Why You Should Learn Rust

Even a JavaScript developer can benefit from learning Rust

Outline

1. Rust Features
2. Rust Ecosystem
3. Rust In The Wild

What is Rust

A popular survival MMO video game.

What is Rust

- A system language.
- Some called Rust "Pragmatic Haskell".
- Produtivity, performance, safety. Take three.
- Wins "most loved language" title in 3 consecutive years. (Stackoverflow Developer Survey 2016, 2017, 2018)

What is a system language

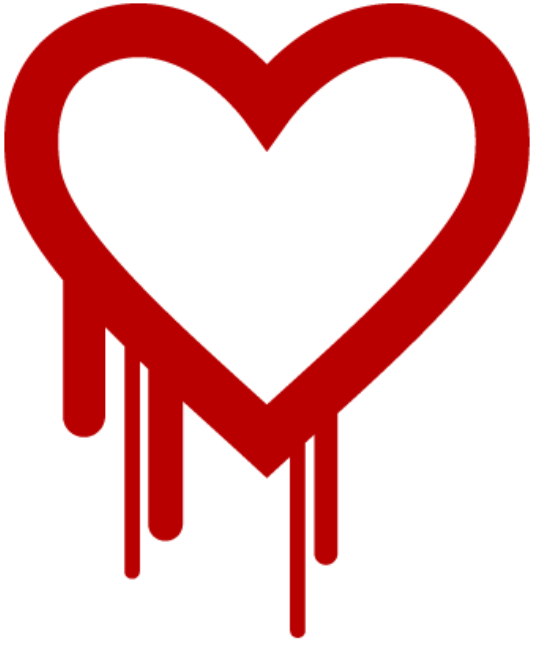
- Does system programming
  - Operating systems
  - Device drivers
  - Compilers
  - Embedded systems
- Minimal runtime should be considered
- C, C++, ~~Go~~[1]

> 1: 为什么微软不出一门像go或者rust的跨平台系统级语言?以它的实力完全可以做得到的?

Why not stick with C/C++

Why not stick with C/C++

OpenSSL heartbleed



Before started

You can copy-paste code and play around with Rust Playground.

Rust Features

- **Memory safety (without GC)**
- **No data race**
- Minimal runtime
- High-level abstraction
- **Great build system**
- Powerful macro

Define Memory Safety

- No null pointer dereferences
- No dangling pointers
  - No double free
  - No use-after-free
- No out-of-bound-access
  - No buffer overflow
- No data race??

Wiki: Memory safety

Unsafe in C++

```cpp
#include <string>
#include <iostream>

using std;

int main() {
    std::vector<std::string> v;

    v.push_back("Hello, ");

    std::string &x = v[0];

    v.push_back("world!"); // Problem occurs

    std::cout << x;
}
```

... and cannot compile this snippet on my macbook

Memory Safety in Rust

```
fn main() {
    let mut s = Vec::new();

    s.push("Hello, ");

    let x = &s[0];

    s.push("world!");

    println!("{:?}", s);
}
```

```
error[E0502]: cannot borrow `s` as mutable because it is also borrowed
                  as immutable
  --> src/main.rs:8:5
6  |     let x = &s[0];
   |              - immutable borrow occurs here
7  |
8  |     s.push("world!");
   |     ^ mutable borrow occurs here
...
11 | }
   | - immutable borrow ends here
```

Memory Safety in Rust

Forced initialization + Ownership + Restricted aliasing

= Memory safety

all at compile time!

Forced Initialization

```
fn main() {
    let x: i32;
}
```

Compiled without crashes.

```
fn main() {
    let x: i32;
    println!("{}", x);
}
```

```
error[E0381]: use of possibly uninitialized variable: `x`
 --> src/main.rs:3:20
  |
3 |     println!("{}", x);
  |                    ^ use of possibly uninitialized `x`
```

Forced Initialization

```
fn main() {
    let x: i32;
    if true {
        x = 10;
    } else {
        x = 20;
    }
    println!("{}", x);
}
```

Initialize `x` before using. Compiled without surprises.

What about null pointer?

Rust has no null pointers

Instead, Rust borrows the concepts Option type from functional programming.

An option type encapsulates null inside a container, forcing programmers to **think about the nullability before implementing**.

Without Option type, every single varible could be null and cause null pointer exceptions.

