Does println! borrow or own the variable?

Asked 6 years, 7 months ago
Active 1 year, 6 months ago
Viewed 5k times

65

10

I am confused with borrowing and ownership. In the Rust documentation about reference and borrowing

```
let mut x = 5;
{
  let y = &mut x;
  *y += 1;
}
println!("{}", x);
```

They say

> `println!` can borrow `x`.

I am confused by this. If `println!` borrows `x`, why does it pass `x` not `&x`?

I try to run this code below

```
fn main() {
  let mut x = 5;
  {
    let y = &mut x;
    *y += 1;
  }
  println!("{}", &x);
}
```

This code is identical with the code above except I pass `&x` to `println!`. It prints '6' to the console which is correct and is the same result as the first code.

rust ownership

Share
Improve this question
Follow
edited Oct 14 '18 at 19:36
Kim Kern
39.3k 14 143 158

asked May 26 '15 at 5:48
kevinyu
1,087 9 12

Add a comment

1 Answer

Active Oldest Votes

81

The macros `print!`, `println!`, `eprint!`, `eprintln!`, `write!`, `writeln!` and `format!` are a special case and implicitly take a reference to any arguments to be formatted.

These macros do not behave as normal functions and macros do for reasons of convenience; the fact that they take references silently is part of that difference.

```
fn main() {
  let x = 5;
  println!("{}", x);
}
```

Run it through `rustc -Z unstable-options --pretty expanded` on the nightly compiler and we can see what `println!` expands to:

```
#![feature(prelude_import)]
#[prelude_import]
use std::prelude::v1::*;
#[macro_use]
extern crate std;
fn main() {
    let x = 5;
    {
        ::std::io::_print(::core::fmt::Arguments::new_v1(
            &["", "\n"],
            &match (&x,) {
                (arg0,) => [::core::fmt::ArgumentV1::new(
                    arg0,
                    ::core::fmt::Display::fmt,
                )],
            },
        ));
    };
}
```

Tidied further, it's this:

```
use std::{fmt, io};

fn main() {
    let x = 5;
    io::_print(fmt::Arguments::new_v1(
        &["", "\n"],
        &[fmt::ArgumentV1::new(&x, fmt::Display::fmt)],
        //                  ^^
    ));
}
```

Note the `&x`.

If you write `println!("{}", &x)`, you are then dealing with two levels of references; this has the same result because there is an implementation of std::fmt::Display for `&T` where `T` implements `Display` (shown as `impl<'a, T> Display for &'a T where T: Display + ?Sized`) which just passes it through. You could just as well write `&&&&&&&&&&&&&&&&&&&&&&&&x`.

Share
Improve this answer
Follow
edited Jun 3 '20 at 18:37

Shepmaster
305k • 59 • 824 • 1083

answered May 26 '15 at 6:47

Chris Morgan
76.8k • 20 • 194 • 203

2

I don't understand why you'd call those macros a "special case". This kind of implicit reference-passing can be implemented for any macro.
– Markus Unterwaditzer
Nov 29 '15 at 13:06

13

@MarkusUnterwaditzer: Sure, but the thing is that it looks normal but isn't. And sure, other macros can make themselves special cases too. The fact is that it's strongly advised against in general.
– Chris Morgan
Dec 3 '15 at 12:55

4

Maybe this could be pointed out in the book? Got me confused as well.
– mauleros
Oct 16 '20 at 2:02

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

Linked

1
how does rust's println deal with arguments' ownerships?

0
Forwarding Parameter References to Functions in Rust

Why does the println! function use an exclamation mark in Rust?

Why does the Rust compiler allow index out of bounds?

What is the usage of the asterisk symbol in Rust?

What benefits are there with making println a macro?

Why is the size of a pointer to something on the heap larger than the size of a stack variable?

How does println! interact with multiple levels of indirection?

How does the stack handle popping values off in a different order than they are created?

Why I can't return fmt::Arguments<'a> from &'a T?

See more linked questions

Related

"the immutable borrow prevents mutable borrows" when pumping events with rust-sdl2

What are move semantics in Rust?

What is the difference between iter and into_iter?

Borrow errors for multiple borrows

Why doesn't the lifetime of a mutable borrow end when the function call is complete?

Why does the closure take ownership of the vector here?

Why does the Rust compiler not optimize code assuming that two mutable references cannot alias?

Borrow a hashmap without Borrowing his content

Hot Network Questions

- Why did the JWST solar array deploy early?
- Changing a color of a link
- What was the plutonium for, that was stolen at the start of The Amazing Spider-Man 2?
- Is there a deterministic guide to landing?
- What did John the Baptist say about Jesus that resulted in many Jews in Jesus' time to put faith in him? John 10:41-42

more hot questions

Question feed