Returning Parts of a Mutable Borrow Triggers the Borrow Checker

help

neurolag July 17, 2025, 5:18pm 1

Hey all!
I'm currently working on an `swc` plugin which replaces parts of a JavaScript DOM.
I came to realize that working with `swc`s complex data structures involving nested `Box`es and `enum`s which must be `match`ed against is quite cumbersome when it comes to borrow checking.

This is an excert of my method which parses a node and, in case it's not a `nameof` expression, returns `None` and otherwise returns an `Err` (possibly holding immutable parts of the node) or an `Ok` (holding mutable parts of the node):

Rust Playground

Sadly, I can't find myself able to find a way to work around the error reported in this code snippet.
Could you help me understand what I'm doing wrong or how I can improve this piece of code?

Best regards

jumpnbrownweasel July 17, 2025, 5:42pm 2

I didn't try to identify the underlying cause or to come up with a clever solution. I often find that error types with lifetimes are a cause of borrowing problems because they're propagated upwards, and since they are infrequently created (often but not always), I sometimes resort to cloning their contents so I can remove the lifetime annotation. That worked in this case.
playground

2 Likes

neurolag July 17, 2025, 6:10pm 3

Oooh amazing, thank you so much for your rapid answer!
So, with this being a clone of a mere reference, this doesn't raise the memory consumption considerably, right?

quinedot July 17, 2025, 6:32pm 4

If you don't want to change to non-borrowing error types, the OP error is due to a conditional return of a borrow situation, which Polonius should eventually accept. This crate may provide a stop-gap (I didn't try it).

2 Likes

jumpnbrownweasel July 17, 2025, 6:39pm 5

Cloning the String (in the Atom) does allocate a new String. That's why I mentioned that I normally only use this approach when the error is infrequently created; if infrequent, allocations don't matter nearly as much.

1 Like

SReichelt July 17, 2025, 7:02pm 6

You can work around the problem by performing the entire matching twice, like this: Rust Playground

I would hope that the compiler can actually optimize away the second call, but I haven't checked, and it's only the error path anyway.

2 Likes

neurolag July 20, 2025, 7:21pm 7

Thank you so much - this actually was a really good read and helped me get my head halfway wrapped around what this error is and why it happens. So... If I understand correctly, references created in `if let` and `match` calls and then featured in `return`s live to the end of the function for all code paths.

I addressed this issue by using the same `call` object for all `return`s in the function:

```
impl NameofVisitor {
   fn get_nameof_expression<'a>(
       &self,
       node: &'a mut Expr,
   ) -> Option<NameofResult<'a, NameofExpression<'a>>> {
       match node {
           Expr::Call(call) => match call {
               CallExpr {
                   callee: Callee::Expr(callee),
                   ..
```

```
                } => Some(Ok(NameofExpression::Normal {
                    method: match &**callee {
                        Expr::Ident(ident) if self.is_global_nameof(ident) => None,
                        Expr::Member(MemberExpr {
                            obj,
                            prop: MemberProp::Ident(prop_name),
                            ..
                        }) => match &**obj {
                            Expr::Ident(ident) if self.is_global_nameof(ident) => {
                                match prop_name.sym.as_str() {
                                    "full" => Some(NameofMethod::Full),
                                    "interpolate" => Some(NameofMethod::Interpolate),
                                    "array" => Some(NameofMethod::Array),
                                    "split" => Some(NameofMethod::Split),
                                    _ => {
                                        return Some(Err(NameofError::InvalidMethod(
                                            call.callee
                                                .as_expr()
                                                .unwrap()
                                                .as_member()
                                                .unwrap()
                                                .prop
                                                .as_ident()
                                                .unwrap(),
                                        )))
                                    }
                                }
                            }
                            _ => return None,
                        },
                        _ => return None,
                    },
                    call,
                })),
                _ => None,
            },
            _ => None,
        }
    }
}
```

This seems to be working out great!

Thank you so much for all you guy's help!

1 Like

system Closed October 18, 2025, 7:22pm 8

This topic was automatically closed 90 days after the last reply. We invite you to open a new topic if you have further questions or comments.

Related topics

Topic		Replies	Views	Activity
Different borrowing in match variants help	11	604	April 1, 2021	
Why does this fail borrow check? help	4	990	January 12, 2023	
Delay mutable borrows to destructured in if let and match expressions without boilerplate help	6	187	August 17, 2024	
How to finish borrow inside a match arm? help	4	1064	August 8, 2022	

Which borrow checker issue is this? Can't perform immutable borrow in none branch of optional mutable reference match help	3	527	December 23, 2021