In Rust, what is the purpose of a mod.rs file?

Asked 9 years, 9 months ago Modified 8 months ago Viewed 50k times

83

In some Rust projects I've seen (i.e pczarn/rustboot), I've seen `mod.rs` files in directories for whatever reason. I've not been able to find documentation about this, and I've seen it in many other Rust projects. What is the purpose of a mod.rs file, and when should I use it?

module rust

Share Improve this question Follow

asked Oct 18, 2014 at 0:18
Liam Marshall
1,503 1 13 21

3 Answers

Sorted by: Highest score (default)

94

Modules are important to understand, but I find most documentations often leave you scratching your head on that matter.

Coming from Python or Javascript?

Roughly, `mod.rs` is kind of like `__init__.py` in python or `index.js` in javascript. But only kind of. This is a bit more complicated in Rust.

Rust is different

Folders are not immediately ready to use as modules in Rust.

You have to add a file named `mod.rs` in a folder to expose a new module named like that folder

From the Rust reference:

> Note: Previous to rustc 1.30, using mod.rs files was the way to load a module with nested children. It is encouraged to use the new naming convention as it is more consistent, and avoids having many files named mod.rs within a project.

Complete example

```
src
    utils
        bar.rs
        foo.rs
    main.rs
```

At this point, the compiler doesn't know about `src/utils/foo.rs` and `src/utils/bar.rs`.

First, you must expose `src/utils/`. As seen above, you have 2 options:

- add the file: `src/utils/mod.rs`
- add the file `src/utils.rs` (named exactly like the folder, without the extension)

Now, relative to the src folder (aka the crate level), a module named `utils` is available.

Second, you must expose the files `src/utils/foo.rs` and `src/utils/bar.rs`.

To do that, the `utils` module must declare 2 new submodules named after these files. So the content of `src/utils/mod.rs` (or `src/utils.rs`) should be:

```rust
pub mod bar;
pub mod foo;
```

Now whatever is public in those 2 files is available in other modules! 🎉

And you may write the following in `src/main.rs`:

```
        foo.rs
        mod.rs
    main.rs
```

Option 2 • `<folder_name>.rs` (the preferred way):

```
src
    utils
        bar.rs
        foo.rs
    utils.rs
    main.rs
```

More advanced details on how modules work

> This remains a surface explanation, your next destination is the official documentation 👩‍🎓

There is a third way to declare modules (core language):

```
mod utils {
  // module code goes here. That's right, inside of the file.
}
```

But it is also possible to just write `mod utils;`. In that case, as seen above, Rust knows to search for either of `src/utils.rs` or `src/utils/mod.rs`.

See, when you try to use a module in a file (in `src/main.rs` for example), you may reference it in the following ways:

- from inside: `src/main.rs`
  - `mod module { ... }`
- from nested modules inside: `src/main.rs`

A file or a folder containing `mod.rs` does not become a module.
Rather, the Rust language lets you organize modules (a language feature) with a file hierarchy.

What makes it really interesting is that you are free to mix all approaches together.

For example, you may think you can't directly reference `src/utils/foo.rs` from `main.rs`.
But you can:

```
// src/main.rs
mod utils {
  pub mod foo;
}
```

Important notes:

- modules declared in files will always take precedence (because you in fact never need to search the file hierarchy)
- you can't use the other 2 approaches to reference the same module

For example, having both `src/utils.rs` and `src/utils/mod.rs` will raise the following error at compile time:

```
error[E0761]: file for module `utils` found at both "src/utils.rs" and
"src/utils/mod.rs"
 --> src/main.rs:1:1
  |
1 | mod utils;
  | ^^^^^^^^^^
  |
  = help: delete or rename one of them to remove the ambiguity
```

Let's wrap up. Modules are exposed to the compiler:

- from top to bottom
- by reference only (That's why you don't have intellisense until your modules are "imported")
- starting from an entry point (which is `src/main.rs` or `src/lib.rs` by default. But it may be

3. `src/utils/foo.rs -> crate::utils::foo`

Share Improve this answer Follow edited Nov 11, 2023 at 21:02 answered Aug 29, 2021 at 12:37

