Rust & Trait

—

By Sun@CypherLink

Who AM I ?

Sun

CypherLink 联合创始人
TDN 作者
RUST.CC 最菜的 

A cryptographer, programmer, idealist. 一位喜欢开源文化和社区的程序员，目前付诸精力于密码学和区块链的研究和应用落地。 2019 年联合成立了 CypherLink (cypherlink.io) 组织，探索解决数据安全和隐私保护问题。

Github: sunhuachuang

主题

1. 什么是 Rust 语言
2. Rust 语言的发展
3. Rust 语言的特性
4. Rust 语言的 Trait

什么是 Rust 语言

A language empowering everyone to build reliable and efficient software.

一门赋予每个人构建可靠且高效软件能力的语言。

通用型 | 编译型 | 强类型 | 多范式

常用文件后缀： .rs .rlib

官网： rust-lang.org

仓库： github.com/rust-lang

中文社区： rust.cc

Rust 语言的发展

Rust 语言原本是 Mozilla 员工 Graydon Hoare 的私人项目；

Mozilla 于 2009 年开始赞助这个项目，并且在 2010 年首次揭露了它的存在；

第一个有版本号的 Rust 编译器于 2012 年 1 月发布。 Rust 1.0 于 2015 年 5 月 15 日发布；

现在 Rust 由 Mozilla 支持，社区协作发展，总体来说，属于社区项目；

每 3 个月发布一次新版本： Stable ， Beta ， Nightly 。

Rust 语言的特性

1 高性能

Rust 速度惊人且内存利用率极高。

- 编译成可执行文件
- 内存利用率高
- 没有 GC
- 支持嵌入式
- 轻松与其他语言集成

2 可靠性

Rust 让您在编译期就能够消除各种各样的错误。

- 内存安全
- 线程安全

3 生产力

生产力工具包丰富。

- Rustc 编译器错误提示
- Rust Doc 自动化文档
- Cargo 包管理与构建
- rls 智能编辑器支持
- clippy 错误和改进建议
- rustfmt 自动格式化

Rust 语言的特性

所有权与生命周期

宏 - macro_rules / derive

模式匹配 - match

类型推导 - let

类型系统 - trait / impl

异步 - async / await

测试与性能 - test / bench

枚举 - enum

面向对象式

结构体 - struct

智能指针 - Box / Rc / Weak / Deref / RefCell

函数式 - fn

基础类型

错误处理 - panic / Result (Option)

...

