use of moved value, which is non-copyable [E0382] [E0277]

Ask Question

Asked 10 years, 3 months ago

Modified 10 years, 3 months ago

Viewed 1k times

2

I have an ownership problem which I don't understand well. Basically I try to create some hardlinks on my file system and to remove them after being created. Therefore I created a range of integers which I map to the actual file names I like to create and destroy. My naive solution looks like this:

```
use std::fs;

const src_file: &'static str = "a.txt";
const file_ext: &'static str = ".txt";

fn create_hardlink(dest_file: &str) {
   fs::hard_link(&src_file, &dest_file);
}

fn main() {

   let create = (0..10000).map(|x| x.to_string() + file_ext);
   let remove = (0..10000).map(|x| x.to_string() + file_ext);

   for file in create {
       create_hardlink(&file);
   }

   for file in remove {
       fs::remove_file(&file);
   }
}
```

But what I actually like to accomplish is a solution, where I don't have to repeat my self for creating the static collection with the file-names and can reuse `files` for a second for-loop:

```
...

fn main() {

   let files = (0..10000).map(|x| x.to_string() + file_ext);

   for file in files {
       create_hardlink(&file);
   }

   for file in files {
       fs::remove_file(&file);
   }
}
```

So when I try this the compiler complains, that the second usage of `files` is not possible,

```
src/main.rs:20:17: 20:22 error: use of moved value: `files` [E0382]
src/main.rs:20     for file in files {
```

because `files` already moved into the first for-loop:

```
src/main.rs:16:17: 16:22 note: `files` moved here because it has type `core::iter::Map<core::ops::Range<i32>, [closure@src/mai
```

after reading the explanation for `rustc --explain E0382` I decided to change the code as follows:

```
...

fn main() {

   let files = Rc::new(RefCell::new((0..10000).map(|x| x.to_string() + file_ext)));

   for file in files.clone() {
       create_hardlink(&file);
   }
```

```
   for file in files.clone() {
       fs::remove_file(&file);
   }
}
```

But this does not work as expected to me:

```
src/main.rs:16:5: 18:6 error: the trait `core::iter::Iterator` is not implemented for the type `alloc::rc::Rc<core::cell::RefC
src/main.rs:16     for file in files.clone() {
src/main.rs:17         create_hardlink(&file);
src/main.rs:18     }
note: in expansion of for loop expansion
src/main.rs:16:5: 18:6 note: expansion site
src/main.rs:16:5: 18:6 help: run `rustc --explain E0277` to see a detailed explanation
src/main.rs:16:5: 18:6 note: `alloc::rc::Rc<core::cell::RefCell<core::iter::Map<core::ops::Range<_>, [closure@src/main.rs:14:5
src/main.rs:16     for file in files.clone() {
src/main.rs:17         create_hardlink(&file);
src/main.rs:18     }
note: in expansion of for loop expansion
src/main.rs:16:5: 18:6 note: expansion site
src/main.rs:16:5: 18:6 note: required by `core::iter::IntoIterator::into_iter`
src/main.rs:16     for file in files.clone() {
src/main.rs:17         create_hardlink(&file);
src/main.rs:18     }
```

What can I do? Do I really have to implement the `core::iter::Iterator` for the type `alloc::rc::Rc<core::cell::RefCell<core::iter::Map<core::ops::Range<_>` like `rustc --explain E0277` is telling me? I hope not...

Is there a simple solution like defining `files` statically as `static`or as `const`? Or is my approach with mapping a `Range` non rusty?

Why do I have a type like `<core::iter::Map<core::ops::Range<_>` and not something like `<core::iter::String>`?

I hope you can help me out with that and enlighten a bit the Rust ownership principle to a novice like me.

- vector
- iterator
- rust
- clone
- ownership

Share

Improve this question

Follow

edited Aug 24, 2015 at 14:41

asked Aug 24, 2015 at 6:10

swiesend

1,10011 gold badge1313 silver badges2525 bronze badges

Add a comment |

2 Answers

Sorted by: Reset to default

Highest score (default) Trending (recent votes count more) Date modified (newest first) Date created (oldest first)

5

Rust iterators are only forward iterators, as far as I understand, so they can only be iterated once. You can either `collect` them into a vector or use a function to generate your iterator:

```
// 1st option
let files: Vec<_> = (0..10000).map(|x| x.to_string() + file_ext).collect();

for f in &files { ... } // Borrow `files`

// 2nd option
let files = || (0..10000).map(|x| x.to_string() + file_ext);

for f in files() { ... } // Call the closure to get an iterator
```

Share

Improve this answer

Follow

answered Aug 24, 2015 at 6:51

filmor

32.6k66 gold badges5353 silver badges4848 bronze badges

Sign up to request clarification or add additional context in comments.

3 Comments

Add a comment

Vladimir Matveev

Vladimir Matveev Over a year ago

I'd say that the first option is *the* idiomatic solution. The second option will result to extra 10000 allocations because the range will be run through twice.

2015-08-24T08:10:19.337Z+00:00

0

Reply

- Copy link

filmor

filmor Over a year ago

But the first one will allocate all memory at once, depending on the amount of files that might not be so nice either.

