How to have multiple references for a single node in a tree structure using Rust

Asked 2 years, 1 month ago

Active 2 years, 1 month ago

Viewed 756 times

▲

2

▼

1

Trying to create a tree in rust with the following struct:

```
pub struct Node{
    pub children: Vec<Box<Node>>,
    pub parent: Option<Box<Node>>,
    pub value: f32,
    //.....
}
```

To build a new node the following function is used:

```
pub fn build_node(parent: Option<Box<Node>>)-> Node{
    Node{
        children: vec![],
        parent,
        value: 0.0,

    }
}
```

When trying to add nodes, for example with:

```
let mut root_nd = tree::build_node(None, 5, state);
let mut next_nd = tree::build_node(Some(Box::new(root_nd)), 2);
root_nd.children.push(Box::new(next_nd));
```

There will be errors, because I am borrowing for example `root_nd` and then trying to add `next_nd` to the `root.children` list, and even if there wasnt this error I would still need to have a reference for `next_nd` after adding it to the children of `root_nd`. I know that in rust it is not possible to have several mutable references simultaneously for the same element. So the question is how is it possible to make a tree-like data structure, with bi-directional references in rust? In my head this is a conflict since rust does not want multiple references but I need a node in the middle of the tree to be referenced by both his parent node and his children nodes.

reference rust tree

Share
Improve this question
Follow

asked Nov 3 '19 at 18:27

Miguel
1,154 ● 6 ● 22

Add a comment

1 Answer

Active Oldest Votes

▲

4

▼

✓

I've been meddling with trees in Rust for quite a bit recently. To work with trees in rust, you will need `Rc` [(A single-threaded reference-counting pointer)](#) so that you can have multiple ownership. And you'll also need `RefCell` to enable interior mutability since multiple mutable references are not allowed by the compiler. With `Rc` and `RefCell`, you can define your *TreeNode* as following:

```
use std::rc::Rc;
use std::cell::RefCell;

pub struct TreeNode {
    pub children: Vec<Rc<RefCell<TreeNode>>>,
    pub parent: Option<Rc<RefCell<TreeNode>>>,
    pub value: f32,
}
```

And here is one way to create two nodes that references each other:

```
impl TreeNode {
  #[inline]
  pub fn new(value: f32) -> Self {
    TreeNode {
      value,
      children: vec![],
      parent: None
    }
  }
}

let mut root_nd = Rc::new(RefCell::new(TreeNode::new(5.0)));
let mut child_nd = Rc::new(RefCell::new(TreeNode::new(2.0)));

child_nd.borrow_mut().parent = Some(root_nd.clone());  // use Rc::clone to create a new reference to root_nd
root_nd.borrow_mut().children.push(child_nd.clone());
```

Since we use `Rc::clone` to create a new reference to the node, `root_nd` and `child_nd` are not consumed and can still be accessed in later program.

More examples on Trees in Rust:

- leetcode 95 Unique Binary Search Trees
- leetcode 94 Binary Tree Inorder Traversal
- leetcode 100 Is Same Tree

Share
Improve this answer
Follow
edited Nov 3 '19 at 19:14

answered Nov 3 '19 at 18:54
Psidom
188k • 24 • 281 • 306

There are other solutions, like using arenas to store the nodes and use indexes or ids as "pointers" (look for example for id-arena).
– Denys Séguret
Nov 3 '19 at 19:29

So every time I want to use it, I should pass the full rc<refcell<..>> to the function and then use `borrow_mut()` to extract and access its elements?
– Miguel
Nov 3 '19 at 20:00

1

It depends on what you want to do. Normally a node is represented as `let node: Option<Rc<RefCell<TreeNode>>>` . And when you pass the reference to the function, you probably need to clone it so that the current reference is not moved. i.e. `somefun(node.clone())` . And to access its elements, use `borrow` (immutable) or `borrow_mut` (mutable).
– Psidom
Nov 3 '19 at 21:18

@Psidom I got it working after some grinding! Thank you. Just one more thing, why do you like to use option on the <Rc<RefCell? In your use case it can be passed as None? Because I did not need it yet.
– Miguel
Nov 4 '19 at 0:44

Use `Option` here b/c the parent can be None, and btw `None` by itself is of type `Option` so if a value can be `None` , then its type must be `Option` .
– Psidom
Nov 4 '19 at 3:29

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 reference rust tree 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

Related

What is the most efficient/elegant way to parse a flat table into a tree?

528
How to implement a tree data-structure in Java?

Borrow errors for multiple borrows

How can I deserialize a tree of references in Rust?

Is there a way to iterate over a mutable tree to get a random node?

Why Rust prevents from multiple mutable references?

0
How do I properly insert into a Rust AVL Tree?

Hot Network Questions

Does saying "Keep it up" put me in an authoritative position?

Why doesn't dkim sign the letter?

Can a Pyromancer Sorcerer deal fire damage to a creature under the effect of Invulnerability?

Powering 2 5v strips using a 12v power supply

Why did the JWST solar array deploy early?

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