as we want, we can make as many `Things` as we want. To achieve this we can use `Cells` and `RefCells` (refer to documentation to see which one will be better in you case). Additionally, if you are in a multi-threaded environment, you can use `Mutex`, as cells are not thread safe. Here is an example of using `RefCell` to introduce intrerior mutability. (thanks @kmdreko)

As you can see, the data of our `ThingMaker` was wrapped in a `RefCell`:

```
struct ThingMaker {
   data: RefCell<u64>,
}
```

Which we can then use to get mutable references to our data:

```
fn process_thing(obj: &Thing) {
   let mut data = &mut *obj.parent.data.borrow_mut();
   *data += 1;
   println!("New data: {}", data);
}
```

Note that we only need an immutable reference to mutate our object, so we can get away with only storing immutable references inside our `Things`:

```
struct Thing<'a> {
   pub parent: &'a ThingMaker,
}
```

Share

Improve this answer

Follow

answered Jul 6, 2022 at 2:03

Dizzar

7744 bronze badges

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

1 Comment

Add a comment

eggyal

eggyal Over a year ago

To be fair, both approaches use interior mutability to solve the problem posed in your question. The refcounted smart-pointers add "shared ownership", so that each `Thing` can "own" its maker: this avoids a whole host of lifetime shenanigans that would otherwise arise, but is actually separate from the mutability question posed.

2022-07-07T13:53:45.587Z+00:00

0

Reply

- Copy link

Your Answer

Thanks for contributing an answer to Stack Overflow!

- Please be sure to *answer the question*. Provide details and share your research!

But *avoid* …

- Asking for help, clarification, or responding to other answers.
- Making statements based on opinion; back them up with references or personal experience.

To learn more, see our tips on writing great answers.

Sign up or log in

Sign up using Google

Sign up using Email and Password

Submit

Post as a guest

Name

Email

Required, but never shown

Post as a guest

Name

Email

Required, but never shown

Post Your Answer Discard

By clicking “Post Your Answer”, you agree to our terms of service and acknowledge you have read our privacy policy.

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

17
How to represent shared mutable state?

Related

2
In Rust (0.5 and/or trunk), how do I create a mutable vector of mutable objects?
48
How can I create my own data structure with an iterator that returns mutable references?
3
How can I make a structure with internal references?
34
How do I create a heterogeneous collection of objects?
4
How do I give a mutable reference to a builder but only an immutable reference to the built object?
1
How do I create a struct of references to traits when one object might implement multiple of the traits?
0
Mutable pass-by-mutable-reference in Rust
0
immutable and mutable reference needed for object inside struct
1
Rust: one mutable reference to a particular piece of data at a time?
4
Immutable struct with mutable reference members

Hot Network Questions

- Recent time traveling TV show. The lead might have been Asian(?), set in San Francisco(?)
- How do I add a text editor to my (debian-based) initramfs?
- Download is performed unsandboxed as root - Why am I getting this message, and how do I fix it?
- Who is Patrick and why is he referred to at anti-AfD demonstrations?
- Vertical alignment of equations inside tabularx
- Is "We will review application starting from..." a deadline?
- I2C bus problems when using two distance sensors in Arduino
- Is it possible to use LaTeX to create RPG-related PDFs, such as character sheets or adventure modules?
- Should these weep hole covers be removed from the exterior of sliding glass door?
- How many alien worlds could exist within 100 ly without us noticing?
- Storm Sphere and Obstruction
- PSE Advent Calendar 2025 (Day 3): A gift from my travel agent
- Do something and "win"
- How can I secure a loose kitchen cabinet?

- The fourth moment of the Riemann zeta function without absolute values
- How do we define time-ordering operations in QFT?
- Statistical testing for bimodal/multimodal sample
- Why is "cmd | xargs" to join lines of cmd with spaces wrong
- Displaying features in a layer of geometrytype = GeometryCollection25D
- TV program where real actors interact with 2-Dimensional characters
- How does supporting "One China" work without also supporting Taiwan-China reunification?
- What's the point of swapping polarity between mod slots?
- The state of icy worlds during differentiation and accretion, and how they survive with their water
- Ping sweep of IP subnets

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.

Stack Overflow

- Questions
- Help
- Chat

Business

- Stack Internal
- Stack Data Licensing
- Stack Ads

Company

- About
- Press
- Work Here
- Legal
- Privacy Policy
- Terms of Service
- Contact Us
- Cookie Settings
- Cookie Policy

Stack Exchange Network

- Technology
- Culture & recreation
- Life & arts
- Science
- Professional
- Business
- API
- Data
- Blog
- Facebook
- Twitter
- LinkedIn
- Instagram