ORGANIZER: GFI VBI | POWERED BY: WEB3HACKFEST | DEV PARTNER: CODE mely VU NGUYEN {C<>[)≡R}

RUST DEVELOPER BOOTCAMP

19:30 - 21:00 | 03/07/2023

Discord, Zoom Online

Trait trong Rust

VBI Academy

1. Trait là gì ?

Trait là một cách để định nghĩa một tập hợp các phương thức được chia sẻ giữa các kiểu dữ liệu khác nhau

//share behaviour

1. Trait là gì?

```
struct Circle {
    radius: f64,
}
impl Circle {
    fn area(&self) -> f64 {
        std::f64::consts::PI * self.radius * self.radius
    }
}
```

```
struct Rectangle {
    width: f64,
    height: f64,
}
impl Rectangle {
    fn area(&self) -> f64 {
        self.width * self.height
    }}
```

Chúng ta định nghĩa 2 object Circle và Rectangle. Sự giống nhau và khác nhau trong 2 đoạn code trên ?

2. Cách định nghĩa Trait trong Rust

Trait sẽ giải quyết vấn đề trên bằng cách???

Định nghĩa trait bằng cách sử dụng từ khóa **trait** và liệt kê các phương thức và tính chất của **trait** đó

2. Cách định nghĩa Trait trong Rust

```
trait Drawable {
    fn draw(&self);
    fn area(&self) -> f64;
}
```

Từ khoá : trait + <Tên Trait>
Mô tả các phương phức được chia sẻ cho các kiểu dữ liệu khác nhau như **draw**, **area**

3. Sử dụng trait trong Rust

Đối với object Circle, chúng ta đã implement trait Drawable cho Circle

```
impl Drawable for Circle {
    fn draw(&self) {
        println!("Drawing a circle.");
    }

    fn area(&self) -> f64 {
        std::f64::consts::PI * self.radius * self.radius
    }
}
```

3. Sử dụng trait trong Rust

Đối với object Rectangle, chúng ta đã implement trait Drawable cho Rectangle

```
impl Drawable for Rectangle {
    fn draw(&self) {
        println!("Drawing a rectangle.");
    }

fn area(&self) -> f64 {

     self.width * self.height

  }}
```

4. Trait Bound

Trait Bound là một cách để giới hạn các kiểu dữ liệu được sử dụng trong một “**generic function**” hoặc “**generic struct**”

```
fn print_area<T: Drawable>(shape: T) {
    shape.draw();
    println!("Area: {}", shape.area());
}
```

Generic Function

4. Trait Bound

Generic Struct

```
struct Rectangle<T> {
   width: T,
   height: T,
}
impl<T: std::ops::Mul<Output = T> + Copy+ Into<f64>> Drawable for Rectangle<T> {
   fn area(&self) -> f64 {
      let result: T = self.width * self.height;
      result.into()}
...
```

Returning Trait

Mô tả: Có thể trả về một trait từ một hàm.

```
fn create_shape() -> impl Drawable {
    Circle { radius: 5.0 }
}

fn main() {
    let new_circle = create_shape();
    println!("New circle:{:?}", new_circle);
}
```

`impl Drawable` cannot be formatted using `{:?}` because it doesn't implement `Debug`

Supertraits

Mô tả: Supertraits là một cách để yêu cầu một trait phải triển khai một hoặc nhiều trait khác.

```
trait Drawable: std::fmt::Debug {
    fn draw(&self);
    fn area(&self) -> f64;
}
```

Associated Type

```rust
pub trait Iterator<T> {
    fn next(&mut self) -> T;
}
```

```rust
struct Counter{
    x:u32
}
impl Iterator<u32> for Counter{

    fn next(&mut self) -> u32 {
        self.x = self.x +1;
        self.x
    }
}
```

Khi dùng generic

Associated Type

```
pub trait Iterator {
    type Item;
    fn next(&mut self) -> Self::Item;
}
```

```
struct Counter{
    x:u32
}
impl Iterator for Counter{
    type Item =u32;
    fn next(&mut self) -> Self::Item{
        self.x = self.x +1;
        self.x
    }
}
```

Khi dùng associated type

Associated Type

Vấn đề khi sử dụng Generic Type

```
struct Container(i32, i32);

trait Contains<A, B> {
    fn contains(&self, _: &A, _: &B) -> bool;
    fn first(&self) -> i32;
    fn last(&self) -> i32;
}
```

```
impl Contains<i32, i32> for Container {
    fn contains(&self, number_1: &i32, number_2: &i32) -> bool {
        (&self.0 == number_1) && (&self.1 == number_2)
    }

    fn first(&self) -> i32 { self.0 }

    fn last(&self) -> i32 { self.1 }
}
```

Associated Type

Vấn đề khi sử dụng Generic Type

```
struct Container(i32, i32);

trait Contains<A, B> {
    fn contains(&self, _: &A, _: &B) -> bool;
    fn first(&self) -> i32;
    fn last(&self) -> i32;
}
```

```
impl Contains<i32, i32> for Container {
    fn contains(&self, number_1: &i32, number_2: &i32) -> bool {
        (&self.0 == number_1) && (&self.1 == number_2)
    }

    fn first(&self) -> i32 { self.0 }

    fn last(&self) -> i32 { self.1 }
}
```

Associated Type

Vấn đề khi sử dụng Generic Type

```
fn difference<A, B, C>(container: &C) -> i32 where
    C: Contains<A, B> {
    container.last() - container.first()
}
```

```
trait Contains {
    type A;
    type B;
    fn contains(&self, &Self::A, &Self::B) -> bool;
}
```

Associated Type

Vấn đề khi sử dụng Generic Type

```
impl Contains for Container {

    type A = i32;
    type B = i32;
```

```
fn difference<C: Contains>(container: &C) -> i32 {
    container.last() - container.first()
}
```

Trait Object

Mô tả: Trait Object là một cách để làm việc với các đối tượng có kiểu trait chung, cho phép tạo ra một danh sách đa dạng các đối tượng có cùng kiểu trait

```
let shapes: Vec<Box<dyn Drawable>> = vec![
Box::new(Circle { radius: 2.5 }),
Box::new(Rectangle { width: 5.0, height: 3.0 }),
];
```

Box & dyn ???