2015-08-24T08:40:33.32Z+00:00

2

Reply

- Copy link

swiesend

swiesend Over a year ago

Thank you for your solutions and comments! Would it be a valuable approach to create a macro to avoid the double allocation from option 2 like this? `macro_rules! files_vec { ( $range:expr ) => { $range.map(|x| x.to_string() + file_ext).collect() } }` which is quite inflexible, but pre compiled

2015-08-24T15:50:14.763Z+00:00

0

Reply

- Copy link

Add a comment

0

There are several problems here.

The first is that calling

```
for f in files { ... }
```

will take `files` by value. This is avoidable by taking a reference instead:

```
for f in &files { ... }
```

because `(&foo).into_iter()` effectively resolves to `foo.iter()`.

The second is that `files` must be `mut`, and the reference `&mut` if you are iterating an iterator. If you had some vector, it would make sense to iterate `&my_vector` - you can iterate it without modifying it. However, if you have an iterator itself, the state is kept and updated in the iterator itself.

```
let mut files = (0..10000).map(|x| x.to_string() + file_ext);

for file in &mut files {
   create_hardlink(&file);
}

for file in files {
   fs::remove_file(&file);
}
```

The third is that even if you did these things, since you are using a single iterator, you can only iterate each element once! The second loop will be empty. This is the problem @filmor offers solutions for.

Share

Improve this answer

Follow

answered Aug 24, 2015 at 12:48

Veedrac

60.7k1515 gold badges120120 silver badges177177 bronze badges

1 Comment

Add a comment

swiesend

swiesend Over a year ago

Thank you for your explanation! As you point out my code makes more sense not to have an iterator and thus I accepted the answer from @filmor.

2015-08-24T15:03:25.97Z+00:00

0

Reply

- Copy link

Your Answer

Thanks for contributing an answer to Stack Overflow!

- Please be sure to *answer the question*. Provide details and share your research!

But *avoid* …

- Asking for help, clarification, or responding to other answers.
- Making statements based on opinion; back them up with references or personal experience.

To learn more, see our tips on writing great answers.

Sign up or log in

Sign up using Google

Sign up using Email and Password

Submit

Post as a guest

Name

Email

Required, but never shown

Post as a guest

Name

Email

Required, but never shown

Post Your Answer Discard

By clicking “Post Your Answer”, you agree to our terms of service and acknowledge you have read our privacy policy.

Start asking to get answers

Find the answer to your question by asking.

Ask question

Explore related questions

- vector
- iterator
- rust
- clone
- ownership

See similar questions with these tags.

- The Overflow Blog
  Introducing Stack Overflow AI Assist—a tool for the modern developer
  Treating your agents like microservices
- Featured on Meta
  Chat room owners can now establish room guidelines
  AI Assist is now available on Stack Overflow
  Policy: Generative AI (e.g., ChatGPT) is banned

Visit chat

Related

10
Operator overloading by value results in use of moved value
5
Clone not invoked for Moved value?
3
Use of moved value error even with use of cloned()
9
Why are iterators not copyable?
14
How can I solve "use of moved value" and "which does not implement the `Copy` trait"?
2
Why doesn't clone() allow for this move?
18
move occurs because value has type Vec<T>, which does not implement the `Copy` trait
4
How to solve [E0382]: use of moved value in a for loop?
1
How to fix Rust error "value used here after move"?
7
Why the semantic of a moved variable after reinitiating is not move?

Hot Network Questions

- Is my homebrew Species "Attuned" balanced for D&D 2024?
- Can a 10*10 square be paved with 1*4 rectangular stone plates?
- What does Felix mean by telling Bond to pick a contact point while "standing up" in Diamonds Are Forever?
- Generating Irregular, Separated Polygonal Cells along a Curve in Geometry Nodes
- is it possible to keep the space when using url in latex
- Why does Blender import PLY vertex colors incorrectly (89 → 0.1 instead of 0.349)?

- Displaying features in a layer of geometrytype = GeometryCollection25D
- Is Codepage 1252 guaranteed to be available on all Windows systems?
- What is the difference between an ultraviolet limit and ultraviolet stability?
- How to prevent a batch file call from escaping a caret character?
- Ping sweep of IP subnets
- "Save as webpage, HTML Only" Fail while offiline, Firefox
- How to handle Expeditious Retreat with the optional chase rules
- Is there any way to restore hearts without consuming anything, without hot springs, and without shrines?
- Unity HDRP custom pass stencil buffer not working in build mode
- Swords and sandals movie where the ancient Greek hero fights in space
- Recent time traveling TV show. The lead might have been Asian(?), set in San Francisco(?)
- How to add blur inside subtracted element in Figma?
- PSE Advent Calendar 2025 (Day 5): Cinnamon overdose
- How do I add a text editor to my (debian-based) initramfs?
- Why does clang zero "eax" before calling a function with unspecified parameters, but gcc doesn't?
- Looking for a proper formal substitute for "quick fix" for formal letter
- variable name change causes segfault
- I2C bus problems when using two distance sensors in Arduino

more hot questions

Question feed

Subscribe to RSS

Question feed

To subscribe to this RSS feed, copy and paste this URL into your RSS reader.

- Blog
- Facebook
- Twitter
- LinkedIn
- Instagram