



Products

# Rust's lazy\_static implies objects cannot ever be deleted?

[Log in](#) [Sign up](#)

[Ask Question](#)

Asked 1 year, 6 months ago

Active 1 year, 6 months ago

Viewed 194 times



0



Rust's lazy\_static crate enables us to create static objects in Rust:

```
lazy_static! {
    static ref HASHMAP: HashMap<u32, &'static str> = {
        let mut m = HashMap::new();
        m.insert(0, "foo");
        m.insert(1, "bar");
        m.insert(2, "baz");
    };
}
```

but as you see, the HashMap must store static strings as well. Does that mean that everytime I add something to this hashmap, it can never be deleted, since it's static?

[rust](#)

Share

Improve this

question

Follow

asked Jun 19 '20 at 4:35



[Querlando OCs](#)

887 1 20 61

It's a global variable, when do you expect it would be deleted?

– [mcarton](#)

Jun 19 '20 at 8:34

[Add a comment](#)

1 Answer

[Active](#) [Oldest](#) [Votes](#)



1



Any string that you can reference with a &'static str must live until the program terminates. So yes, the way that you have it set up, you would never be able to delete any of the string data that is put into the HashMap. Moreover, outside of the lazy\_static block you will also never be able to add anything to the HashMap. This is because outside the lazy\_static block you will only be able to obtain a shared reference to the HashMap, and you cannot perform mutation on a HashMap with only a shared reference (i.e., a &HashMap).

If you want to be able to modify the HashMap, you can do so by wrapping it in a Mutex and using owned strings instead of &'static str:

```
#[macro_use]
extern crate lazy_static;

use std::collections::HashMap;
use std::sync::Mutex;

lazy_static! {
    static ref HASHMAP: Mutex<HashMap<u32, String>> = {
        let mut m = HashMap::new();
        m.insert(0, "foo".to_string());
        m.insert(1, "bar".to_string());
        m.insert(2, "baz".to_string());
    };
}

fn main() {
    println!("{}", HASHMAP.lock().unwrap().iter());
    HASHMAP.lock().unwrap().remove(&1);
    println!("{}", HASHMAP.lock().unwrap().iter());
}
```

**Your privacy**

By clicking "Accept all cookies", you agree Stack Exchange can store cookies on your device and disclose information in accordance with our [Cookie Policy](#).

[Accept all cookies](#) [This gives an output like:](#) [CUSTOMIZE settings](#)

```
[(0, "foo"), (1, "bar"), (2, "baz")]
[(0, "foo"), (2, "baz")]
```

However, it might be worth asking whether a static object is really what you need. In Rust, typically it is more idiomatic (and simpler and better performing) to use local variables instead of statics.

[Share](#)

[Improve this answer](#)

Follow

answered Jun 19 '20 at 5:18



[Brent Kerby](#)

1,278 • 8 • 12

---

Note that if there are no threads, you can use a [RefCell](#) instead of the `Mutex` to mutate the value.

– [Jmb](#)

Jun 19 '20 at 6:25

Unfortunately, `RefCell` won't work in a `lazy_static` because it requires its items to be `Sync`.

– [Brent Kerby](#)

Jun 19 '20 at 6:57

You could, however, put a `RefCell` inside a `thread_local` block instead of `lazy_static`.

– [Brent Kerby](#)

Jun 19 '20 at 7:47

[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](#) or [ask your own question](#).

#### The Overflow Blog

- Sequencing your DNA with a USB dongle and open source code
- Don't push that button: Exploring the software that flies SpaceX rockets and...

#### Featured on Meta

- Providing a JavaScript API for userscripts
- Congratulations to the 59 sites that just left Beta

Related

[What are the differences between Rust's 'String' and 'str'?](#)

[What are Rust's exact auto-dereferencing rules?](#)

[Factory method: instance does not live long enough](#)

[How can I implement Rust's Copy trait?](#)

[What does Rust's unary || \(parallel pipe\) mean?](#)






[Need holistic explanation about Rust's cell and reference counted types](#)

[Why is Rust's assert\\_eq! implemented using a match?](#)

[How does Rust's 128-bit integer 'i128' work on a 64-bit system?](#)

How do I avoid "cannot move out of static item" when using a lazy\_static variable?

## Hot Network Questions

-  Question on connections in general relativity and particle physics
-  Have there been no deaths due to omicron in Africa?
-  Regular expressions within QGIS expressions: logical operator AND
-  Is it possible to convert a taproot address into a native segwit address?
-  Calculating mean follow up from ONLY sample size and range

[more hot questions](#)

 Question feed

## STACK OVERFLOW

[Questions](#)  
[Jobs](#)  
[Developer Jobs Directory](#)  
[Salary Calculator](#)  
[Help](#)  
[Mobile](#)

## PRODUCTS

[Teams](#)  
[Talent](#)  
[Advertising](#)  
[Enterprise](#)

## 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](#)

site design / logo © 2021 Stack Exchange Inc; user contributions licensed under [cc by-sa](#). rev 2021.12.22.41046