JUGS #7L17M

RUST

2020-02-19
FOREWORD
WHO I AM
UWE KLOTZ

uklotzde
WHAT WE DO
SOFTWARE ENGINEERING
• General purpose IT systems - Full stack
• Industrial automation - IIoT
• Embedded/Real-time systems
CONSULTING & COACHING
• Systems architecture
• Software design
• Development processes
PERSONAL EXPERIENCE
(MAIN) PROGRAMMING LANGUAGES

> 20 years of C/C++

Java (JEE, Spring)

Rust (since 2018)
WHY RUST FOR ME?

- less complexity
- more control
- more efficiency
- more reliability
we tried to move faster in IT. we came up with microservices, which needed containers, which needed schedulers, which needed ... now we drown in added IT complexity, most of us not moving any faster, but at a higher risk. imo, we desperately need to strive for more simplicity ...

https://twitter.com/ufried/status/1164566228440104960
“Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.”

– Antoine de Saint-Exupéry, Airman’s Odyssey
TRY RUST

https://play.rust-lang.org/
fn main() {
    println!("Hello, world!");
}
# FREE FUNCTIONS

```rust
fn say_hello(to_whom: &str) {
    println!("Hello, {}!", to_whom);
}

fn main() {
    say_hello("JUGS");
    say_hello("iteratec");
}
```

Hello, JUGS!
Hello, iteratec!
fn say_hello(to_whom: &str) {
    println!("Hello, {}!", to_whom)
}

fn main() {
    say_hello("JUGS"); // string literal
    say_hello("iteratec");
}
RETURN TYPES

```rust
fn say_hello(to_whom: &str) -> () { // unit type
    println!("Hello, {}!", to_whom);
}
```
fn say_hello(to_whom: &str) {
    println!("Hello, {}!", to_whom);
}

fn main() {
    let recipients = ["JUGS", "iteratec",
                      ];
    for i in 0..recipients.len() { // range expression
        let who = recipients[i];
        say_hello(who);
    }
}
RANGE CHECKS

```rust
def say_hello(to_whom: &str) {
    println!("Hello, {}!", to_whom);
}

def main() {
    let recipients = [
        "JUGS",
        "iteratec",
    ];
    for i in 0..=recipients.len() {
        // inclusive upper bound
        let who = recipients[i];
        say_hello(who);
    }
}
```

thread 'main' panicked at 'index out of bounds: the len is 2 but the index is 2', src/main.rs:11:19
FOR LOOP SYNTAX

// expression evaluates to an iterator
for var in expression {
  code
}
let recipients = [
  "JUGS",
  "iteratec",
];

for who in &recipients {  // ≡ &recipients[..]
  say_hello(who);
}
ZERO COST ABSTRACTIONS

Incur no runtime penalty

At least as good as manual implementation
CLOSURES

```rust
for who in &recipients {
    say_hello(who);
}

recipients.iter().for_each(|who| say_hello(who));
```
let say_greetings = all.iter().map(|who| say_hello(who));

println!("Give a warm welcome to everyone");

// trigger evaluation
let greeted_count = say_greetings.count();

println!("Greeted {} recipient(s)", greeted_count);
let greeted_count = say_greetings.count();

let greeted_phrase = if greeted_count == 0 {
    "no recipients"
} else if greeted_count == 1 {
    "one recipient"
} else {
    debug_assert!(greeted_count > 1);
    "multiple recipients"
};
let greeted_phrase = match greeted_count {
  0 => "no recipients",
  1 => "one recipient",
  n if n > 5 => "many recipients",
  _ => "some recipients",
};
error[E0596]: cannot borrow `recipients` as mutable, as it is not declared as mutable
Module std::vec

```rust
datastruct Vec<T> {
    // private fields
}

impl<T> Vec<T> {
    /// Appends an element to the back of a collection.
    /// ...
    pub push(&mut self, value: T) {
        // MUTABLE BORROW of self
        // append `value` to `self`
    }
}
```
pub fn push(&mut self, value: T) [src]

Appends an element to the back of a collection.

Panics

Panics if the number of elements in the vector overflows a usize.