Option<T>

Option is just an enum in Rust.

```
pub enum Option<T> {
    None,
    Some(T),
}
```

Pattern matching with `Option` as an normal enum is like a breeze.

```
fn main() {
    let maybe = Some(2);
    match maybe {
        Some(p) => println!("has value {}", p),
        None => println!("has no value"),
    }
    // or `maybe.unwrap()`
}
```

```
> has value 2
```

Ownership

Rust Ownership Rules

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 (RAII).

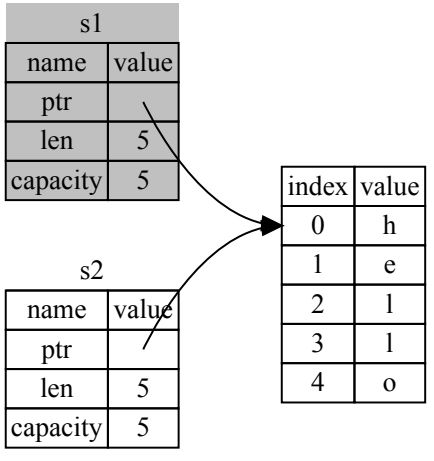
> Rust guarantees its memory safety by restricting variables from aliasing.

Play around with this playgournd

Move semantics

```
let s1 = String::from("hello");
let s2 = s1; // s2 is the new owner.

println!("{}, world!", s1); // Error. Value is moved.
```



Scope and RAII

```
fn make_string() {
    let s = "hello";   // s is valid from this point forward
    println!("{}", s);
}

fn main() {
    make_string();
    // the scope of make_string is over.
    // s is dropped immediately. No GC.
}
```

How to share data between variables?

Answer: resctrict aliasing with borrow checker.

> This is the one of the most intimidating part of Rust.

Borrow

| ˈbɒrəʊ |

verb

Take and use (something belonging to someone else) with the intention of **returning** it

Borrow Checker

To guarantee data synchronization and prevent data racing, all references in Rust must follow at lease two rules:

- Having
  - several immutable references (`&T`) or
  - exact one mutable reference (`&mut T`).
- A reference must always be valid even it references to null.
  (use `Option<T>` representing null pointer)

> Q: When will the borrowing returned from a reference?
> A: A reference would return its borrowing when it goes out of scope (RAII).

Borrow Checker: several immutable refs

```
fn main() {
    let s = String::from("hello");
    let r1 = &s;
    let r2 = &s;
    let r3 = &s;
    println!("{} {} {}", r1, r2, r3);
}
```

```
> hello hello hello
```

Borrow Checker: exact one mutable ref

```
fn main() {
    let mut s = String::from("hello");

    {
        let r1 = &mut s;
        println!("r1 {}", r1);
    } // r1 returns the borrowing, so we can make a new reference.

    let r2 = &mut s;
    println!("r2 {}", r2);
}
```