Static Dispatch

Hiểu như thế nào là dispatch ?

Dispatch nghĩa là gọi hàm/ phương thức (method) liên quan tới trait

Static Dispatch

Static dispatch

+ Xảy ra khi kiểu dữ liệu cụ thể của đối tượng được biết trong quá trình biên dịch (nghĩa là kích thước của object sẽ biết trong quá trình biên dịch -> cấp phát bộ nhớ cho object)

+ Việc triển khai phương thức(method call) được xử lý trong quá trình biên dịch dựa trên kiểu dữ liệu cụ thể

Static Dispatch

Cách định nghĩa Static Dispatch

+ parameter : sử dụng generic type -> constraint bởi trait Drawable
+ Hoặc parameter là 1 kiểu dữ liệu cụ thể

```
fn draw_static<T: Drawable>(shape: &T) {
    shape.draw();
}
```

Static Dispatch

Cách định nghĩa Static Dispatch

+ parameter : sử dụng generic type -> constraint bởi trait Drawable
+ Hoặc parameter là 1 kiểu dữ liệu cụ thể

```
fn draw_static(shape: &Circle) {
    shape.draw();
}
```

```
fn draw_static(shape: &Rectangle) {
    shape.draw();
}
```

Static Dispatch

Chúng ta biết dữ liệu cụ thể ở static dispatch

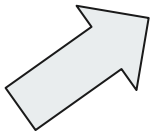
Static Dispatch

Cách sử dụng Static Dispatch

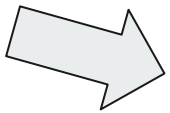
```
fn main() {
    let circle: Circle = Circle { radius: 5.0 };
    let rectangle: Rectangle = Rectangle { width: 10.0, height: 20.0 };
    // Static dispatch
    draw_static(&circle);
    draw_static(&rectangle);
}
```

Static Dispatch

```
fn draw_static<T: Drawable>(shape: &T) {
    shape.draw();
}
```



```
fn draw_static_rectangle(shape: &Rectangle) {
    shape.draw();
}
```



```
fn draw_static_circle(shape: &Circle) {
    shape.draw();
}
```

Dynamic Dispatch for Trait Object

Dynamic dispatch

+ Xảy ra khi kiểu dữ liệu cụ thể của object không được biết trong quá trình biên dịch và được quyết định bởi runtime (trong quá trình chạy logic)
+ Con trỏ (pointer) của trait (trait object)

Dynamic Dispatch for Trait Object

Dynamic dispatch

+ Xảy ra khi kiểu dữ liệu cụ thể của object không được biết trong quá trình biên dịch và được quyết định bởi runtime (trong quá trình chạy logic)
+ Con trỏ (pointer) của trait (trait object)

Dynamic Dispatch for Trait Object

Cách định nghĩa Dynamic Dispatch

+ parameter : Trait object
+ Thêm **dyn** để chỉ rõ đây là trait object

```
fn draw_dynamic(shapes: &[&dyn Drawable]) {
    for shape in shapes {
        shape.draw();
    }
}
```

Dynamic Dispatch for Trait Object

Cách sử dụng Dynamic Dispatch

```
fn main() {
    let circle: Circle = Circle { radius: 5.0 };
    let rectangle: Rectangle = Rectangle { width: 10.0, height: 20.0 };

    // Dynamic dispatch
    let shapes: Vec<&dyn Drawable> = vec![&circle, &rectangle];
    draw_dynamic(&shapes);
}
```

Dynamic Dispatch for Trait Object

Chúng ta không biết dữ liệu cụ thể ở dynamic dispatch

Sự khác nhau giữa Static Dispatch và Dynamic Dispatch

Giống nhau

+ Sử dụng để gọi các phương thức(method) từ một trait

+ Tính đa hình trong Rust

+ Tính Monomorphization thông qua việc triển khai phương thức của trait trên các kiểu dữ liệu khác nhau

Sự khác nhau giữa Static Dispatch và Dynamic Dispatch

Khác nhau

	Static Dispatch	Dynamic Dispatch
Size known at compile-time	Yes	No
Size known at runtime	No	Yes
Kiểu dữ liệu	Cụ thể	con trỏ(*) hoặc tham chiếu(&) của trait (trait object)
Performance (Thời gian chạy phương thức)	Nhanh (static)	Chậm (dynamic)

Khi nào sử dụng Static Dispatch và Dynamic Dispatch

https://users.rust-lang.org/t/choice-between-static-and-dynamic-dispatch/88029

https://www.reddit.com/r/rust/comments/rggekc/when_is_it_better_to_use_static_vs_dynamic/

Tóm tắt: Thường sẽ sử dụng static dispatch