Examples

```rust
code-

let mut vec = vec![1, 2];
vec.push(3);
assert_eq!(vec, [1, 2, 3]);
```

let mut recipients = vec![
    "JUGS",
    "iteratec",
];

recipients.push("world");
THE PROGRAMMING LANGUAGE

...not the game!
MILESTONES

• 2006: Personal project by Graydon Hoare, Mozilla
• 2010: Public announcement
• 2015: v1.0 - Edition 2015
• 2018: v1.31 - Edition 2018
• 2019: v1.40 - async/await
STAKEHOLDERS

- Language development coordinated by Mozilla
- RFC community process
- Recently proposed: Independent Rust Foundation
WHAT IS RUST

- Multi-paradigm programming language
- Systems programming language
- Strongly typed
- Minimal/no runtime
Rust

A language empowering everyone to build reliable and efficient software.

Why Rust?

Performance
Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability
Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enable you to eliminate many classes of bugs at compile-time.

Productivity
Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

...take three.
RUST IS NOT OBJECT-ORIENTED

...at least not in the original sense.
NO INHERITANCE

Algebraic types

Functions

Methods

Traits

Generics
DYNAMIC DISPATCH

- Trait objects (opt-in)
- Require object-safe methods
VISIBILITY

Public

Private
RUST TOOLING
INSTALL RUST

curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

https://rustup.rs/
MANAGE TOOLCHAINS

$ rustup toolchain list
stable-x86_64-unknown-linux-gnu (default)
nightly-2020-01-23-x86_64-unknown-linux-gnu

Toolchain = Rust channel/version + Target
TOOLCHAIN - CHANNEL

Release cycle: Every 6 weeks

- **Stable channel**
  - Build using the Stable version: 1.41.0
- **Beta channel**
  - Build using the Beta version: 1.42.0-beta.3
  - (2020-02-07 86f329b419dbac59da59)
- **Nightly channel**
  - Build using the Nightly version: 1.43.0-nightly
  - (2020-02-18 e620d0f337d0643c757b)
WHICH CHANNEL?

- Use stable
- Use nightly only if you really need to
  - Example: Rocket.rs web framework
NIGHTLY TOOLCHAIN

Check component availability!

https://rust-lang.github.io/rustup-components-history/

<table>
<thead>
<tr>
<th>Component</th>
<th>2020-02-14</th>
<th>2020-02-13</th>
<th>2020-02-12</th>
<th>2020-02-11</th>
<th>2020-02-10</th>
<th>2020-02-09</th>
<th>2020-02-08</th>
<th>Last available</th>
</tr>
</thead>
<tbody>
<tr>
<td>cargo</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>2020-02-14</td>
</tr>
<tr>
<td>clippy</td>
<td>present</td>
<td>present</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>2020-02-14</td>
</tr>
<tr>
<td>lldb-preview</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>N/A</td>
</tr>
<tr>
<td>llvm-tools-preview</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>2020-02-14</td>
</tr>
<tr>
<td>miri</td>
<td>missing</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>2020-02-13</td>
</tr>
<tr>
<td>rls</td>
<td>present</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>2020-02-14</td>
</tr>
<tr>
<td>rust</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>2020-02-14</td>
</tr>
<tr>
<td>rust-analysis</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>2020-02-14</td>
</tr>
<tr>
<td>rust-docs</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>2020-02-14</td>
</tr>
<tr>
<td>rust-mingw</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>N/A</td>
</tr>
<tr>
<td>rust-src</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>2020-02-14</td>
</tr>
<tr>
<td>rust-std</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>2020-02-14</td>
</tr>
<tr>
<td>rustc</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>2020-02-14</td>
</tr>
<tr>
<td>rustc-dev</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>present</td>
<td>2020-02-14</td>
</tr>
<tr>
<td>rustfmt</td>
<td>present</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>missing</td>
<td>2020-02-14</td>
</tr>
</tbody>
</table>
EDITIONS

Release cycle: As needed, years

```
[package]
name = "tokio-modbus"
edition = "2018"
```
TOOLCHAIN - TARGET TRIPLE(T)

- **Machine**: i686, x86_64, aarch64, wasm32, …
- **Vendor**: apple, pc, unknown, …
- **OS**: darwin, ios, windows-msvc, linux-gnu, linux-musl, unknown, …
PROJECT TASKS
WHAT DO YOU WANT TO DO?

Run
Build and run the code, showing the output. Equivalent to `cargo run`.

Build
Build the code without running it. Equivalent to `cargo build`.