Rust 语言中的 Trait

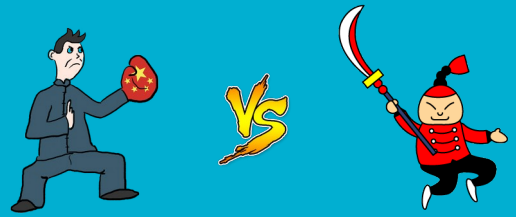
抽象

Rust 语言中的 Trait

抽象

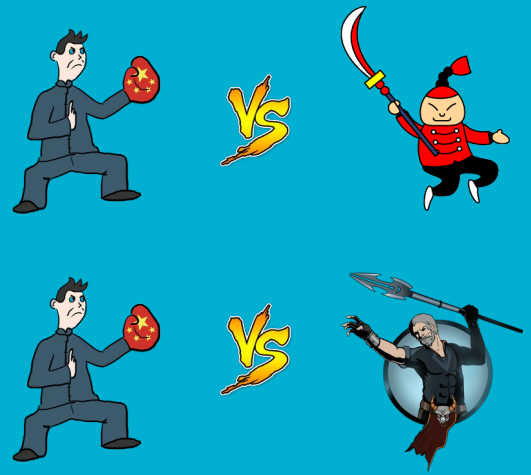
抽象就是把复杂的东西简单化。

Rust 语言中的 Trait



灰色衣服的红色衣服的在比武

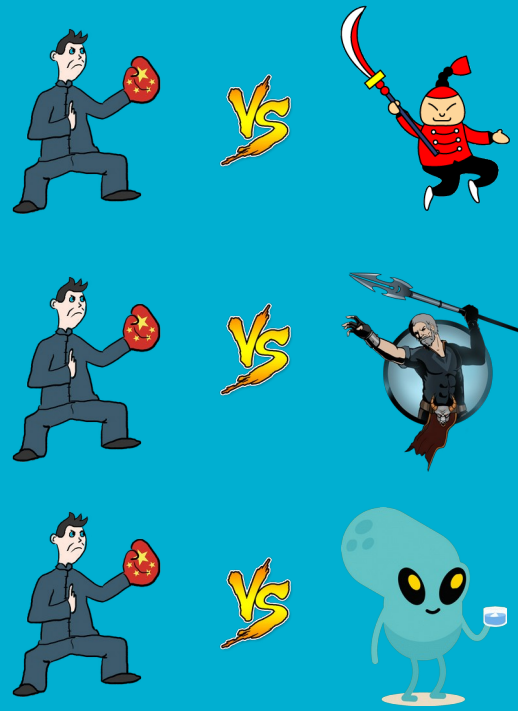
Rust 语言中的 Trait



灰色衣服的红色衣服的在比武

中国人和外国人在比武

Rust 语言中的 Trait



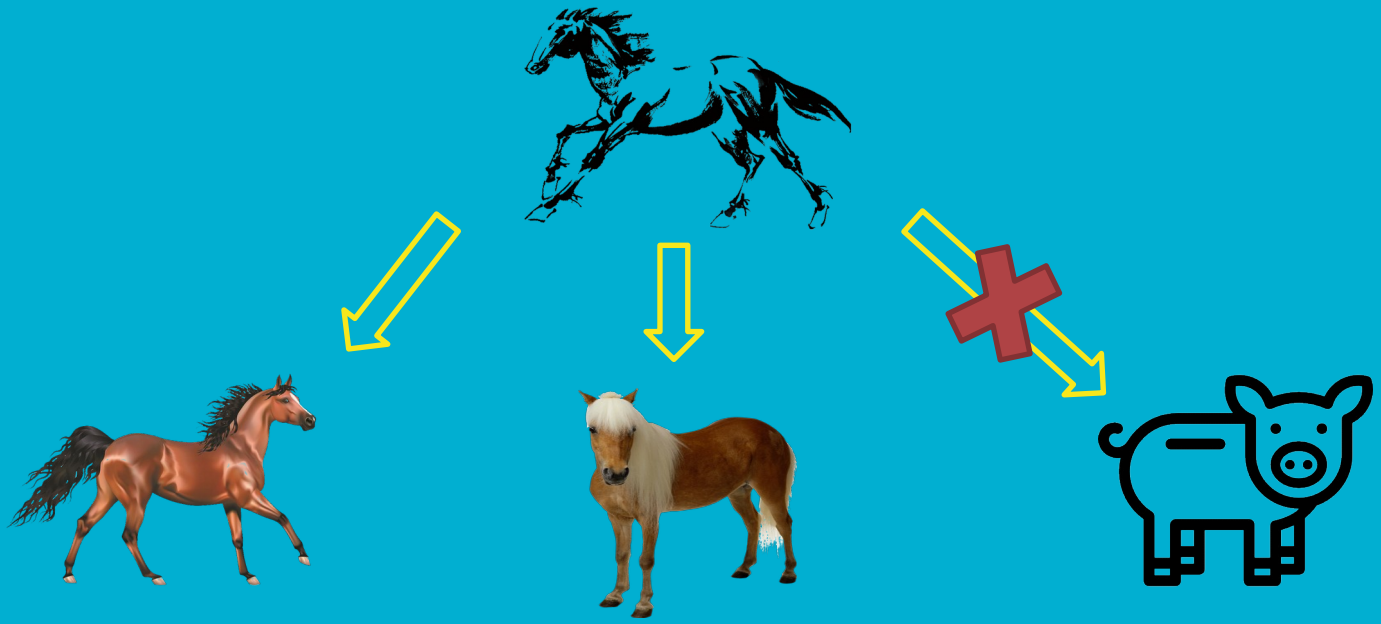
灰色衣服的红色衣服的在比武

中国人和外国人在比武

地球人和火星人在比武

Rust 语言中的 Trait

Rust 语言中的 Trait



Rust 语言中的 Trait

```rust
#![allow(dead_code)]

trait See {
    fn see(color: Color, age: u8) -> Self;
}

#[derive(Debug)]
enum Color {
    Red,
    White,
    Blue,
}

struct Horse {
    color: Color,
    age: u8,
}

impl See for Horse {
    fn see(color: Color, age: u8) -> Self {
        Horse { color, age }
    }
}

fn main() {
    let horse = Horse::see(Color::Red, 2);
    println!("Horse color: {:?}, age: {}", horse.color, horse.age);
}
```

Rust 语言中的 Trait

泛型约束

Rust 语言中的 Trait

师者传道授业解惑也。
与一位自称老师的人交流的时候，那么就可以直接向他提出疑惑

他就天然具有 ask 方法

Rust 语言中的 Trait

```
type Answer = String;

enum Question {
    Language(String),
    Math(String),
    English(String),
}

trait  Teacher {
    fn ask(&self, q: Question) -> Option<Answer>;
}
```

Rust 语言中的 Trait

```
struct LanguageTeacher;

impl  Teacher  for  LanguageTeacher {
    fn ask(&self, q: Question) -> Option<Answer> {
        match q {
            Question::Language(s) => Some(s),
            _ => None,
        }
    }
}
```

Rust 语言中的 Trait

```
struct MathTeacher;

impl Teacher for MathTeacher {
    fn ask(&self, q: Question) -> Option<Answer> {  ...  }
}

struct EnglishTeacher;

impl Teacher for EnglishTeacher {
    fn ask(&self, q: Question) -> Option<Answer>  { ... }
}
```

