The Rust 2018 Module System

Josh Triplett
josh@joshtriplett.org

RustConf 2019

History

2 0 1 6

Rust

CONF

RFC: In order to form a more perfect `union`

Adding C-style unions to Rust

to support building virtual machines

based on Linux and `/dev/kvm`

crosvm

Firecracker

rust-vmm

Cloud Hypervisor

Not just about a specific language feature

RFC: In order to form a more perfect Rust

Case study on the RFC process and evolving Rust

Language team and Cargo team

New module system developed for Rust 2018

Language team and processes

Ergonomics

Simplicity

Simplicity and consistency

January 2017: Module system discussion begins

Rust 2015 modules

module: organizational unit of code

crate: library, a group of modules

```rust
mod example; // module from example.rs

mod example {
    // inline module contents
}
```

Cargo.toml dependencies:

```
clap = "version"
```

main.rs or lib.rs:

```
extern crate clap;
use clap::Arg; // optional

fn parse_args() {
    let app = clap::App::new("example")
```

Top-level module: main.rs or lib.rs

Most crates start with just one module

Paths for crates worked differently in submodules, discouraging modularity

```
extern crate clap;

mod submodule {
    use clap::Arg;

    fn parse_args() {
        let app = ::clap::App::new("example");
    }
}
```

Paths for submodules also worked differently in submodules

```
mod m {
    pub struct S1;
    pub struct S2;
}

use m::S1;

fn f(arg1: S1, arg2: m::S2) {}
```

```
mod submodule {
    mod m {
        pub struct S1;
        pub struct S2;
    }

    use self::m::S1; // absolute by default

    fn f(arg1: S1, arg2: m::S2) {}
}
```

Big surprises when introducing modules to an existing one-file project

Many other improvements desired

Development and consensus processes

Requirements

- No `extern crate`
- Same syntax for top-level module and submodules

Extensive discussion in lang team meetings, and on internals.rust-lang.org and Discord

Extensive discussion in lang team meetings, and on internals.rust-lang.org and Discord

`extern::` (and variants) versus `crate::`

Extensive discussion in lang team meetings, and on internals.rust-lang.org and Discord

`extern::` (and variants) versus `crate::`

Three RFCs (~~2108~~, ~~2121~~, 2126)

Approach: *anchored use paths*

(Named later when we needed to distinguish it.)

`use` paths always started with a crate:

`cratename::`	An external crate
`crate::`	The top of the current crate
`self::`	The current module
`super::`	The parent module

Language team consensus on RFC 2126 “Path Clarity”

Language team consensus on RFC 2126 “Path Clarity”

Not fully satisfying

Language team consensus on RFC 2126 “Path Clarity”

Not fully satisfying

Mixed community reaction

"These situations are particularly bad in Rust 2015 because the code works without `self::` at the top level module, but not elsewhere."

"These situations are particularly bad in Rust 2015 because the code works without `self::` at the top level module, but not elsewhere."

"Rust 2018's current design helps by making the code not work anywhere."

Would have required changing most existing Rust code

`rustfix` could have helped, but still…

June 2018: I reached out to Aaron,
to propose and discuss an alternative.

New Requirements

- No `extern crate`
- Same syntax for top-level module and submodules
- Uniform paths between `use` and expressions
- Compatible with most Rust 2015 code

Concept: Uniform path resolution

Check identifiers in scope,
then crates and prelude

```
// Rust 2018
mod submodule {
    mod m {
        pub struct S1;
        pub struct S2;
    }

    use m::S1; // Looks for self-relative names

    fn f(arg1: S1, arg2: m::S2) {}
}
```

```rust
// Rust 2018
mod submodule {
    use clap::Arg; // Looks for crate names

    fn parse_args() {
        let app = clap::App::new("example");
        // crate names work in expressions too
    }
}
```

Only ambiguous if a name in scope conflicts with an extern crate name

Only ambiguous if a name in scope conflicts with an extern crate name

Require disambiguation:

```
crate::toplevelname
self::localname
::cratename
```

`uniform_paths`

vs

`anchored_use_paths`

Reluctance towards a new round of debate

Reluctance towards a new round of debate

Do not meddle with name resolution,
for it is subtle and quick to anger

Reluctance towards a new round of debate

Do not meddle with name resolution,
for it is subtle and quick to anger

Meaningful differences in values and
preferences among lang team members

Successful technical implementation

Sought community feedback

Careful discussion and introspection on conflicting core values

Released 1.31 and Rust 2018
with a compromise solution:

Error on ambiguity
Forward compatibility with either approach

Lang team collaboratively wrote a document about both alternatives

Made the decision shortly thereafter

Finished uniform_paths in 1.32

A few more details. . .

Macros

```rust
// Rust 2015
#[macro_use] extern crate clap;
// Imports *all* macros

fn main() {
    println!("{}", crate_name!());
}
```

```rust
// Rust 2018
use clap::crate_name;

fn main() {
    println!("{}", crate_name!());
}
```

```rust
// Rust 2018
fn main() {
    println!("{}", clap::crate_name!());
}
```

Macro paths work like function paths

You can have `foo.rs` and `foo/bar.rs`
You no longer need to use `foo/mod.rs` instead

Crate renaming in Cargo

Replacement for `extern crate foo as bar`

```
cargo-features = ["rename-dependency"]

[dependencies]
bar = { package="foo", version="..." }
```

Possible future work

Implicit `mod example;` for `example.rs`

Implicit `mod example;` for `example.rs`

Separated due to controversy

Interest remains

`crate` visibility
synonym for `pub(crate)`

`crate` visibility
synonym for `pub(crate)`

Separated due to corner case:

```
struct S(crate ::T));
```

Visibility or scope?

Reflecting on difficult decisions

async/await

async/await

Final syntax stabilized for 1.39!

Summary

- Beware of satisficing solutions

Summary

- Beware of satisficing solutions
- Raise issues early; people grow attached to "experimental" and "interim" solutions

Summary

- Beware of satisficing solutions
- Raise issues early; people grow attached to “experimental” and “interim” solutions
- Introspect on core values, including your own

Summary

- Beware of satisficing solutions
- Raise issues early; people grow attached to “experimental” and “interim” solutions
- Introspect on core values, including your own
- Work collaboratively towards each others’ values

Summary

- Beware of satisficing solutions
- Raise issues early; people grow attached to “experimental” and “interim” solutions
- Introspect on core values, including your own
- Work collaboratively towards each others’ values
- Seek satisfying solutions whenever possible

Acknowledgments

- Aaron Turon
- eddyb, cramertj, and petrochenkov

Acknowledgments

- Aaron Turon
- eddyb, cramertj, and petrochenkov
- The Rust language team
- The incredible Rust community

Thank you!

`josh@joshtriplett.org`

Twitter: `@josh_triplett`