Can I mutate a vector with a borrowed element?

Asked 10 months ago

Active 10 months ago

Viewed 201 times

1

I'm attempting to store a reference to an element of a mutable vector to use later. However, once I mutate the vector, I can no longer use the stored reference. I understand that this is because borrowing reference to the element also requires borrowing a reference to the vector itself. Therefore, the vector cannot be modified, because that would require borrowing a mutable reference, which is disallowed when another reference to the vector is already borrowed.

Here's a simple example

```
struct Person {
    name: String,
}

fn main() {
    // Create a mutable vector
    let mut people: Vec<Person> = ["Joe", "Shavawn", "Katie"]
        .iter()
        .map(|&s| Person {
            name: s.to_string(),
        })
        .collect();

    // Borrow a reference to an element
    let person_ref = &people[0];

    // Mutate the vector
    let new_person = Person {
        name: "Tim".to_string(),
    };
    people.push(new_person);

    // Attempt to use the borrowed reference
    assert!(person_ref.name == "Joe");
}
```

which produces the following error

```
error[E0502]: cannot borrow `people` as mutable because it is also borrowed as immutable
  --> src/main.rs:21:5
   |
15 |     let person_ref = &people[0];
   |                       ------ immutable borrow occurs here
...
21 |     people.push(new_person);
   |     ^^^^^^^^^^^^^^^^^^^^^^^ mutable borrow occurs here
...
24 |     assert!(person_ref.name == "Joe");
   |             --------------- immutable borrow later used here
```

I've also tried boxing the vector elements as suggested here, but that doesn't help. I thought it might allow me to drop the reference to the vector while maintaining a reference to the element, but apparently not.

```
struct Person {
    name: String,
}

fn main() {
    // Create a mutable vector
    let mut people: Vec<Box<Person>> = ["Joe", "Shavawn", "Katie"]
        .iter()
        .map(|&s| {
            Box::new(Person {
                name: s.to_string(),
            })
        })
        .collect();

    // Borrow a reference to an element
    let person_ref = people[0].as_ref();

    // Mutate the vector
    let new_person = Box::new(Person {
        name: "Tim".to_string(),
    });
    people.push(new_person);

    // Attempt to use the borrowed reference
    assert!(person_ref.name == "Joe");
}
```

This still produces the same error.

```
error[E0502]: cannot borrow `people` as mutable because it is also borrowed as immutable
  --> src/main.rs:23:5
   |
17 |     let person_ref = people[0].as_ref();
   |                      ------ immutable borrow occurs here
...
23 |     people.push(new_person);
   |     ^^^^^^^^^^^^^^^^^^^^^^^ mutable borrow occurs here
...
26 |     assert!(person_ref.name == "Joe");
   |             --------------- immutable borrow later used here
```

Is there a way to do this, or am I trying to do something impossible?

rust reference borrow-checker borrowing

Share
Improve this question
Follow

asked Feb 16 at 5:44

Oliver Evans
1,150 ● 10 ● 13

Is this a theoretical or real-world example? The reason is that since the option to use `assert!(people[0].name == "Joe");` is implicitly discarded, it's not clear if the question is theoretical, or it's a real-world one with a bigger picture that prevents the solution mentioned.
– Marcus
Feb 16 at 15:36

@Marcus - I'm not sure what you mean "the option to use ... is implicitly discarded". That was just a way to attempt to access `people[0].name`. The question does come from a real-world example of attempting to fill a mutable vector in a loop while storing references to the elements in a hash map. Context.
– Oliver Evans
Feb 16 at 16:51

1

The context clarifies everything :) I think there's no better solution than your answer in this case, given that one reference may go out of scope.
– Marcus
Feb 16 at 18:35

Add a comment

1 Answer

Active Oldest Votes

0

I found that using a reference counted smart pointer allows me to accomplish what I'm attempting. It makes sense that a shared ownership is necessary, because otherwise the element reference would become invalid if the original vector were to go out of scope (which would deallocate the element, with or without the `Box`).

The following code compiles successfully.

```
use std::rc::Rc;

struct Person {
    name: String,
}

fn main() {
    // Create a mutable vector
    let mut people: Vec<Rc<Person>> = ["Joe", "Shavawn", "Katie"]
        .iter()
        .map(|&s| {
            Rc::new(Person {
                name: s.to_string(),
            })
        })
        .collect();

    // Borrow a reference to an element
    let person_ref = Rc::clone(&people[0]);

    // Mutate the vector
    let new_person = Rc::new(Person {
        name: "Tim".to_string(),
    });
    people.push(new_person);

    // Attempt to use the borrowed reference
    assert!(person_ref.name == "Joe");
}
```

If anyone else has any corrections, improvements or further insight, I'd be glad to hear it. But if not, I feel satisfied with this answer for now.

Share
Improve this answer
Follow

answered Feb 16 at 5:44

Oliver Evans
1,150 ● 10 ● 13

1

"otherwise the element reference would become invalid if the original vector were to go out of scope" the vector can't go out of scope here, the issue is that mutating the vector can invalidate the reference entirely. Specifically here, `push` can cause a resize of the vector, which may require the contents of the vector to be *moved* to a new allocation. The reference would therefore become *dangling*.
– Masklinn
Feb 16 at 6:58

So do you think there *is* a way to do this using `Box` instead of `Rc`, then?
– Oliver Evans
Feb 16 at 7:36

1

Well yes and no, at a technical level the indirection would be there but it would bee rejected because the type system would not understand it. So if you absolutely have to do this Rc seems like the least bad way to do it.
– Masklinn
Feb 16 at 7:39

It sounds like your're really saying "just no". If you know of a way, could you provide an example?
– Oliver Evans
Feb 16 at 7:41

1

@Oliver You could do it with `unsafe` code, but that muddies the ownership and doesn't scale well. Another common thing to do in Rust is store indices into a vector rather than references to its items (example).
– trent formerly cl
Feb 17 at 1:44

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 borrow-checker borrowing 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

Linked

Reference to element in vector

Related

List changes unexpectedly after assignment. Why is this and how can I prevent it?

Cannot move out of borrowed content / cannot move out of behind a shared reference

Borrow errors for multiple borrows

Borrowing mutable twice while using the same variable

How do I write a rust function that can both read and write to a cache?

How can multiple parts of self be borrowed here? Isn't self borrowed mutably as well as immutably here?

Hot Network Questions

- which one of these paths has the priority: /usr or /usr/local
- What's the social meaning of "He was a student of..."?
- FEM for vector valued problems: reference request
- `apt-mark showmanual` shows almost all packages. messed up?
- Which Advent is it?
- more hot questions

Question feed