Rust

A brief intro

Miguel Palhas (@naps62) / Subvisual

DAVID HEINEMEIER HANSSON

September 7, 2023

Open source hooliganism and the TypeScript meltdown

Types in the past

```
int main() {
  int x = 0;
  long int y = (long) x;

  println("%d", y);
}
```

1. slow to compile
2. type inference wasn't a thing
3. tooling

```
unresolved external symbol "void __cdecl
importStoredClients(class
std::basic_fstream<char,struct std::char_traits<char> > const
&,class
std::vector<class Client,class std::allocator<class Client> >
&)" (?
importStoredClients@@YAXABV?$basic_fstream@DU?
$char_traits@D@std@@@std@@AAV?
$vector@VClient@@V?$allocator@VClient@@@std@@@2@@Z)
referenced in function
_main DataTracker
```

```
1 + "2" == "12"
```

Undefined is not a function

Types of Types

Structural Typing (Typescript, OCaml, ...)


```ts
type Foo = { x: number, y: string };
type Bar = { x: number };

let x: Foo = { x: 1, y: "hello" };
```

Nominal Typing (Rust, C/C++, ...)

```
struct Foo { x: i32, y: String };
struct Bar { x: i32 };

fn main() {
  let x: Foo = Foo { x: 1, y: "hello".to_string() };

  // this would not compile
  // let y: Bar = x;
}
```

What makes Rust good?

1. Type inference

This is the same code

```
let list: Vec<u32> =
  vec![1u32, 2u32, 3u32].iter().map(|v: &u32| v +
1).collect::<Vec<u32>>()
```

```
let list =
  vec![1, 2, 3].iter().map(|v| v + 1).collect()
```

Type inference eliminates most noise.

Exceptions: function headers; ambiguity.

```
fn increment_and_dedup(v: Vec<u32>) -> HashSet<u32> {
  v.iter().map(|v| v + 1).collect()
}
```

2. Memory Safety

even in multi-threaded code

This fails to compile

```
fn main() {
  let mut data = vec![1, 2]; // allocate an array
  let first = &data[0];      // create an immutable ref
  data.push(4);              // attempt to mutate
}
```

`Vec::push()` takes a mutable reference, which needs to be exclusive.

Multi-threading type-safety

`trait Send`	safe to **send** to another thread
`trait Sync`	safe to **share** between threads

3. Powerful compile-time checks

Zero-cost abstractions

The ability to use higher-level features without incurring additional runtime cost.

The trade-off: compile-time complexity

“If it compiles, it works”

not to be taken literally

it’s how strongly typed programming **feels**

Making illegal states unrepresentable

Aim for compile-time enforcements instead of runtime validations

- Type-drive development;
- Abuse `Option`, `Result`, and `enum`;
- Typestate pattern.

Making illegal states unrepresentable

```
enum AccountState {
  Active { email: Email, active_at: DateTime },
  Inactive { email: Email },
  Banned { reason: String },
}

/// Newtype pattern
/// email regex can be enforced on constructor
/// runtime size is the same as String
type Email(String)
```

4. Tooling

```
cargo build
cargo run --package serve
cargo +nightly clippy
cargo fmt
cargo test
cargo build --target wasm32-unknown-unknown
cargo audit
bacon
```

Clippy is awesome

suspicious_arithmetic_impl

What it does

Lints for suspicious operations in impls of arithmetic operators, e.g. subtracting elements in an Add impl.

Why is this bad?

This is probably a typo or copy-and-paste error and not intended.

Example

```
impl Add for Foo {
    type Output = Foo;

    fn add(self, other: Foo) -> Foo {
        Foo(self.0 - other.0)
    }
}
```

Rust-analyzer <=> TS Server

Tips to get started

Don’t get too Rust’y right away

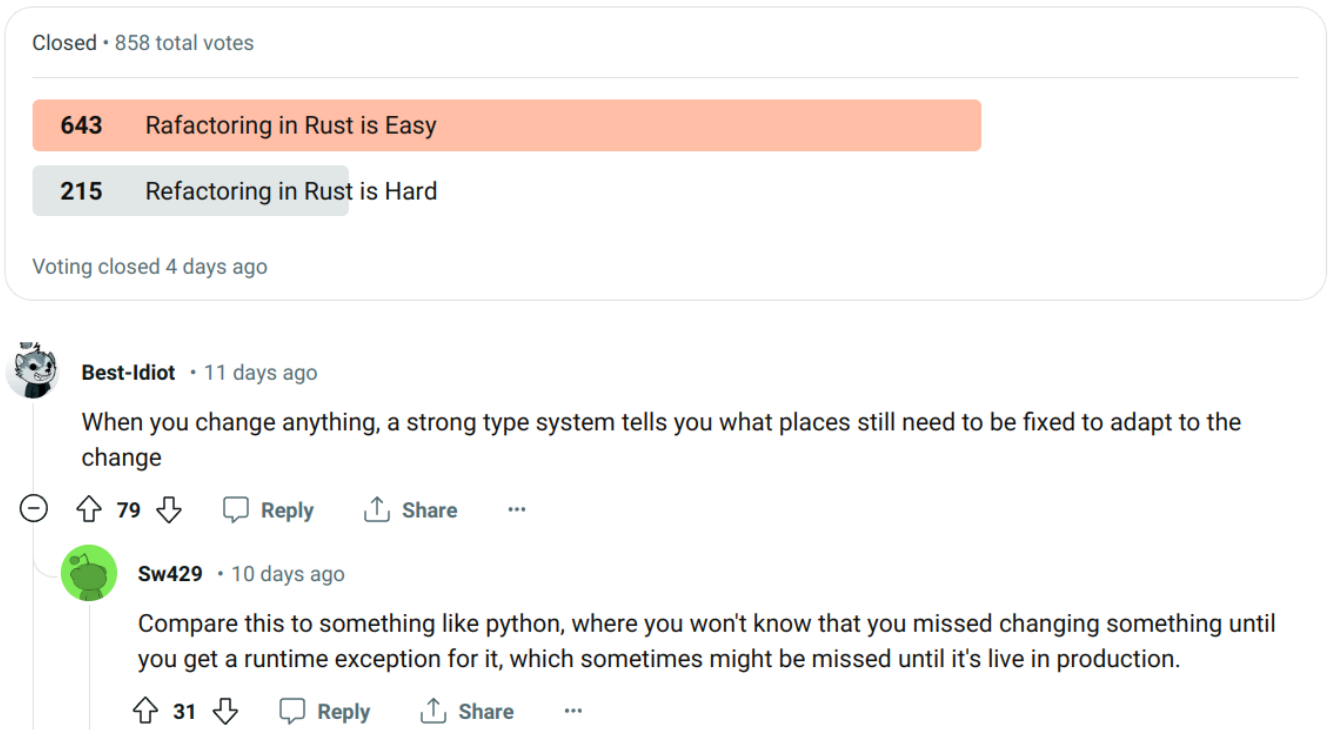
- if you’re writing `Foo<'a>`, you’re gonna have a bad time;
- Abuse `clone()` instead of fighting the borrow checker;
- get v1 working, only then optimize.
- tooling will teach you along the way

Why NOT Rust?

Compilation times?

- Incremental compilation is great(ish)
- Not quite instant-reload, but rather close
- Release builds are more painful
- You should `cargo check` instead of `cargo build`

Refactoring is a slog?



Rust

for JS/TS developers

Miguel Palhas / @naps62