Cannot infer an appropriate lifetime for autoref due to conflicting requirements

Ask Question

Asked 5 years ago
Active 11 months ago
Viewed 20k times

33

6

I'm having lifetime issues with a particular function in my code. I'm following a tutorial in an attempt to learn Rust and SDL. The tutorial was slightly older and the SDL library has changed since its been written, so I'm following along while also adapting it towards the latest version of Rust-SDL.

The lifetime problem is in this function:

```
pub fn ttf_str_sprite(&mut self, text: &str, font_path: &'static str, size: i32, color: Color) -> Option<Sprite> {
    if let Some(font) = self.cached_fonts.get(&(font_path, size)) {
        return font.render(text).blended(color).ok()
            .and_then(|surface| self.renderer.create_texture_from_surface(&surface).ok())
            .map(Sprite::new)
    }
    //::sdl2_ttf::Font::from_file(Path::new(font_path), size).ok()
    self.ttf_context.load_font(Path::new(font_path), size as u16).ok()
        .and_then(|font| {
            self.cached_fonts.insert((font_path, size), font);
            self.ttf_str_sprite(text, font_path, size, color)
    })
}
```

particularly with the line `self.ttf_context.load_font(Path::new(font_path), size as u16).ok()` . The commented line above it is the old SDL version's font loading method.

```
error[E0495]: cannot infer an appropriate lifetime for autoref due to conflicting requirements
  --> src\phi/mod.rs:57:26
   |
57 |         self.ttf_context.load_font(Path::new(font_path), size as u16).ok()
   |                          ^^^^^^^^^
   |
help: consider using an explicit lifetime parameter as shown: fn ttf_str_sprite(&'window mut self, text: &str, font_path: &'static str,
            size: i32, color: Color) -> Option<Sprite>
```

The struct object for that implementation looks like this:

```
pub struct Phi<'window> {
    pub events: Events,
    pub renderer: Renderer<'window>,
    pub ttf_context: Sdl2TtfContext,

    cached_fonts: HashMap<(&'static str, i32), ::sdl2_ttf::Font<'window>>
}
```

The method is trying to load a font from Phi's `ttf_context` and load it into the hashmap. The Rust compiler suggested I add a lifetime to `self` in the function parameters, which, when I did that, caused a cascading effect to adding lifetimes to every method calling the original one, all the way down to `main()` and didn't help anything.

Since I'm still new to Rust, I'm not sure where the lifetime conflict resides or why this is happening. As a guess, I'm thinking that the `Font` object that is being generated is supposed to die with the end of that method but instead it's being loaded into a hashmap with a lifetime of `'window` and those two conflict. I don't know enough about Rust to fix that, though, or if that's even correct.

rust sdl lifetime

Share
Improve this question
Follow

edited Dec 21 '16 at 20:09

Shepmaster
305k ● 59 ● 824 ● 1083

asked Dec 21 '16 at 18:59

Brad Ziolko
335 ● 1 ● 3 ● 6

Add a comment

1 Answer

Active Oldest Votes

33

Here's a smaller example that reproduces the problem:

```
struct FontLoader(String);
struct Font<'a>(&'a str);

impl FontLoader {
    fn load(&self) -> Font {
        Font(&self.0)
    }
}

struct Window;

struct Phi<'window> {
    window: &'window Window,
    loader: FontLoader,
    font: Option<Font<'window>>,
}

impl<'window> Phi<'window> {
    fn do_the_thing(&mut self) {
        let font = self.loader.load();
        self.font = Some(font);
    }
}

fn main() {}
```

```
error[E0495]: cannot infer an appropriate lifetime for autoref due to conflicting requirements
  --> src/main.rs:20:32
   |
20 |         let font = self.loader.load();
   |                            ^^^^
   |
note: first, the lifetime cannot outlive the anonymous lifetime #1 defined on the method body at 19:5...
  --> src/main.rs:19:5
   |
19 |     fn do_the_thing(&mut self) {
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^
note: ...so that reference does not outlive borrowed content
  --> src/main.rs:20:20
   |
20 |         let font = self.loader.load();
   |                    ^^^^^^^^^^^
note: but, the lifetime must be valid for the lifetime `'window` as defined on the impl at 18:6...
  --> src/main.rs:18:6
   |
18 | impl<'window> Phi<'window> {
   |      ^^^^^^^
note: ...so that the expression is assignable
  --> src/main.rs:21:21
   |
21 |         self.font = Some(font);
   |                     ^^^^^^^^^^
   = note: expected `Option<Font<'window>>`
              found `Option<Font<'_>>`
```

The problem is indeed that you have constructed an impossible case. Specifically, the code states these points:

1. `Phi` is going to include a reference to a `Window`. That referred-to value lives for the lifetime `'window`.
2. `Phi` is going to include a `Font`, which contains a reference. That referred-to value lives for the lifetime `'window`.
3. `FontLoader` returns a `Font` which contains a reference to a value with the lifetime **of the loader**. This is due to lifetime inference, which when expanded looks like:

```
impl FontLoader {
    fn load<'a>(&'a self) -> Font<'a> {
        Font(&self.0)
    }
}
```

I highly encourage adding `#![deny(rust_2018_idioms)]` to your crate, which will disallow this specific type of lifetime inference.

