How to apply Rust in Real World?

Hiroshi Hatake

Technical information sharing seminar

Introduction

What is Rust?

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

Introduction

What is Rust?

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

- zero-cost abstractions

Introduction

What is Rust?

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

- zero-cost abstractions
- guaranteed memory safety

Introduction

What is Rust?

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

- zero-cost abstractions
- guaranteed memory safety
- threads without data races

Introduction

What is Rust?

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

- zero-cost abstractions
- guaranteed memory safety
- threads without data races
- trait-based generics

Introduction

What is Rust?

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

- zero-cost abstractions
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching

Introduction

What is Rust?

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

- zero-cost abstractions
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inference

How to apply Rust in real world – Motivation

Motivation

- Rust runs fast and guarantee safety.

How to apply Rust in real world – Motivation

Motivation

- Rust runs fast and guarantee safety.
- And, Rust runs on Windows same as *nix systems.

An example – grnenv-rs

grnenv – switch Groonga versions

grnenv is one of the switching Groonga version tool.

[1]Perhaps, it can handle executables for Windows 95.

An example – grnenv-rs

grnenv – switch Groonga versions

grnenv is one of the switching Groonga version tool.

- It requires bash.

[1]Perhaps, it can handle executables for Windows 95.

An example – grnenv-rs

grnenv – switch Groonga versions

grnenv is one of the switching Groonga version tool.

- It requires bash.
- It assumes that Linux environment.

[1] Perhaps, it can handle executables for Windows 95.

An example – grnenv-rs

grnenv – switch Groonga versions

grnenv is one of the switching Groonga version tool.

- It requires bash.
- It assumes that Linux environment.
- But, I want to use switching tool like this on Windows.

[1] Perhaps, it can handle executables for Windows 95.

An example – grnenv-rs

grnenv – switch Groonga versions

grnenv is one of the switching Groonga version tool.

- It requires bash.
- It assumes that Linux environment.
- But, I want to use switching tool like this on Windows.
- If possible, I want to create **executables**. Because Windows can handle PE(portable executables) for Windows 2k or older.[1]

[1] Perhaps, it can handle executables for Windows 95.

Rust is suitable for this purpose?

Rust is suitable for this purpose?

Halfly yes, halfly not. Because....

Rust is suitable for this purpose?

Rust is suitable for this purpose?

Halfly yes, halfly not. Because....

- It requires Visual C++ 2015 Runtime. No more needed.

Rust is suitable for this purpose?

Rust is suitable for this purpose?

Halfly yes, halfly not. Because....

- It requires Visual C++ 2015 Runtime. No more needed.
- It can create executables.

Rust is suitable for this purpose?

Rust is suitable for this purpose?

Halfly yes, halfly not. Because....

- It requires Visual C++ 2015 Runtime. No more needed.
- It can create executables.
- But, it is hard to handle **lifetime**.

Rust is suitable for this purpose?

Rust is suitable for this purpose?

Halfly yes, halfly not. Because....

- It requires Visual C++ 2015 Runtime. No more needed.
- It can create executables.
- But, it is hard to handle **lifetime**.
- Also, It is hard to handle and extend Trait in some cases.

Rust has difference type system than ever.

Rust can represent abnormal value in type.

Rust has **Option** and **Result** types.

Rust has difference type system than ever.

Rust can represent abnormal value in type.

Rust has **Option** and **Result** types.

- **Option** can contain normal value within “**Some**” and **None** which represents “abnormal value” like as “NULL”.

Rust has difference type system than ever.

Rust can represent abnormal value in type.

Rust has **Option** and **Result** types.

- **Option** can contain normal value within “**Some**” and **None** which represents “abnormal value” like as “NULL”.
- **Result** can contain normal value in “**Ok**” and “**Err**” which can contain error information.

What is benefits for these types?

I will show a few questions.

What is benefits for these types?

Will this function return abnormal value?

```rust
pub fn read_dir<P: AsRef<Path>>(path: P) -> ?
```

What is benefits for these types?

First. Will this function return abnormal value?
Yes. It returns **Result**.

```
type Result<T> = Result<T, Error>;
pub fn read_dir<P: AsRef<Path>>(path: P)
  -> std::io::Result<ReadDir>
```

What is benefits for these types?

Second. Will this function return abnormal value?

```
// write all buffer into writing target.
fn write_all(&mut self, buf: &[u8]) -> ?
```

What is benefits for these types?

