Struct ownership

Asked 1 year, 2 months ago

Active 1 year, 2 months ago

Viewed 357 times

1

1

```
struct Haha {
    pub a: u32,
    pub b: Vec<u32>,
}


let example = Haha {
    a: 32,
    b: vec![1],
};
let new_a = example.a;
let new_b = example.b;
```

My understanding is:

1. `new_a` is a copy of `example.a` so `example` still owns `example.a` .
2. `new_b` now owns `example.b` since `example.b` was moved.

Does rust implicitly copy `example.a` because it has `Copy` trait? And since `example.b` , which is a `Vec` , does not implement `Copy` trait, ownership of `example.b` is moved rather than copied?

rust ownership

Share
Improve this question
Follow

edited Oct 17 '20 at 3:10

Herohtar
4,788 ● 3 ● 28 ● 36

asked Oct 16 '20 at 14:24

pandawithcat
233 ● 1 ● 6

Check out this blog post medium.com/@bugaevc/… "Fortunately, Rust has the Copy trait. Types that implement it (all the primitive ones do) use copy semantics when assigning, all the other types use move semantics."
– ccheneson
Oct 16 '20 at 14:57

Add a comment

1 Answer

Active Oldest Votes

4

Your understanding is correct. `a` is copied while `b` is moved. You can confirm this by trying to access the two fields afterwards.

```
println!("{:?}", example.a);
```

This prints `32` . `example.a` is still accessible because it was copied, not moved.

```
println!("{:?}", example.b);
```

Accessing `example.b` fails to compile with the error message:

```
error[E0382]: borrow of moved value: `example.b`
  --> src/main.rs:13:22
   |
12 |     let _new_b = example.b;
   |                  --------- value moved here
13 |     println!("{:?}", example.b);
   |                      ^^^^^^^^^ value borrowed here after move
   |
   = note: move occurs because `example.b` has type `std::vec::Vec<u32>`, which does not implement the `Copy` trait
```

Which confirms exactly what you said, that `example.b` was moved because it doesn't implement the `Copy` trait.

Share
Improve this answer
Follow

answered Oct 16 '20 at 14:34

John Kugelman
321k ● 66 ● 492 ● 542

---

thanks for a quick response! So for rust newbies like me(who might not know vec does not implement copy trait), do we have to rely on the compiler to check ownership since copy is done implcitily?
– pandawithcat
Oct 16 '20 at 14:39

@LouisLee you can see in the API documentation whether a type is `Copy` or not.
– Jesper
Oct 16 '20 at 14:43

@LouisLee If a variable's size is known at compile time (for example i32), it can be stored in the stack, thus it is fast to copy. If a variable's size it not known at compile time (for example vectors), it will be stored in the heap and a pointer to the memory location in the heap is stored in the stack. Rust doesn't duplicate memory in the heap because it's slow and inefficient. You can assume that if a variable is using the heap, it won't implement the copy trait.
– Bruno Robert
Oct 16 '20 at 16:48

i have one more question. I thought structs and enums were saved in stack by default unless some fields have to be allocated in the heap. So if i do this ``` struct haha{ pub a: i32, pub b: inner } struct inner{ pub c: u32 } ``` although I didn't derive copy or clone trait for struct inner, i was expecting struct inner to be stack allocated but the compiler tells me it isn't. So are struct's stored in the heap unless they derive copy trait?
– pandawithcat
Oct 17 '20 at 2:52

@LouisLee No, Rust does not automatically decide whether something is allocated on the stack vs the heap. By default things are allocated on the stack. Some structs such as `Vec` are implemented in such a way that they allocate things on the heap. Just making something `Copy` or not does not change whether it's allocated on the stack or heap.
– Jesper
Oct 17 '20 at 7:44

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 ownership 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

Is it possible to make a type only movable and not copyable?

Ownership and conditionally executed code

Is it possible for one struct to extend an existing struct, keeping all the fields?

How does Rust move stack variables that are not Copyable?

How to take ownership of a C pointer in Rust and drop it appropriately?

Using Trait Object for Struct - ERROR: wrong number of type arguments

3
Can we implement the Copy and Clone traits to Command struct?

Hot Network Questions

EU ETS: if within-EU flight emissions are limited to climate goals, is there still a reason not to fly as long as you can afford it?

Alternatives to replace tandem single pole MWBC breakers?

Applied mathematics or Computer science PhD?

Why did the JWST solar array deploy early?

Dataframe from a character vector where variable name and its data were stored jointly

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