Then the code attempts to load a `Font` from the `FontLoader` in `Phi`, which **does not** have the lifetime `'window` and store that `Font` into `Phi`. `FontLoader` (and thus `Font`) does not live long enough, so it cannot be stored in `Phi`.

The compiler has correctly prevented incorrect code.

Your next attempt would probably be to introduce a second lifetime:

```
struct Phi<'window, 'font> {
    window: &'window Window,
    loader: FontLoader,
    font: Option<Font<'font>>,
}

impl<'window, 'font> Phi<'window, 'font> {
    fn do_the_thing(&'font mut self) {
        let font = self.loader.load();
        self.font = Some(font);
    }
}
```

This will actually compile, but probably doesn't do what you want. See Why can't I store a value and a reference to that value in the same struct? for further information.

More likely, you want to take a reference to the font loader:

```
struct Phi<'a> {
    window: &'a Window,
    loader: &'a FontLoader,
    font: Option<Font<'a>>,
}

impl<'a> Phi<'a> {
    fn do_the_thing(&mut self) {
        let font = self.loader.load();
        self.font = Some(font);
    }
}
```

Here, I've renamed the lifetime as it isn't strictly for the window anymore.

Share
Improve this answer
Follow
edited Jan 18 at 14:37

answered Dec 21 '16 at 20:31

Shepmaster
305k ● 59 ● 824 ● 1083

4

FontLoader (and thus Font) does not live long enough: Isn't the lifetime of FontLoader the same as the Phi that contains it ?
– Carl Levasseur
Sep 24 '17 at 17:32

1

@CarlLevasseur yes, the lifetime of FontLoader and its containing Phi are the same. Why do you ask?
– Shepmaster
Sep 24 '17 at 18:26

4

Doesn't that mean it has the 'window lifetime then ? i don't understand why it does not live long enough if its lifetime is the same as the Phi object and therefore, the same as Phi.font ? In what case would the font loader be free'd before then end of the 'window lifetime ?
– Carl Levasseur
Sep 24 '17 at 18:37

1

@CarlLevasseur *In what case would the font loader be free'd before then end of the 'window lifetime* — in **every** case, I believe. *Doesn't that mean it has the 'window lifetime* — no. It contains a reference that has the window lifetime, but itself has a different lifetime.
– Shepmaster
Sep 24 '17 at 18:41

2

@MarioIshac since lifetimes prevent incorrect code from compiling, you need to use raw pointers to demonstrate the problem. Here's one such possibility. Note that the pointer now points to an invalid location.
– Shepmaster
Jan 18 at 14:44

Show 1 more 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 sdl lifetime or ask your own question.

Congratulations to the 59 sites that just left Beta

Linked

1
Managing SDL2 (rust) Texture lifetime

0
Rust - cannot infer an appropriate lifetime for autoref due to conflicting requirements

0
Is there a way to store a texture inside a struct using rust-sdl2?

-3
How to solve this rust lifetime bound issue of SDL2?

1
Cannot infer an appropriate lifetime for autoref due to conflicting requirements

0
rust-sdl2 ttf fonts and lifetimes

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

`cannot infer an appropriate lifetime for autoref due to conflicting requirements` but can't change anything due to trait definition constraints

How to read a lifetime error without looking at the code?

2
Is there a way to have a struct contain a reference that might no longer be valid?

See more linked questions

Related

1
error: cannot infer an appropriate lifetime for autoref due to conflicting requirements [E0495]

Cannot infer an appropriate lifetime due to conflicting requirements in a recursive struct

Rust: error[E0495]: cannot infer an appropriate lifetime for autoref due to conflicting requirements

cannot infer an appropriate lifetime for borrow expression due to conflicting requirements

Rust: cannot infer an appropriate lifetime for autoref due to conflicting requirements

Hot Network Questions

EU ETS: if within-EU flight emissions are limited to climate goals, is there still a reason not to fly as long as you can afford it?

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

'Cloth shop' and 'Clothes shop'

Would I be able to avoid the wash sale rule if I buy back the security on January 1st after selling it on December 31st?

On what basis do countries repay international loans?

more hot questions

Question feed

Instagram