链滴

Rust 自定义类型

作者：lingyundu

原文链接：https://ld246.com/article/1608184396054

来源网站：链滴

Rust 自定义类型主要有两种：结构体和枚举类型。

结构体

和元组一样，结构体中的值可以是不同的数据类型，但结构体有自己的名称，并且需定义结构体中各数据的名称和类型，称之为**字段**（field）。

结构体分为三类：

- C 语言风格结构体（C struct）
- 元组结构体（uple struct）
- 单元结构体（unit-like struct）

C 语言风格结构体

示例： 结构体的定义和实例化

```
// 定义 User 结构体
// 使用 `struct` 关键字，指定结构体名称，在大括号内指定字段（field）名和字段类型
struct User {
    username: String,
    email: String,
    sign_in_count: u64,
    active: bool,
}
// 创建一个 User 结构体的实例
let user1 = User {
    email: String::from("someone@example.com"),
    username: String::from("someusername123"),
    active: true,
    sign_in_count: 1,
};
// 修改字段的值
user1.email = String::from("anotheremail@example.com");

// 使用更新语法（struct update syntax）利用现有实例创建新的实例
let user2 = User {
    email: String::from("another@example.com"),
    username: String::from("anotherusername567"),
    ..user1 // 剩余字段与 user1 相同
};
```

元组结构体

元组结构体（tuple struct）是一种没有字段名的特殊结构体。

示例： 定义元组结构体

```
struct Color(i32, i32, i32);
struct Point(i32, i32, i32);
```

```
let black = Color(0, 0, 0);
let origin = Point(0, 0, 0);
```

单元结构体

单元结构体不包含任何字段，它的实例不存放数据。

示例：定义一个单元结构体

```
struct Nil
```

定义方法

方法与函数类似：它们使用 fn 关键字和名称声明，可以拥有参数和返回值。**方法**和结构体关联，并它的第一个参数通常是self（self指调用该方法的结构体实例）。

示例：定义结构体Rectangle的area方法

```
#[derive(Debug)]
struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    fn area(&self) -> u32 {
        self.width * self.height
    }
}

fn main() {
    let rect1 = Rectangle { width: 30, height: 50 };

    println!(
        "The area of the rectangle is {} square pixels.",
        rect1.area() // 调用area方法
    );
}
```

关联函数

可以在 impl 块中定义不以 self 作为参数的函数。这被称为**关联函数**（associated functions），因它们与结构体相关联。

关联函数经常被用作返回一个结构体新实例的构造函数。比如：String::from就是一个关联函数。

示例：定义结构体Rectangle的关联函数square

```
// 定义一个关联函数
impl Rectangle {
    fn square(size: u32) -> Rectangle {
        Rectangle { width: size, height: size }
    }
```

```
}
// 调用关联函数
let sq = Rectangle::square(3);
```

枚举类型

Rust 中的枚举类型与 C 语言中的类似，但功能更强大。

定义枚举类型

使用enum关键字定义枚举类型。

```
// 定义一个描述IP类型的枚举类型
enum IpAddrKind {
    V4,
    V6,
}

// 定义一个描述IP地址的枚举类型
enum IpAddr {
    V4(u8, u8, u8, u8), // 枚举的成员也可以是结构体
    V6(String),
}

// 定义一个描述网络事件的枚举
enum WebEvent {
    // 单元结构体
    PageLoad,
    PageUnload,
    // 元组结构体
    KeyPress(char),
    Paste(String),
    // 普通的结构体
    Click { x: i64, y: i64 }
}
```

使用枚举

通常可以使用::符号访问枚举成员。

示例:

```
enum SpecialPoint {
    Point(i32, i32),
    Special(String),
}
fn main() {
    let sp = SpecialPoint::Point(0, 0);
    match sp {
        SpecialPoint::Point(x, y) => {
            println!("I'm SpecialPoint(x={}, y={})", x, y);
        }
```

```
        SpecialPoint::Special(why) => {
            println!("I'm Special because I am {}", why);
        }
    }
}
```

使用use声明后，可以直接使用枚举的成员。

示例：

```
enum Status {
    Rich,
    Poor,
}

fn main() {
    use Status::{Poor, Rich};
    // use Status::*; // 也可以使用这种方式

    // `Poor` 等价于 `Status::Poor`。
    let status = Poor;

    match status {
        Rich => println!("The rich have lots of money!"),
        Poor => println!("The poor have no money..."),
    }
}
```

定义方法

与结构体一样，可以在impl块中给枚举类型定义方法。

示例：

```
enum Operations {
    Add,
    Subtract,
}

impl Operations {
    fn run(&self, x: i32, y: i32) -> i32 {
        match self {
            Self::Add => x + y,
            Self::Subtract => x - y,
        }
    }
}
```

Option<T>

许多其他语言都有的空值功能。**空值**（Null）是一个值，它代表没有值。在这些语言中，一个变量要是一个空值，要么是一个非空值。但这样的设计很容易引发错误，比如 Java 中常见的空指针异常（NullPointerException），通常是开发者忘记了处理变量值为 Null 的情况，并且编译器无法检查出来。

Tony Hoare，Null 的发明者，在他 2009 年的演讲 “Null References: The Billion Dollar Mistake” 中也曾经说到：

I call it my billion-dollar mistake. At that time, I was designing the first comprehensive type system for references in an object-oriented language. My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implment. This has led to innumerable errors, vulnerabilities, and system crashes, which have probbly caused a billion dollars of pain and damage in the last forty years.

Rust 没有空值，它采用了另一种方式来表示有值还是没有值。它在标准库定义了一个枚举类型：Option<T> 。其定义如下：

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

Option<T> 是预加载的，可以不需要 Option:: 前缀来直接使用 Some 和 None。Some 成员可以包任意类型的数据，None 表示空值。

当在使用 Option<T> 类型时，编译器就会知道可能会有 None，它会检查代码中有没有对 None 进处理。并且 Option<T> 和 T（即：Option包含的类型，它可以是任何类型）是不同的类型，必须将ption<T> 拆箱之后才能和 T 类型值进行运算，这也间接的提醒开发者必须对 None 进行处理。

相关资料

Rust Programming Language

Rust By Example

RustPrimer