Why is "&&" being used in closure arguments?

Asked 4 years, 7 months ago

Active 2 years, 5 months ago

Viewed 2k times

18

5

I have two questions regarding this example:

```
let a = [1, 2, 3];

assert_eq!(a.iter().find(|&&x| x == 2), Some(&2));
assert_eq!(a.iter().find(|&&x| x == 5), None);
```

1. Why is `&&x` used in the closure arguments rather than just `x`? I understand that `&` is passing a reference to an object, but what does using it twice mean?

   I don't understand what the documentation says:

   > Because `find()` takes a reference, and many iterators iterate over references, this leads to a possibly confusing situation where the argument is a double reference. You can see this effect in the examples below, with `&&x`.

2. Why is `Some(&2)` used rather than `Some(2)`?

reference rust borrowing

Share

Improve this question

Follow

edited Jul 10 '19 at 14:04

Shepmaster

305k ● 59 ● 824 ● 1083

asked May 7 '17 at 5:03

Sajuuk

2,001 ● 3 ● 13 ● 31

Add a comment

2 Answers

Active Oldest Votes

24

✓

`a` is of type `[i32; 3]`; an array of three `i32` s. `[i32; 3]` does not implement an `iter` method, but it *does* dereference into `&[i32]`. `&[i32]` implements an `iter` method which produces an iterator. This iterator implements `Iterator<Item=&i32>`.

It uses `&i32` rather than `i32` because the iterator has to work on arrays of *any* type, and not all types can be safely copied. So rather than restrict itself to copyable types, it iterates over the elements by reference rather than by value.

`find` is a method defined for all `Iterator` s. It lets you look at each element and return the one that matches the predicate. Problem: if the iterator produces non-copyable values, then passing the value into the predicate would make it impossible to return it from `find`. The value cannot be re-generated, since iterators are not (in general) rewindable or restartable. Thus, `find` has to pass the element to the predicate by-reference rather than by-value.

So, if you have an iterator that implements `Iterator<Item=T>`, then `Iterator::find` requires a predicate that takes a `&T` and returns a `bool`. `[i32]::iter` produces an iterator that implements `Iterator<Item=&i32>`. Thus, `Iterator::find` called on an array iterator requires a predicate that takes a `&&i32`. That is, it passes the predicate a pointer to a pointer to the element in question.

So if you were to write `a.iter().find(|x| ..)`, the type of `x` would be `&&i32`. This cannot be directly compared to the literal `i32` value `2`. There are several ways of fixing this. One is to explicitly dereference `x`: `a.iter().find(|x| **x == 2)`. The other is to use pattern matching to destructure the double reference: `a.iter().find(|&&x| x == 2)`. These two approaches are, in this case, doing *exactly* the same thing. [1]

As for why `Some(&2)` is used: because `a.iter()` is an iterator over `&i32`, *not* an iterator of `i32`. If you look at the documentation for `Iterator::find`, you'll see that for `Iterator<Item=T>`, it returns an `Option<T>`. Hence, in this case, it returns an `Option<&i32>`, so that's what you need to compare it against.

[1]: The differences only matter when you're talking about non-`Copy` types. For example, `|&&x| ..` wouldn't work on a `&&String`, because you'd have to be able to move the `String` out from behind the reference, and that's not allowed. However, `|x| **x ..` *would* work, because that is just reaching inside the reference without moving anything.

Share

Improve this answer

Follow

answered May 7 '17 at 6:05

DK.

47.4k ● 3 ● 150 ● 142

2

thanks! Additionally, in case if anyone doesn't understand what is ***"dereference into &[i32] "***, an array `a` can be automatically borrowed as a slice `&a` according to [this](),
– Sajuuk
May 7 '17 at 7:10

Add a comment

▲

4

▼

1) I thought the book explanation was good, maybe my example with `.cloned()` below will be useful. But since `.iter()` iterates over references, you have to specify reference additionally because `find` expects a reference.

2) `.iter()` is iterating over references; therefore, you find a reference.

You could use `.cloned()` to see what it would look like if you didn't have to do deal with references:

```
assert_eq!(a.iter().cloned().find(|&x| x == 2), Some(2));
```

Share
Improve this answer
Follow

answered May 7 '17 at 5:50

Akavall

72.9k ● 43 ● 190 ● 235

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 reference rust borrowing 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
Why does `find` take a parameter by reference and then dereference it?

0
Reference to reference in rust

What is an idiomatic way to collect an iterator of &T into a collection of Ts?

Is Vec<&&str> the same as Vec<&Str>?

Can't compare `&Thing` with `Thing`

How to use a vector of string slices

0
References to references usage

Related

Why is 'this' a pointer and not a reference?

List changes unexpectedly after assignment. Why is this and how can I prevent it?

When to use references vs. pointers

Why does the argument for the find closure need two ampersands?

3
When to use Box instead of reference?

Why is it discouraged to accept a reference to a String (&String), Vec (&Vec), or Box (&Box) as a function argument?

0
Return and consume an iterator of mutable references from a closure

Mutating fields of Rc Refcell depending on its other internal fields

Hot Network Questions

Is it possible to convert a taproot address into a native segwit address?

How am I able to simulate gears on my single speed?

Split polyline to equal parts using QGIS

Numbers, Racked Up

Company kept previous personal phone number

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