Ownership: differences between tuples and arrays in Rust

Asked 1 year, 6 months ago
Active 1 year, 6 months ago
Viewed 330 times

7

I am starting to learn Rust, and while experimenting, I have found a difference in how ownership is applied to tuples and arrays I do not understand. Basically, the following code shows the difference:

```
#![allow(unused_variables)]

struct Inner {
    in_a: u8,
    in_b: u8
}
struct Outer1 {
    a: [Inner; 2]
}

struct Outer2 {
    a: (Inner, Inner)
}

fn test_ownership(num: &mut u8, inner: &Inner) {
}

fn main() {
    let mut out1 = Outer1 {
        a: [Inner {in_a: 1, in_b: 2}, Inner {in_a: 3, in_b: 4}]
    };
    let mut out2 = Outer2 {
        a: (Inner {in_a: 1, in_b: 2}, Inner {in_a: 3, in_b: 4})
    };

    // This fails to compile
    test_ownership(&mut out1.a[0].in_a, &out1.a[1]);
    // But this works!
    test_ownership(&mut out2.a.0.in_a, &out2.a.1);
}
```

The first invocation of `test_ownership()` does not compile, as expected Rust emits an error complaining about taking both a mutable and immutable reference to `out1.a[_]`.

```
error[E0502]: cannot borrow `out1.a[_]` as immutable because it is also borrowed as mutable
  --> src/main.rs:27:41
   |
27 |     test_ownership(&mut out1.a[0].in_a, &out1.a[1]);
   |     -------------- ------------------   ^^^^^^^^^^ immutable borrow occurs here
   |     |              |
   |     |              mutable borrow occurs here
   |     mutable borrow later used by call
```

But the thing I do not understand, is why the second invocation of `test_ownership()` does not make the borrow checker go nuts? It seems as if arrays are considered as a whole independently of the indexes being accessed, but tuples allow multiple mutable references to their different indexes.

rust

Share
Improve this question
Follow

edited Jun 21 '20 at 10:44

asked Jun 20 '20 at 21:22

doragasu
115 • 7

Add a comment

3 Answers

Active Oldest Votes

5

Tuples are like anonymous structs, and accessing an element in a tuple behaves like accessing a **struct field**.

Structs can be partially borrowed (and also partially moved), so `&mut out2.a.0.in_a` borrows only the first field of the tuple.

The same does not apply to indexing. The indexing operator can be overloaded by implementing `Index` and `IndexMut`, so `&mut out1.a[0].in_a` is equivalent to `&mut out1.a.index_mut(0).in_a`. While `a.0` just accesses a field, `a[0]` calls a function! Functions can't partially borrow something, so the indexing operator must borrow the entire array.

Share
Improve this answer
Follow
answered Jun 21 '20 at 1:57

Aloso
4,380 ● 4 ● 21 ● 36

Add a comment

▲
2
▼

That is indeed an interesting case. For the second case it works because compiler understands that different part of the structure is borrowed (there's a section in nomicon for that). For the first case compiler is, unfortunately is not that smart (indexing is generally performed by a runtime calculated value), so you need to destruct it manually:

```
let [mut x, y] = out1.a;
test_ownership(&mut x.in_a, &y);
```

Share
Improve this answer
Follow
answered Jun 20 '20 at 21:32

Kitsu
2,506 ● 8 ● 22

Interesting. Destructuring the array as you wrote, requires moving `out1.a`, so if later I wanted to use it, that would not work, right? Could it be possible doing something similar using references? I suspect it might not be possible without going the unsafe route.
– doragasu
Jun 20 '20 at 21:56

2

@doragasu You can use `split_first_mut` or another one of the split functions to do this safely on references on arrays.
– mcarton
Jun 20 '20 at 22:11

1

As for the array case, I'd rather prefer bind-by-ref: `let [ref mut x, ref y] = out1.a;`, because splitting might be tedious for array sizes >2.
– Kitsu
Jun 21 '20 at 6:29

@Kitsu: Tested as you wrote with the `ref` keyword, and it works nice and easy. Got to give the `split_first_mut` a try also.
– doragasu
Jun 21 '20 at 10:43

Add a comment

▲
1
▼

The difference between the tuple case and the indexing case is what is being used to perform the indexing. In the tuple case we are using syntax sugar over what is effectively an identifier into a struct. As it is an identifier, which field is accessed must be static. As such there is a simple set of rules the borrow checker can follow to determine lifetimes, which follows the same rules used for struct fields.

In the case of an indexing, this is in stable rust an inherently dynamic operation. The index is not an identifier, but an expression. Since the stable typesystem does not yet have a concept of constant expressions or type level values, the borrow checker always treats the index as if it dynamic, even in trivial cases such as yours which are are clearly static. Since they are dynamic, it can't prove non-equality of the indexes, so the borrow conflicts.

Share
Improve this answer
Follow
answered Jun 20 '20 at 22:39

user1937198
4,641 ● 4 ● 18 ● 31

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`?

Multiple mutable borrows when generating a tree structure with a recursive function in Rust

6
cannot move out of borrowed content when unwrapping a member variable in a &mut self method

Why does HashMap have iter_mut() but HashSet doesn't?

Rust iterators and looking forward (peek/multipeek)

Object Orientated Rust (The rust book chapter 17 blog)

0
Rust: How to pass a mutable as immutable to a function

1
Prevent cannot borrow `*self` as immutable because it is also borrowed as mutable when accessing disjoint fields in struct?

Hot Network Questions

- Vizier of the Menagerie and cost reducing
- Are they already planning a successor to the JWST?
- Regular expressions within QGIS expressions: logical operator AND
- Is it common practice to apply identical processing effects to a batch of photos?
- 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?

more hot questions

Question feed