TinTin

Office Hour 4

老师: Mike Tang
助教: Bob Liu

I 课程关键内容回顾

TinTin

Generics + Trait

- 类型是一种约束
- 理解泛型
- 函数参数中的泛型
- 结构体中的泛型
- 了解 PhantomData
- 枚举中的泛型(Result + Option)
- 实现单态化方法

- Trait 是对类型的约束(Trait Bounds)
- 类型的约束依赖
- Where 语句
- 默认实现(Default Implementation)
- 全局实现(Blanket Implementation)
- 关联类型
- Trait Object(dyn Trait)

I 课程关键内容回顾

TinTin

dyn Trait(Trait Object)

dyn Trait is a not a type but a type constructor.

It is parameterized with a lifetme, similar to how references are.

`dyn Trait + 'a` is

- a concrete, statically known type
- created by type erasing implementors of `Trait`
- used behind wide pointers to the type-erased value and to a static vtable
- `dyn trait` is dynamically sized (unsized, does not implement `Sized`) `:?Sized` and `dyn Trait + 'a` is a statically known type, a concrete type.
- an implementor of `Trait` via dynamic dispatch
- *not* a supertype of all implementors
- *not* dynamically typed
- *not* a generic
- *not* creatable from all values
- *not* available for all traits

I 课程关键内容回顾

TinTin

Generics function vs impl Trait(APIT: Argument Position Impl Trait)

```
fn foo(d: impl Display) { println!("{d}"); }
fn foo<D: Display>(d: D) { println!("{d}"); } // for now, almost the same
```

I 课程关键内容回顾

TinTin

Generics function vs dyn Trait

```rust
fn generic<T: Trait>(_rt: &T) {} // monomorphization
fn not_generic(_dt: &dyn Trait) {} // only one

// Owned or borrowed generics
fn foo1<T: Trait>(t: T) {}
fn bar1<T: Trait + ?Sized>(t: &T) {}

// Owned or borrowed `dyn Trait`
fn foo2(t: Box<dyn Trait + '_>) {}
fn bar2(t: &dyn Trait) {}
```

TinTin

Associate type vs Trait

Associate type: specified when implement

```
trait MyTrait {
    type MyAssociatedType;
    // Other trait methods and associated functions go here...
}

struct MyStruct;

impl MyTrait for MyStruct {
    type MyAssociatedType = i32;
    // Implement the trait methods and associated functions here...
}
```

Trait: specified when use.

Shell

```
struct Point<T> {
    x: T,
    y: T,
}

impl<T> Point<T> {
    fn new(x: T, y: T) -> Point<T> {
        Point { x, y }
    }
}
```

I 作业点评

TinTin

1. 使用枚举包裹三个不同的类型，并放入一个Vec中，对Vec进行遍历，调用三种不同类型的各自的方法。
2. 定义三个不同的类型，使用Trait Object，将其放入一个Vec中，对Vec进行遍历，调用三种不同类型的各自的方法。同时，说明其上两种不同实现方法的区别。

https://github.com/crimson629/rust_homework/blob/main/lesson_4_task_1/src/main.rs
https://github.com/fyang1024/rust_polymorhism/tree/main
https://github.com/SunTiebing/learn_rust/blob/main/rust_course/src/course_four/README.md

I 作业点评

TinTin

搜索相关文档，为你自己定义的一个类型或多个类型实现加法运算（用符号 +），并构思使用Trait Object实现类型方法的调用。

https://github.com/fyang1024/rust_add_operator/blob/main/src/main.rs
https://quinedot.github.io/rust-learning/dyn-safety.html#use-of-self-limitations

I 分享

TinTin

- https://cheats.rs/#generics-constraints
- NEAR 合约代码结构以及面向 Trait 开发模块.
- Foundry Cheatcode 实现与 REVM 核心 Tait

答疑讨论

THANKS

Twitter | YouTube | Discord