Rust lifetime in closure environment

Asked 3 years, 10 months ago

Active 3 years, 10 months ago

Viewed 553 times

2

I want to implement a graph structure in Rust. For this goal, I wrote simple abstractions:

```
pub struct Graph<'a> {
    pub nodes: Vec<Node>,
    pub edges: Vec<Edge<'a>>,
}

#[derive(Debug)]
pub struct Node {
    pub id: String,
    pub label: String,
}

pub struct Edge<'a> {
    pub source: &'a Node,
    pub target: &'a Node,
}
```

`Graph` contains vectors of `Nodes` and `Edges` . Every `Edge` has a ref to a `Node` in the same `Graph` .

I don't know it's a possible write something like this.

I tried to write a static method that builds a new `Graph` instance from a JSON representation:

```
impl<'a> Graph<'a> {
    pub fn from_json(json: &String) -> Graph {
        if let json::JsonValue::Object(deserialized) = json::parse(json.as_ref()).unwrap() {
            let nodes: Vec<Node> = deserialized
                .get("nodes")
                .unwrap()
                .members()
                .map(|v| {
                    if let json::JsonValue::Object(ref val) = *v {
                        return Node {
                            id: val.get("id").unwrap().to_string(),
                            label: val.get("label").unwrap().to_string(),
                        };
                    }
                    panic!("Invalid structure of json graph body.")
                })
                .collect::<Vec<Node>>();
            let edges: Vec<Edge> = deserialized
                .get("edges")
                .unwrap()
                .members()
                .map(|v| {
                    if let json::JsonValue::Object(ref val) = *v {
                        let source = (*nodes)
                            .iter()
                            .find(|&v| v.id == val.get("source").unwrap().to_string())
                            .unwrap();
                        let target = (*nodes)
                            .iter()
                            .find(|&v| v.id == val.get("target").unwrap().to_string())
                            .unwrap();
                        return Edge { source, target };
                    }
                    panic!("Invalid structure of json graph body.")
                })
```

When I compile, I get this error:

```
error[E0373]: closure may outlive the current function, but it borrows `nodes`, which is owned by the current function
  --> src/graph.rs:30:22
   |
30 |                 .map(|v| {
   |                      ^^^ may outlive borrowed value `nodes`
31 |                     if let json::JsonValue::Object(ref val) = *v {
32 |                         let source = (*nodes).iter().find(|&v| v.id ==  val.get("source").unwrap().to_string()).unwrap();
   |                                       ----- `nodes` is borrowed here
   |
help: to force the closure to take ownership of `nodes` (and any other referenced variables), use the `move` keyword
   |
30 |                 .map(move |v| {
   |                      ^^^^^^^^
   |

error: aborting due to previous error
```

A possible solution to this problem is to add `move` before the closure parameters, but I need the `nodes` vector to build the `Graph` instance.

What am I doing wrong?

rust closures lifetime

Share
Improve this question
Follow

edited Feb 18 '18 at 14:32

Shepmaster
305k ● 59 ● 824 ● 1083

asked Feb 18 '18 at 10:49

Дмитрий Сулохин
31 ● 6

1

Useful question Why can't I store a value and a reference to that value in the same struct?
– red75prime
Feb 18 '18 at 13:06

1

This question could be helpful too: Implement graph-like datastructure in Rust
– red75prime
Feb 18 '18 at 13:56

@red75prime Second link is very helpful, thank you for help
– Дмитрий Сулохин
Feb 18 '18 at 14:20

This was cross-posted to the Rust user's forum.
– Shepmaster
Feb 18 '18 at 16:23

Add a comment

1 Answer

Active Oldest Votes

▲
0
▼

✓

After some research, I found this article's: Rust doc. Smart pointers, Users Rust Lang, and I understood my mistakes. The first one: I remove lifetime parameters from structs definitions.

```
use std::rc::Rc;
#[derive(Debug)]
pub struct Graph {
    pub nodes: Vec<Rc<Node>>,
    pub edges: Vec<Edge>
}
#[derive(Debug)]
pub struct Node {
    pub id: String,
    pub label: String
}
#[derive(Debug)]
pub struct Edge {
    pub source: Rc<Node>,
    pub target: Rc<Node>
}
```

Second thing: I rewrote the code of `from_json` function for using `Rc<T>` instead of raw references.

```
impl Graph {
    pub fn from_json(json: & String) -> Graph {
        if let json::JsonValue::Object(deserialized) = json::parse(json.as_ref()).unwrap() {
            let nodes : Vec<Rc<Node>> = deserialized.get("nodes").unwrap().members()
                .map(|v| {
                    if let json::JsonValue::Object(ref val) = *v {
                        return Rc::new(Node {
                            id: val.get("id").unwrap().to_string(),
                            label: val.get("label").unwrap().to_string()
                        });
                    }
                    panic!("Invalid structure of json graph body.")
            }).collect::<Vec<Rc<Node>>>();
            let edges : Vec<Edge> = deserialized.get("edges").unwrap().members()
                .map(|v| {
                    if let json::JsonValue::Object(ref val) = *v {
                        let source = nodes.iter().find(|&v| v.id == val.get("source").unwrap().to_string()).unwrap();
                        let target = nodes.iter().find(|&v| v.id == val.get("target").unwrap().to_string()).unwrap();
                        return Edge {
                            source: Rc::clone(&source),
                            target: Rc::clone(&target)
                        };
                    }
                    panic!("Invalid structure of json graph body.")
                }).collect::<Vec<Edge>>();
            return Graph {
                nodes,
                edges
            }
        }
        panic!("Incorrect struct of json contains!");
    }
}
```

Now it works. Thanks for sharing useful links. I found a lot of helpful information about building graph structs in `Rust` such as: Graph structure in Rust

Share
Improve this answer
Follow

answered Feb 20 '18 at 15:55

Дмитрий Сулохин
31 ● 6

This is interesting: why the element type of Vec could change the borrow behavior of the Vec itself?
– zsf222
Mar 3 at 5:25

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 closures lifetime 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

Why can't I store a value and a reference to that value in the same struct?

Implement graph-like data structure in Rust

Related

What is the difference between a 'closure' and a 'lambda'?

Access to Modified Closure

What is the lifetime of a static variable in a C++ function?

JavaScript closure inside loops – simple practical example

In PHP, what is a closure and why does it use the "use" identifier?

Why doesn't println! work in Rust unit tests?

Why are explicit lifetimes needed in Rust?

lifetime with closure captures in rust

Confused by Rust closure lifetime

Specify Rust closures lifetime

Hot Network Questions

Why is 20m band waterfall showing signals every 50 kHz?

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?

What do I do when my boss is sabotaging interviews?

Company kept previous personal phone number

What edges are not in a Gabriel graph, yet in a Delauney graph?

more hot questions

Question feed

STACK OVERFLOW

Questions
Jobs
Developer Jobs Directory
Salary Calculator
Help
Mobile

PRODUCTS

Teams
Talent
Advertising
Enterprise

COMPANY

About
Press
Work Here
Legal
Privacy Policy
Terms of Service
Contact Us
Cookie Settings
Cookie Policy

STACK EXCHANGE NETWORK

Technology
Culture & recreation
Life & arts
Science
Professional
Business
API
Data

Blog
Facebook
Twitter
LinkedIn
Instagram