Simple rust interview questions

By **flakm**

May 8, 2022 - 8 minutes read - 1511 words

What is a good interview question?

For some time now I've been using interview questions on Reddit and SO to check my progress in learning Rust. Sadly apart from a huge will to share and teach in the community ❤️ I've also seen some ego-boosting toxicity. Maybe suggesting my perspective will interest you.

I strongly believe that the main objective of a technical interview is determining whether you would enjoy spending a better part of your waking hours coding, side by side while managing to create immense value.

The **good interview** questions should:

1. Check the ability to think outside the box
2. Invite discussion and cooperation
3. Check deep understanding of key concepts
4. Check the ability to communicate tricky parts of the technology
5. Check formal knowledge of the domain

I believe that we can focus too much on a single objective (mainly #5) while ignoring the rest.

Table of contents:

Checking creativity (non-rust specific)

Here are some awesome questions that may inspire you:

1. How would you QA the light inside the fridge?
2. How could you explain what docker does without words: `container`, `isolation`, and `virtual`?
3. How can you track the movements of the snail over time?
4. What was the most awkward technical decision you have ever made?
5. During the yearly international race of pandas, as an embedded developer what kind of a smart device would you prepare for your

favorite one to make it win?
6. What features can a smart teaspoon have?

Key concepts of rust

Those are the most important topics that technology touches on. Understanding those concepts is probably required to describe all other topics. They might introduce follow-up questions and discussions. Possible followups:

- Why do you think this feature is important?
- What are alternative solutions (maybe other languages)?
- Could you describe where this concept was very helpful?

What does ownership mean in rust?

Set of rules defining how the rust program manages memory. They are checked during compilation and they allow the compiler to add code that frees no longer used regions of memory.

The rules are:

1. Each value in Rust has a variable that's called its owner.
2. There can only be one owner at a time.
3. When the owner goes out of scope, the value will be dropped.

Resources:

- The book - what is ownership?

What does borrow do in rust?

More often than not we want to access some value without taking the ownership. Rust allows us to do this safely using borrowing (during the compilation the borrow checker tests if all rules are obeyed).

1. The borrow cannot outlive the owner. Reference has to be valid throughout its life.
2. Only one can be true at the same time:
    - There can be one or more references `&T` to an owned value
    - There can be only one mutable reference `&mut T` at the time

Followup questions:

1. Could you give examples where rust needs some support with understanding what value is it borrowing from?

Resources

- The book - references & borrowing

Explain `std::thread::spawn` signature (mainly `'static` part)

```
#[stable(feature = "rust1", since = "1.0.0")]
pub fn spawn<F, T>(f: F) -> JoinHandle<T>
where
    F: FnOnce() -> T,
    F: Send + 'static,
    T: Send + 'static,
{
    Builder::new().spawn(f).expect("failed to spawn thread")
}
```

`spawn` creates a new thread using provided closure that has lifetime `'static` so it has no other dependencies to borrowed values and might live until the end of the program and it returns the value of type `T` that is also `'static`.

The `'static` lifetime is required due to how threads work. The spawned thread might outlive the calling call so it needs to own all the data.

The `Send` trait means that provided closure `f` of type `F` and its return value `T` is safe to send to another thread.

Resources:

- Rust-lang docs: spawn
- Thread definition
- Rust-lang docs: FnOnce

Difficult and tricky parts of technology

These questions check the understanding and ability to describe complex topics specific to rust.

Whats the difference between `String` vs `&str`?

`String` is UTF-8 encoded dynamic owned string type backed by heap allocation. `str` otherwise called string slice is an immutable and borrowed sequence of UTF-8 encoded bytes. Since `str` size is unknown (its DST) it can be handled via reference `&str` (this type consists of a `pointer` and a `len`). `str` can point to either heap, static storage (binary code loaded into memory) or stack (ie created from slice of bytes)

Resources:

- Stackoverflow - short answer
- Faster than lime awesome article
- DST

Describe the `async` keyword in rust

The `async` keyword can change `fn`, `closure` or `block` into Future. The `Future` represents asynchronous computation: so one is not performed by blocking the current caller but by allowing to obtain those results sometime in the future.