Rust 语言中的 Trait

```
fn ask1<T: Teacher>(q: Question, teacher: T) -> Option<Answer> {
    teacher.ask(q)
}

fn ask2(q: Question, teacher: impl Teacher) -> Option<Answer> {
    teacher.ask(q)
}

fn ask3(q: Question, teacher: Box<dyn Teacher>) -> Option<Answer> {
    teacher.ask(q)
}

fn main() {
    let answer1 = ask1(Question::Language(" 语文 ".to_owned()), LanguageTeacher);
    let answer2 = ask2(Question::Math("1+1".to_owned()),          MathTeacher);
    let answer3 = ask3(Question::English("code".to_owned()),   Box::new(MathTeacher));
}
```

静态分发

静态分发

动态分发

Rust 语言中的 Trait

```
trait Teacher : Debug + Clone {
    fn ask(&self, q: Question) -> Option<Answer>;
}
```

trait 的 trait 约束 :
实现 Teacher 特性的对象，必须同时实现 Debug 和 Clone 特性

Rust 语言中的 Trait

继承与多态

Rust 语言中的 Trait

Rust 语言中的 Trait

```
trait Father {
    fn money(input: u32) -> u32 {
        input - 1
    }
}

trait Mother {
    fn age(&self) -> u32 {
        18 // comment: always
    }
}
```

Rust 语言中的 Trait

```
struct Son {
    age: u32,
}

impl Father for Son {}

impl Mother for Son {}

fn main() {
    let me = Son { age: 20 };
    assert_eq!(9, Son::money(10));
    assert_eq!(18, me.age());
}
```

Rust 语言中的 Trait

Rust 语言中的 Trait

```
trait Mother {
    fn money(input: u32) -> u32 {
        input + 1
    }
}
```

Rust 语言中的 Trait

```
trait Mother {
    fn money(input: u32) -> u32 {
        input + 1
    }
}
```

```
error[E0034]: multiple applicable items in scope
  --> src/main.rs:27:20
   |
27 |     assert_eq!(18, Son::money(10));
   |                    ^^^^^^^^^^ multiple `money` found
   |
note: candidate #1 is defined in an impl of the trait `Father` for the type `Son`
  --> src/main.rs:2:5
   |
2  |     fn money(input: u32) -> u32 {
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^
   = help: to disambiguate the method call, write `Father::money(...)` instead
note: candidate #2 is defined in an impl of the trait `Mother` for the type `Son`
  --> src/main.rs:12:5
   |
12 |     fn money(input: u32) -> u32 {
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^
   = help: to disambiguate the method call, write `Mother::money(...)` instead
```

Rust 语言中的 Trait

Rust 语言中的 Trait

```
impl Father for Son {
    fn money(input: u32) -> u32 {
        input + 1
    }
}

assert_eq!(11, Son::money(10));
```

覆盖，重新实现

Rust 语言中的 Trait

Rust 语言中的 Trait

关联类型

```
trait Father {
    type Money;
    fn money(input: Self::Money) -> Self::Money;
}

struct RMB(u32);
impl Father for Son {
    type Money = RMB;
    fn money(input: Self::Money) -> Self::Money { RMB(input.0 - 1) }
}
```

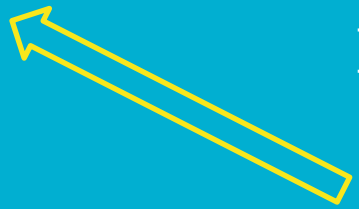
Rust 语言中的 Trait

Rust 语言中的 Trait

```
use std::fmt::Debug;
trait Father {
    type Money: Debug;
    fn money(input: Self::Money)
        -> Self::Money
    {
        println!("{:?}", input);
        input
    }
}
```

```
#[derive(Debug)]
struct RMB(u32);

impl Father for Son {
    type Money = RMB;
}
```



关联类型的特性约束

Rust 语言中的 Trait

Rust 语言中的 Trait

复杂关联类型的特性约束

```
use std::ops::Add;

trait Father {
    type Money: Add<Output = Self::Money> + Copy;

    fn money(input: Self::Money) -> Self::Money {
        input + input
    }
}
```

Rust 语言中的 Trait

```
#[derive(Eq, PartialEq, Ord, PartialOrd, Debug, Copy, Clone)]
struct RMB(u32);

impl Add for RMB {
    type Output = Self;

    fn add(self, other: RMB) -> Self {
        RMB(self.0 + other.0)
    }
}
```

Rust 语言中的 Trait

Rust 语言中的 Trait

```
#![feature(associated_type_defaults)]

trait Father {
    type Money = RMB;

    fn money(input: RMB) -> RMB {
        input + input
    }
}

impl Father for Son {}
```

关联类型的默认值

Rust 语言中的 Trait

幸福一家人

QA

RUST.CC

Rust语言中文社区

https://rust.cc

https://rust-china.org

CypherLink

微信公众号

https://cypherlink.io