Struggling with borrows

help

frosklis January 17, 2021, 12:09am 1

Newbie question. I think I understand why the code below fails. I have a function that gets a Vec of items (not a ref, the vector itself) and I'd like to do some things based on what is in the vector.
It is complaining because it returns something that references items, which is borrowed in the for loop.

How do I get "around" that?

```
impl TryFrom<Vec<Item>> for Ledger<'_> {
   type Error = Error;

   fn try_from(items: Vec<Item>) -> Result<Self, Self::Error> {
       let mut ledger = Ledger::new();

       // 1. Populate the lists
       for item in items.iter() {
           match item {
               Item::Comment(_) => {}
               Item::Transaction(parsed) => {
                   for p in parsed.postings.iter() {
                       let account = Account::from(p.account.clone());
                       ledger.accounts.push(account);

                       // Currencies
                       if let Some(c) = &p.money_currency {
                           ledger.currencies.push(Currency::from(c.as_str()));
                       }
                       if let Some(c) = &p.cost_currency {
                           let currency = Currency::from(c.as_str());
                           ledger.currencies.push(currency);
                       }
                       if let Some(c) = &p.balance_currency {
                           let currency = Currency::from(c.as_str());
                           ledger.currencies.push(currency);
                       }
                   }
               }
               Item::Directive => {}
           }
       }

       return Ok(ledger);
}
```

quinedot January 17, 2021, 1:35am 2

If you want to create a `Ledger` consisting of references into collection of `Items`, the `Items` are naturally going to have to be alive longer than your references. Rust isn't garbage collected, and instead uses a system of ownership to manage memory. To get "around" your situation, you're going to have to approach things in such a way that there's an owner of the `Items` which sticks around for at least as long as any references.

Moreover, self-referential structs are another pattern in Rust which can not be achieved safely and is almost never what you actually want. So for your example, storing the `Vec<Item>` in the `Ledger` isn't likely to work out either.

The two most direct adjustments left are:

Store `Items` in the `Ledger`, not references.

- You'll probably drop the `<'_>` part of `Ledger<'_>` in this approach.

`impl<'a> TryFrom<&'a [Item]> for Ledger<'a>`

- The `Vec` (or whatever owns the `Items`) will have to "stick around" at least as long as the resulting `Ledger`.

2 Likes

system Closed April 17, 2021, 1:35am 3

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
Simple question about borrowing and references help	12	488	January 12, 2023	
Why does the borrow checker raise an error when I keep a reference to an element in a vector that is appended to within a loop? help	26	1749	March 6, 2022	
Struct having a reference to a value in one of its variables help	6	1582	February 24, 2022	
How to correctly write struct that owns bunch of items and list of references for those items	10	1825	January 12, 2023	
Why is it borrowed for too long? help	10	1692	January 12, 2023	