Rust does not ship any async runtime in `std` by default. It only specifies interfaces that can be brought by using a separate crate like `tokio` or `async-std`

The async feature plays very nice with os' mechanisms for asynchronous io. It allows for very performant - non blocking networking or fs operations.

Resources

- Rust-lang: Future
- Async Book

Describe the `std`

`std` is a crate that provides the most common abstractions for a regular program that uses platform support for certain features (`io`, `concurrency`).

The most commonly used types from `std` like `Box`, `Option`, `String` or `Vec` are imported by a compiler for each program using prelude.

`std` also provides small runtime that initializes the environment before running the user's `main` function. For instance, on unix systems it casts some dark spells for standard file descriptors (`0-2`) to be sure they are there.

It is perfectly valid in certain scenarios (embedded or wasm) to not use `std` after using `no_std` attribute. You can still opt in for certain features like heap allocations or collections by using custom allocators.

Resources

- Rust-lang: std
- Rust-embedded: no-std

Formal knowledge of the domain

Those questions are sometimes domain-specific (FFI is not used by every project nor is unsafe rust) but the knowledge and ability to describe the

background is very important.

What can you do in unsafe block

The most common are: dereferencing a raw pointer:

```
    let original: u32 = 23;
    let ptr = &original as *const u32;
    // dereference a raw pointer
    let value = unsafe { *ptr };
    println!("{value}")
```

Calling FFI:

```
extern {
    fn hello(source_length: *const u8) -> *const u8;
}

fn main() {
    unsafe {
        hello("hello world!".as_ptr());
    }
}
```

You can call unsafe function or method

Resources:

- Rust lang raw pointers

Whats the difference between `trait` and `dyn trait`

Alternative questions:

1. Describe dynamic vs static dispatch
2. Why the `dyn trait` has to be always put behind a reference (or `Box/Arc`)
3. Why rust uses monomorphization

Monomorphization is used to solve at compile time concrete types for all of the generics before they are called. Rust generates separate copies of assembly for each type, so they have their own address in the resulting binary.

An alternative solution is using dynamic dispatch with `dyn trait` where each implementation reference is stored in vtable so there is an only a single implementation of function that looks up specific implementation for generics types using vtable.

If you define a trait and implement this for a type:

```
trait Multiplier {
    fn multiply(&self, d: impl Display) -> String;
    fn multiply_expanded<T: Display>(&self, d: T) -> String {
        self.multiply(d)
    }
    fn multiply_expanded_2<T>(&self, d: T) -> String
    where
        T: Display,
    {
        self.multiply(d)
    }
    // heres your dynamic dispatch
    fn multiply_dyn(&self, d: &dyn Display) -> String {
        self.multiply(d)
    }
}

struct Doubler;
```

```
impl Multiplier for Doubler {
    fn multiply(&self, d: impl Display) -> String {
        format!("{} {}", d, d)
    }
}
```

Resources:

- Rust for rustaceans chap 2 Compilation and dispatch
- rustc dev guide

Why does rust links dependencies statically

Rust by default links all dependencies statically except for glibc.

This is done because Rust doesn't have a stable ABI (Aplication binary interface) so it can break version to version.

It's perfectly possible to build dynamic library though using `dylib` crate type (or `cdylib` if its to be used by other c code).

Also some crates ship with dynamic dependencies (ie `openssl`)

Resources:

- Discussion
- Discussion 2

Conclusion

Be sure to be **respectful** - on either side.

Apart from domain knowledge try to check/present **creativity** and ability to **express thought processes**.

The whole experience should feel like a group of old friends having a good chat about compilers and GNU/Linux with a cup of coffee or a nice pint of craft beer. It's chaos, be kind!

Additional resources

- Reddit: interview tips
- Stackoverflow: how to ask a good question
- 10: Hands off, baby (embedded.fm podcast)
- 51: There Is No Crying in Strcpy (Repeat) (embedded.fm podcast)

Photo by hʌnzife on Unsplash

rust english interview