What is the idiomatic way to transform vector of structs to another vector of structs

Ask Question

Asked 1 year, 1 month ago

Active 1 year, 1 month ago

Viewed 56 times

▲

1

▼ 🔖 🕑

My function takes a `Vec` of structs as a parameter and places each of it's element into another vector as a new structure field:

```
fn new(cards: Vec<Card<'t>>) -> Self {
    CardsConnection {
        edges: cards.iter().map(|c| CardsConnectionEdge {
            cursor: c.id.clone(),
            node: *c
        }).collect::<Vec<CardsConnectionEdge>>()
    }
}
```

I really want a move, so only new vector should own the `Card` instances. So since map got a reference, I want do dereference it to move the value, but I got the following error:

```
error[E0507]: cannot move out of `*c` which is behind a shared reference
 --> src/model.rs:74:23
   |
74 |          node: *c
   |                ^^ move occurs because `*c` has type `model::Card<'_>`, which does not implement the `Copy` trait

error: aborting due to previous error; 8 warnings emitted
```

While this non "functional-style" code works fine:

```
fn new(cards: Vec<Card<'t>>) -> Self {
    let mut edges = Vec::<CardsConnectionEdge>::with_capacity(cards.len());

    for c in cards {
        edges.push(CardsConnectionEdge {
            cursor: c.id.clone()
            node: c
        })
    }

    CardsConnection {
        edges
    }
}
```

but looks a bit clumsy.

It there a way to do the same thing using iterators or some kind of map function? Or what is the idiomatic way of solving this kind of problems?

`rust` `reference` `iterator` `move` `borrowing`

Share
Improve this
question
Follow

asked Nov 12 '20 at 20:18

Babbleopia
11 ● 1

Add a comment

1 Answer

Active   Oldest   Votes

▲

2

▼ 🕑

You are looking for `into_iter()` The `iter` function iterates over the items by reference, while `into_iter` iterates over the items, moving them into a new scope.

So `for c in cards {...}` is essentially the same as `cards.into_iter().for_each(|c| ... )`. Both move the elements of cards into the `...` scope.

```
fn new(cards: Vec<Card<'t>>) -> Self {
    CardsConnection {
        edges: cards
            .into_iter()
            .map(|c| CardsConnectionEdge {
                cursor: c.id.clone(),
                node: c,
            })
            .collect::<Vec<CardsConnectionEdge>>(),
    }
}
```

Reference: iter vs. into_iter

Share
Improve this answer
Follow
edited Nov 12 '20 at 20:52

answered Nov 12 '20 at 20:41

Ibraheem Ahmed
6,841 ● 1 ● 19 ● 30

2

You could also consider implementing `From<Card>` for `CardsConnectionEdge`. This would allow to simplify the conversion code to `cards.into_iter().map(From::from).collect()` (no explicit type hints needed).
– Sven Marnach
Nov 12 '20 at 21:02

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 reference iterator move borrowing or ask your own question.

Linked

What is the difference between iter and into_iter?

Related

3618
What are the differences between a pointer variable and a reference variable in C++?

What does the "yield" keyword do?

What is the most effective way to get the index of an iterator of an std::vector?

What is the idiomatic Rust way to copy/clone a vector in a parameterized function?

What are the differences between Rust's `String` and `str`?

Cannot move out of borrowed content when trying to transfer ownership

Method not compatible with trait with confusing error message

IntoIterator as a function argument doesn't accept adapter struct

Hot Network Questions

Changing a color of a link

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?

Program that translates a single word into five languages

Why is the light source not showing but light is being cast on the object

Split polyline to equal parts using QGIS

more hot questions

Question feed