Why does this lifetime not "expire"?

Asked 1 year, 3 months ago
Active 1 year, 3 months ago
Viewed 106 times

3

For the following block, when do the lifetimes 'b and 'c end?

```
use core::ops::Deref;

#[derive(Debug)]
struct A<'b, 'c, T> {
  child_b: &'b T,
  child_c: &'c T
}

impl<'b, 'c, T> A<'b, 'c, T> {
  pub fn new_wrapper(b_value: &'b T, c_value: &'c T) -> A<'b, 'c, T> {
    A {
      child_b: b_value,
      child_c: c_value
    }
  }
}

fn doesnt_drop_borrow<T: Copy>(ty: &T) {
  *ty;
}

fn drop<T>(ty: T) { }

fn main() {
  let b: String = "wonderful".into();
  let c: String = "lifetime".into();
  let a = A::new_wrapper(&b, &c);
  println!("hello this {:?} world", &a);
  doesnt_drop_borrow(&a.child_c);
  drop(a.child_c);
  println!("hello this {:?} world", &a);
}
```

generics rust lifetime

Share
Improve this question
Follow
asked Sep 7 '20 at 22:58
David Golembiowski
103 1 14

Add a comment

1 Answer

Active Oldest Votes

4

Since `a.child_c` has the type `&String`, the `ty` parameter in `doesnt_drop_borrow(&a.child_c)` has type `&&String`, and `*ty` has type `&String` so the original `String` (`c`) won't be dropped.

I don't know what precisely motivates your question but I guess it's the fact that you can still use `a` after calling `drop(a.child_c)`. The name of this function is misleading because the `ty` parameter here has type `&String` (the same as `a.child_c`). So the `ty` parameter appears as a new immutable borrow of the original `c`, then is immediately dropped, but the original `c` is not dropped. When a reference is dropped the referred to value is not.

None of these two functions actually move the original `String`, they only deal with references.

So `b` and `c` here live till the end of `main()`, and so does `a` which contains references to them.

Share
Improve this answer
Follow
answered Sep 7 '20 at 23:44
prog-fh

8,988 ● 1 ● 5 ● 24

3

It's also worth mentioning that `&T` is `Copy`, so `drop(a.child_c)` drops a copy of `a.child_c` (i.e. `a.child_c` is not moved).
– Francis Gagné
Sep 8 '20 at 0:08

1

@FrancisGagné: There is even a clippy lint (`drop_copy`) for that.
– rodrigo
Sep 8 '20 at 10:22

1

Well, in this particular case it would be `drop_ref`, I think.
– rodrigo
Sep 8 '20 at 10:28

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 generics rust lifetime 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 is the lifetime of a static variable in a C++ function?

Why does C# forbid generic attribute types?

Is List<Dog> a subclass of List<Animal>? Why are Java generics not implicitly polymorphic?

743
What is a raw type and why shouldn't we use it?

Why are explicit lifetimes needed in Rust?

Why can't I store a value and a reference to that value in the same struct?

Lifetime constraints to model scoped garbage collection

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

Hot Network Questions

Split polyline to equal parts using QGIS

Using a friend to move cash into my checking account

Who (or what) created the atropal?

Why does the prophecy imply Macbeth has to murder the king?

Vizier of the Menagerie and cost reducing

more hot questions

Question feed