Second. Will this function return abnormal value?
Yes. It returns **Result**.

```
fn write_all(&mut self, buf: &[u8]) -> Result<()>
```

What is benefits for these types?

Third. Will this function return abnormal value?

```
// obtain user's home directory.
pub fn home_dir() -> ?
```

What is benefits for these types?

Third. Will this function return abnormal value?
Ofcource, Yes!!!
Because $HOME always does not exist.
When without sudo -E or using more tighten sudo settings, $HOME cannot obtain.

```
pub fn home_dir() -> Option<PathBuf>
```

A person says....

The three laws of disallow NULL.[2]

[2] https://twitter.com/gakuzzzz/status/783616563102388224

A person says....

The three laws of disallow NULL.[2]

- Don't receive NULL

[2] https://twitter.com/gakuzzzz/status/783616563102388224

A person says....

The three laws of disallow NULL.[2]

- Don't receive NULL
- Don't return NULL

[2]https://twitter.com/gakuzzzz/status/783616563102388224

A person says....

The three laws of disallow NULL.[2]

- Don't receive NULL
- Don't return NULL
- Don't write NULL

[2]https://twitter.com/gakuzzzz/status/783616563102388224

A person says....

The three laws of disallow NULL.[3]
For Rust version.

[3] https://twitter.com/gakuzzzz/status/783616563102388224

A person says....

The three laws of disallow NULL.[3]
For Rust version. Rust does not have NULL, Yay!
But Rust has a few danger things like as unwrap().

[3] https://twitter.com/gakuzzzz/status/783616563102388224

A person says....

The three laws of disallow NULL.[3]
For Rust version. Rust does not have NULL, Yay!
But Rust has a few danger things like as unwrap().

- Don't use unwrap()

[3] https://twitter.com/gakuzzzz/status/783616563102388224

A person says....

The three laws of disallow NULL.[3]
For Rust version. Rust does not have NULL, Yay!
But Rust has a few danger things like as unwrap().

- Don't use unwrap()
- Don't dismiss error values

[3]https://twitter.com/gakuzzzz/status/783616563102388224

A person says....

The three laws of disallow NULL.[3]
For Rust version. Rust does not have NULL, Yay!
But Rust has a few danger things like as unwrap().

- Don't use unwrap()
- Don't dismiss error values
- Don't use panic! if it cannot recover

[3]https://twitter.com/gakuzzzz/status/783616563102388224

For safety

Rustish guaranting ways.

For safety

Rustish guaranting ways.

- Don't use unwrap()

```
let f = fs::remove_file(shim_dir.join("source-groonga.sh"))
          .unwrap();
```

For safety

Rustish guaranting ways.

- Don't use unwrap(). Use **try!**.

```
let f = try!(fs::remove_file(shim_dir.join("source-groonga.sh")));
```

For safety

Rustish guaranting ways.

- Don't dismiss error values

```
let _ = f.write_all(&contents.as_bytes());
f.sync_data()
```

For safety

Rustish guaranting ways.

- Don't dismiss error values. Use **pattern matching** to handle error.

```
match f.write_all(&contents.as_bytes()) {
  Ok(_) => return Ok(()),
  Err(e) => println!("{}", e),
}
f.sync_data()
```

For safety

Rustish guaranting ways.

- Don't use panic! if it cannot recover.

```
let home = env::home_dir().unwrap();
```

For safety

Rustish guaranting ways.

- Don't use panic! if it cannot recover.

```
let home = env::home_dir()
            .unwrap_or_else(|| panic!("Cound not found homedir."));
```

Someone would think as....

OK. I studied Rustish guaranting ways.
Always using Option and Result for abnormal values makes everything OK.

Someone would think as....

Answer: Sadly, No.
Current Rust does not handle Abstract data type in main() function. It can handle only i32(=normally, int) type values, like ..., -2, -1, 0, 1, 2, ...[4]

[4] This is intended behaviour. see also: https://github.com/rust-lang/rust/issues/12130#issuecomment-34583413

Conclusion

- Rust works on Windows same as *nix.
- Rust does not have NULL in concept.
- Rust has some of danger method like as unwrap().
- Using Option and Result is better in most cases.
- Some of places is not usable Result type.

Happy without NULL life with Rust! Enjoy!!

Any questions?