How can I conditionally provide a default reference without performing unnecessary computation when it isn't used?

Asked 2 years, 2 months ago
Active 10 months ago
Viewed 252 times

3

I have some variable passed into my function by reference. I don't need to mutate it or transfer ownership, I just look at its contents. If the contents are in some state, I want to replace the value with a default value.

For instance, my function accepts a `&Vec<String>` and if the `Vec` is empty, replace it with `vec!["empty"]`:

```
fn accept(mut vec: &Vec<String>) {
    if vec.len() == 0 {
        vec = &vec!["empty".to_string()];
    }
    // ... do something with `vec`, like looping over it
}
```

But this gives the error:

```
error[E0716]: temporary value dropped while borrowed
 --> src/lib.rs:3:16
  |
1 | fn accept(mut vec: &Vec<String>) {
  |                    - let's call the lifetime of this reference `'1`
2 |     if vec.len() == 0 {
3 |         vec = &vec!["empty".to_string()];
  |         -------^^^^^^^^^^^^^^^^^^^^^^^^^- temporary value is freed at the end of this statement
  |         |      |
  |         |      creates a temporary which is freed while still in use
  |         assignment requires that borrow lasts for `'1`
```

Preventing the `mut` results in the same error as the previous example:

```
fn accept(input: &Vec<String>) {
    let vec = if input.len() == 0 {
        &vec!["empty".to_string()]
    } else {
        input
    };
    // ... do something with `vec`, like looping over it
}
```

The only solution I've come up with is to extract the default value outside the `if` and reference the value:

```
fn accept(input: &Vec<String>) {
    let default = vec!["empty".to_string()];
    let vec = if input.len() == 0 {
        &default
    } else {
        input
    };
    // ... do something with `vec`
}
```

This results in less clean code and also *unnecessarily doing that computation*.

I know and understand the error... you're borrowing the default value inside the body of the `if`, but that value you're borrowing from doesn't exist outside the `if`. That's not my question.

Is there any cleaner way to write out this pattern?

I don't believe this is a duplicate of Is there any way to return a reference to a variable created in a function? because I have a reference I'd like to use *first* if possible. I don't want to dereference the reference or `clone()` it because *that would perform unnecessary computation*.

Can I store either a value or a reference in a variable at the same time?

rust reference conditional-statements default-value

edited Feb 16 at 17:43

Shepmaster
305k • 59 • 824 • 1083

asked Oct 25 '19 at 13:56

Chris Smith
2,722 • 3 • 21 • 51

You're trying to paper over a misunderstanding either in your implementation or in your knowledge of Rust. If you have a method that requires pre-processing of an argument, you should be splitting this up into two calls, and inline the pre-processor, for example. This leaves your "inner" code sane, and leaves the data sanitization to the outside call.
– Sébastien Renauld
Oct 25 '19 at 14:00

1

You need to add an appropriately scoped local variable to own the `Vec` for as long as the reference lives, as in How do I make format! return a &str from a conditional expression? This is what the compiler is trying to suggest by `consider using a `let` binding to create a longer lived value`. Here's a working version. Does this help?
– trent formerly cl
Oct 25 '19 at 14:15

@SébastienRenauld Ok, well in reality this "function" is a match arm part of a much larger algorithm. Only this arm does this type of sanitization to the outer variables. Still, though. What if the "outer" code was also only provided a reference? You'd still have the same issue. I don't feel it's right to be propagating these values up the call stack to their actual creation when the condition may be dependant on local state.
– Chris Smith
Oct 25 '19 at 14:15

3

I imagine the `&Vec` was only for the purpose of illustration, but if it's in your actual code, perhaps you should also read Why is it discouraged to accept a reference to a String (&String), Vec (&Vec), or Box (&Box) as a function argument?
– trent formerly cl
Oct 25 '19 at 14:17

Add a comment

2 Answers

Active Oldest Votes

7

You don't have to create the default vector if you don't use it. You just have to ensure the declaration is done outside the `if` block.

```
fn accept(input: &Vec<String>) {
    let def;
    let vec = if input.is_empty() {
        def = vec!["empty".to_string()];
        &def
    } else {
        input
    };
    // ... do something with `vec`
}
```

Note that you don't have to build a new default vector every time you receive an empty one. You can create it the first time this happens using lazy_static or once_cell:

```
#[macro_use]
extern crate lazy_static;

fn accept(input: &[String]) {
    let vec = if input.is_empty() {
        lazy_static! {
            static ref DEFAULT: Vec<String> = vec!["empty".to_string()];
        }
        &DEFAULT
    } else {
        input
    };
    // use vec
}
```

Share
Improve this answer
Follow
edited Dec 19 '19 at 9:48

answered Oct 25 '19 at 14:02

Denys Séguret
348k ● 78 ● 737 ● 714

Add a comment

0

It sounds like you may be looking for `std::borrow::Cow`, depending on how you're going to use it.

Share
Improve this answer
Follow
edited Feb 16 at 17:34

Shepmaster
305k ● 59 ● 824 ● 1083

answered Oct 26 '19 at 5:28

Daniel Wagner-Hall
1,991 ● 1 ● 16 ● 15

Add a comment

Your Answer

Post Your Answer

Not the answer you're looking for? Browse other questions tagged rust reference conditional-statements default-value or ask your own question.

Linked

Conditionally update argument to function in Rust

1
Is there a way to not free temporary variables created in conditional branches and only destroy them with the stack frame?

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

85
Is there any way to return a reference to a variable created in a function?

Are polymorphic variables allowed?

How do I make format! return a &str from a conditional expression?

4
Does a trait exist for generic code for references and values without having to write it all out twice?

How can I use polymorphism with the Deref trait to have a a single object that can be represented by either a Transaction or Connection?

Related

Default values and initialization in Java

How do I return a reference to something inside a RefCell without breaking encapsulation?

How can I get user input without receiving an "Unsed Variable" warning?

Trying to borrow variable binding from outside of loop

Method not compatible with trait with confusing error message

Iterating through a recursive structure using mutable references and returning the last valid reference

1
Shadowing in Rust and Fighting the Borrow Checker

How do I deserialize a configuration value from an environment variable as a Vec?

How do I read a String from a File, split it, and create a Vec<&str> in one statement?

Why am I getting recursion when trying to implement trait for all reference types

Hot Network Questions

- Given many questions as to whether Jesus was born on 25 December or not, I ask if the ambiguity in scripture is meant to teach us something?
- Calculating mean follow up from ONLY sample size and range
- What was the plutonium for, that was stolen at the start of The Amazing Spider-Man 2?
- Mine sweeping game for the terminal
- Tax implications of large gift

more hot questions

Question feed