Romain Vincent
3,333 2 24 30

7 Thanks, the Rust book is really weak in this regards. When you expose foo and bar, do you then do use utils::foo or just use foo? I would suspect the latter? – Markus Toman Sep 22, 2021 at 8:21

2 I just found in doc.rust-lang.org/reference/items/… that "Note: Previous to rustc 1.30, using mod.rs files was the way to load a module with nested children. It is encouraged to use the new naming convention as it is more consistent, and avoids having many files named mod.rs within a project." – Markus Toman Sep 22, 2021 at 11:57

@MarkusToman To use foo and bar in a file (main.rs for example), you need to bring the module in scope with `mod utils;` and then `use utils::{foo, bar};` – Romain Vincent Sep 22, 2021 at 20:22

What if I don't want a nested directory but want to put the other files at the same level as `main.rs` ? – Abhijit Sarkar Jun 26, 2022 at 0:11

1 @TalnaciSergiuVlad My pleasure. Modules are always a pain, and just like you, I felt a bit let down by all the official docs and examples. – Romain Vincent Nov 11, 2023 at 20:56

84

Imagine the following directory structure:

```
code/
  `- main.rs
   - something/
     `- mod.rs
```

If in `main.rs` you do `mod something;` , then it'll look in the `something/mod.rs` file to use as the contents of the module declaration for `something` .

The alternative to this is to have a `something.rs` file in the `code/` directory.

So to recap, when you write an empty module declaration such as `mod something;` , it looks either in:

1 Could I use any filename instead of `mod.rs` , or is that a builtin. – Liam Marshall Oct 18, 2014 at 1:49

2 You can specify any path you want using the `#[path = "thefile.rs"]` attribute, but it's generally discouraged since the convention is what I specified in my answer. See the reference. – Jorge Israel Peña Oct 18, 2014 at 2:30

2 1. What happens when both the files are present? `something.rs` and `something/mod.rs` ? 2. Isn't that weird that every file's default name is expected to be `mod.rs` , making search a horrible experience? – Coder Dec 29, 2021 at 20:34

1 In Rust 2018, mod.rs is not needed https://doc.rust-lang.org/edition-guide/rust-2018/path-changes.html – Jagger Yu May 11, 2022 at 12:17

1

Each `rs` file, except lib.rs and main.rs, which always match the `crate` module, gets its own module.

There is only one way to declare a module:

```
/* pub */ mod sub_module1;
```

A module cannot be declare outside the root/crate module tree (i.e., going up the module tree, a submodule must always have a parent that is declared directly in `lib.rs` or `main.rs` , so the first program submodule must always be declared there — a tree data structure if it isn't already obvious enough).

There are 2 ways to nest a module inside the module where it is declared:

- in `<module_where_it_is_declared>/<module_name.rs>`
- in `<module_where_it_is_declared>/module_name/mod.rs`

If `module_where_it_is_declared` is the crate module, then this corresponding subfolder is not needed (disappears from the scheme above).

Here is an example, valid for both lib and binary crates:

```
src
```

```
|---b2
|   |---mod.rs ( contains: pub mod bb2; )
|   |---bb2.rs
.   .
```

You can see that you can mix and match (b2 uses `mod.rs` way, bb2 uses the "file"-way).

Here's a way to only use the file pattern that is also valid:

```
src
|---lib.rs ( contains: pub mod b2; )
|---b2.rs ( contains: pub mod bb2; )
|---b2
|   |---bb2.rs (contains: pub mod bbb2; )
|   |---bbb2.rs (contains: pub mod bbbb2; )
|   |---bbb2
|   |   |---bbbb2.rs
.   .   .
```

I guess it depends on you how you want to nest modules.

I like the `mod.rs` syntax for modules that just export other submodules and don't have any other (or very little) code in them, although you can put whatever you want in `mod.rs`.

I use `mod.rs` similar to the barrel pattern from JS/TS world, to rollup several submodules into a single parent module.

Also don't forget modules can be defined (not only declared) inline by adding a scope block:

```
pub mod my_submodule {
    // ...
}
```

Share Improve this answer Follow

edited Nov 24, 2022 at 19:00

answered Nov 24, 2022 at 18:55

Paul-Sebastian Manole
2,624 1 36 38