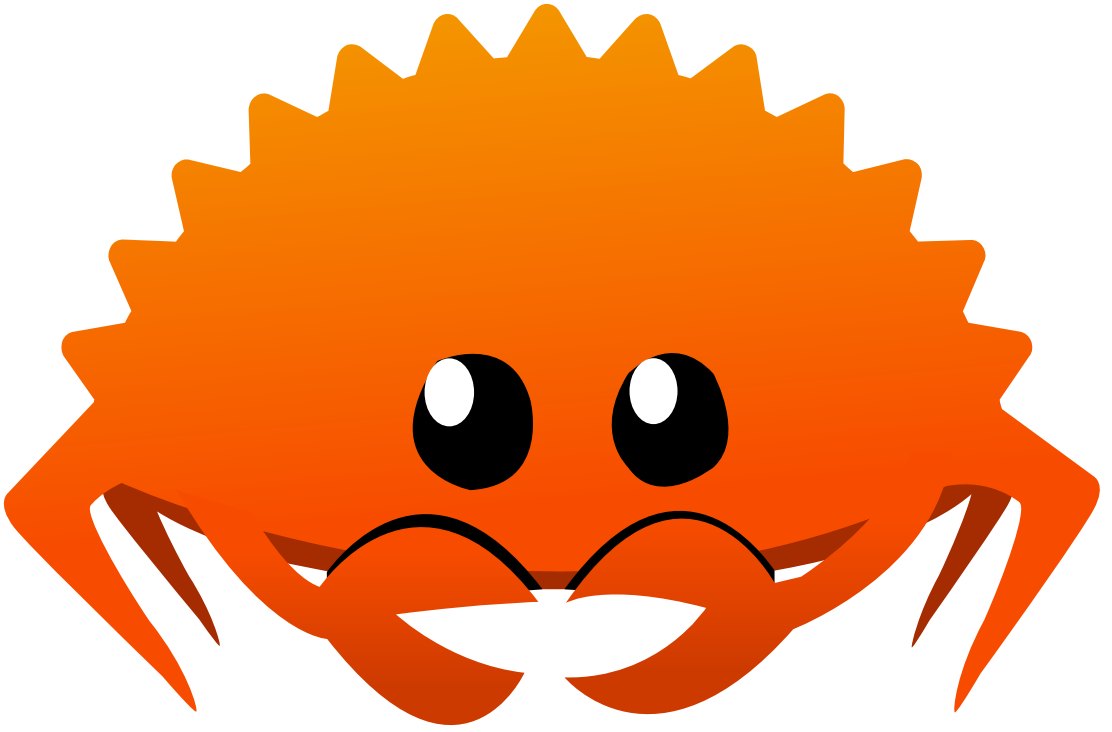
Borrow Checker: cannot mix mutable with immutable refs

```
fn main() {
    let mut s = String::from("hello");

    let r1 = &s; // no problem
    let r2 = &s; // no problem
    let r3 = &mut s; // BIG PROBLEM
}
```

```
error[E0502]: cannot borrow `s` as mutable because it is also borrowed as
immutable
 --> borrow_thrice.rs:6:19
  |
4 |     let r1 = &s; // no problem
  |              - immutable borrow occurs here
5 |     let r2 = &s; // no problem
6 |     let r3 = &mut s; // BIG PROBLEM
  |                   ^ mutable borrow occurs here
7 | }
  | - immutable borrow ends here
```

Hey, Ferris



Make your own Ferris

Rust Ecosystem

crates.io: a centralized package registry hosted by Rust core team.

Rust Ecosystem

Docs.rs: a unofficial documentation site for public crates.

Rust Frontend frameowrk

Yew

```
impl Renderable<Model> for Model {
    fn view(&self) -> Html<Self> {
        html! {
            // Render your model here
            <button onclick=|_| Msg::DoIt,>{ "Click me!" }</button>
        }
    }
}

fn main() {
    yew::initialize();
    App::<Model>::new().mount_to_body();
    yew::run_loop();
}
```

Rust ORM

Diesel.rs

Made by author of Active Record (Ruby)

```
let versions = Version::belonging_to(krate)
  .select(id)
  .order(num.desc())
  .limit(5);
let downloads = try!(version_downloads
  .filter(date.gt(now - 90.days()))
  .filter(version_id.eq(any(versions)))
  .order(date)
  .load::<Download>(&conn));
```

Rust Ecosystem

- Full-fledged Operating system
- WebAssembly (Rust Wasm team)
- Awesome Embedded Rust

Rust In The Wild

Rust is everywhere in your daily life.

- Microsoft
  - Azure IoT Edge
  - VS Code search (ripgrep)
- Sentry: Fixing Python Performance with Rust
- Dropbox
  - Optimizing cloud file-storage
  - Better compression with DivANS
- npm registry: replacing C and rewriting performance-critical bottlenecks in the registry service architecture
- Mercurial RIIR and Facebook is doing that.
- Atlassian: service for analyzing petabytes of source code.
- Cloudflare: replacement for memory-unsafe languages and core edge logic
- Mozilla: Firefox Quantum and Servo
- Canonical (Unbuntu): from server monitoring to middleware.
- Baidu X-Lab 百度安全实验室: Rust SGX SDK and memory-safe and OpenSSL-compatible TLS

More on Friends of Rust.

Community

- Reddit Rust
- Rust Forum
- http://rustacean.net/

Random Good Stuff

- University courses
  - Stanford CS140e
  - UVA CS4414
  - UPenn CIS198
- Online resources
  - Official Rust Book
  - Official Rust Book - 簡體中文翻譯版
  - electronic_blue: 給 C++ 使用者的 Rust 簡介
  - Rust By Example
  - Nick Cameron: Rust for C++ programmers
- Video Talks
  - Sergio Benitez: A Case for Oxidation (Author of Rocket framework)