Test
Build the code and run all the tests. Equivalent to `cargo test`.

ASM
Build and show the resulting assembly code.

LLVM IR
Build and show the resulting LLVM IR, LLVM’s Intermediate representation.

MIR
Build and show the resulting MIR, Rust’s Intermediate representation.

WASM
Build a WebAssembly module for web browsers, in the `.WAT` textual representation.
CARGO CLI

- Resolve dependencies
- Build projects
- Run unit tests
- Generate documentation (incl. example code)
- Format source code
- Check coding style
- …extensible…
[package]
name = "semval"
version = "0.1.3"
license = "Apache-2.0 OR MIT"
authors = ["slowtec GmbH <post@slowtec.de>"
repository = "https://github.com/slowtec/semval"
edition = "2018"

[dependencies]
smallvec = "1"

[features]
default = ["std"]
std = []
IDE SUPPORT - YMMV

- VS Code with extensions
  - Rust (rls)
  - rust-analyzer
- JetBrains IntelliJ Rust
  - “…is in active development”
- Vim, Emacs, …
enum Option<T> {
    None,
    Some(T),
}
fn min_elem<T, I>(iterable: I) -> Option<T>
where
T: Ord,
I: IntoIterator<Item = T>,
{
    iterable.into_iter().fold(None, |min_val, val| {
        let min_val = if let Some(min_val) = min_val {
            min_val.min(val)
        } else {
            val
        };
        Some(min_val)
    })
}
NO EXCEPTIONS

```java
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

Return either success or failure result

Results **must** be handled by caller
RESULT IN ACTION

use std::{fs::File, io::{BufReader, Read, Result}};

//type Result<T> = Result<T, std::io::Error>;

fn read_text_file(filename: &str) -> Result<String> {
    let file = File::open(filename)?;
    let mut text = String::new();
    BufReader::new(file).read_to_string(&mut text)?;
    Ok(text)
}
async fn read_text_file(filename: &str) -> Result<String> {
    let file = File::open(filename).await?;
    let mut text = String::new();
    BufReader::new(file).read_to_string(&mut text).await?;
    Ok(text)
}

- async keyword before function declaration
- .await behind every inner async function call
MOVE SEMANTICS

// Dynamically allocated strings
let mut s1 = String::from("foo");
let s2 = s1;

// error[E0382]: borrow of moved value: `s1`
s1 += "bar";
let mut s1 = String::from("foo");
let s2 = s1.clone();
s1 += "bar";

assert_eq!("foobar", s1);
assert_eq!("foo", s2);
ESCAPE MOVE SEMANTICS

- Clone = “duplicable”
- Copy = trivially copyable
WHAT MORE?
PRODUCT TYPES

```rust
struct Point {
    x: f64,
    y: f64,
}

// tuple
type MyTuple = (u32, f64, String);

// tuple struct
struct GeoCoord(f64, f64, f32);
```
impl TRAITS FOR TYPES

- Own trait on own type ✓
- Own trait on foreign type ✓
- Foreign trait on own type ✓
- Foreign trait on foreign type ✗
DESTRUCTURING

...to anticipate future changes.

Great tool to improve maintainability.
LIFETIMES

...of references/borrows.

Compiler was able to infer all lifetimes for us.
FEARLESS CONCURRENCY

• Types are tagged for thread-safety
  ▪ Send - transfer between threads
  ▪ Sync - safely use between threads

• Synchronization primitives built around types

• Channels: Either built-in or from libraries

• Synchronous & asynchronous
INTEROPERABILITY

Foreign Function Interface (FFI)

Rust ⇔ C ⇔ Rest of the world
HOW TO “FIGHT” THE BRORROW CHECKER?
You shouldn’t need to.
unsafe
WHY NOT RUST

• Long build times …*	extit{compared to Go}*
• Type system has restrictions …*	extit{still evolving}*
• Hierarchical data structures with cross-/cyclic-/self-references …*	extit{explore new design patterns}*
• Desktop UI applications …*	extit{mostly wrappers around C/C++ framework}*
“THE BOOK”

The Rust Programming Language (Covers Rust 2018)

Or read online:

LINKS

- Rust by Example
- Rust Cookbook
- Rust Language Cheat Sheet
- Recent blog post: Rust for professionals
Uwe Klotz
@uclotzde
uwe.klotz@slowtecc.de

slowtecc.de