Office Hour 3

老师： Mike Tang
助教： 彭亚伦

TinTin

a@iy.rs

Rust

复合类型

01 —— 02 **Understanding Rust Type System**

Every variable, item, and value in a Rust program has a type.
The type of a value defines the interpretation of the memory holding it and the operations that may be performed on the value.

课程关键内容回顾

TinTin

本节课程主要讲解两大主要复合类型创建和使用

- Rust 中类型无处不在， 可以说， 一切表达式都有类型
- 复合类型也叫组合类型， Rust中的复合类型可以分为两大类
    1. 结构体（structure type）： 多个类型组合在一起共同表达一个值的复杂数据结构
    2. 枚举（enum type), 即标签联合（tagged union），也叫不相交并集（disjoint union），可以存储一组不同但固定的类型中的某个类型的对象，具体是哪个类型由其标签决定。
- 标准库里的部分集合类型

TinTin

一、结构体（structure type）

- 一个程序的核心骨干（ex. Substrate中的 Pallet 结构体）
  - https://github.com/substrate-developer-hub/substrate-node-template/blob/87b1b4728e29f68c6b2bafde7f3d0f4c8af9b302/pallets/template/src/lib.rs#L26C17-L26C17
- 结构体的创建:
  - 标准方式,
  - 元组结构体，也叫匿名结构体， 优点是无需命名字段， 缺点是如须对字段单独频繁访问时不方便， 适用于整体访问的数据， 如RGB颜色
  - 类单元结构体（unit-like structs), 类似于() 类型（unit），其适用于不需要储存数据， 但可以对外提供方法的场景。如实时世界时刻查询。

TinTin

一、结构体（structure type）

- 结构体的更新语法： .. 关键词 （必须要在末尾使用）
- 结构体定义方法：
  - Impl ： 优势是数据与方法定义有效分离解耦 --> Trait
  - self, &self, &mut self:
    - 其中self 会转移(消耗）所有权， 通常用于消耗转换
    - &self 不可变借用， 通常用于读取字段后计算逻辑
    - &mut self 可变借用， 通常用于改变结构体本身
  - 关联函数（without Self）
    - 通常new方法就是典型的关联函数， 如String::new()
    - from_xxx()的方法很多都是关联函数， 如 String::from_str("abc")
  - Newtype模式, 通常用于对原类型的提供限制或者验证实现, 如struct PhoneNumber(u32)

TinTin

二、枚举

- 枚举（enum type) 是标签联合（tagged union），也叫不相交并集（disjoint union）
- 每一个枚举的具体实例的值只能是其中某个成员
- Rust中最常用的两个枚举：
  - Option：
```
pub enum Option<T> {
    None,
    Some(T),
}
```
  - Result：
```
pub enum Result<T, E> {
    Ok(T),
    Err(E),
}
```
- 枚举同样用impl 定义方法
- 模式匹配：match，if let，和 while let 语法

TinTin

三、一些集合类型

- Vec
- String
- HashMap
- Hashset
- BtreeMap
- ....

- Module std::collections

作业点评

TinTin

题目：

请基于Rust的基本数据结构写一个简单的学生管理系统（比如，学生，社团，班级、课程等），明确类型之间的关系，进行基本的CRUD操作。

Tips:

- 需求分析
- 类型选型： Struct or Enum
    - Vec？ HashMap？BTreeMap？

优秀作业展示

本地简单版：@郭致阳

远程数据库版：@麦仔

```rust
use std::collections::HashMap;
use std::fs::File;
use std::io::prelude::*;
use serde::{Serialize, Deserialize};
use serde_json;

// 学生结构体
#[derive(Clone, Serialize, Deserialize)]
pub struct Student {
    id: u32,
    name: String,
    class_id: u32,
}

// 社团结构体
#[derive(Clone, Serialize, Deserialize)]
pub struct Club {
    id: u32,
    name: String,
    member_ids: Vec<u32>,
}

// 班级结构体
#[derive(Clone, Serialize, Deserialize)]
pub struct Class {
    id: u32,
    name: String,
    student_ids: Vec<u32>,
}

// 课程结构体
#[derive(Clone, Serialize, Deserialize)]
pub struct Course {
    id: u32,
    name: String,
    class_id: u32,
}

// 学生管理系统
#[derive(Serialize, Deserialize)]
pub struct StudentManagementSystem {
    students: HashMap<u32, Student>,
    clubs: HashMap<u32, Club>,
    classes: HashMap<u32, Class>,
    courses: HashMap<u32, Course>,
}

impl StudentManagementSystem {
    // 创建一个新的学生管理系统
    pub fn new() -> Self {
        Self {
            students: HashMap::new(),
            clubs: HashMap::new(),
            classes: HashMap::new(),
            courses: HashMap::new(),
        }
    }
```

```rust
use crate::schema::student::dsl::student;
use crate::schema::student::{title as title_str};
use crate::schema::class::title as c_t_s;
use diesel::pg::PgConnection;
use diesel::prelude::*;
use dotenvy::dotenv;
use std::env;
use crate::models::{Class, NewClass};
use crate::schema::class::dsl::class;

pub mod models;
pub mod schema;

// 创建连接
pub fn establish_connection() -> PgConnection {
    dotenv().ok();
    let database_url = env::var("DATABASE_URL").expect("DATABASE_URL must be set");
    PgConnection::establish(&database_url)
        .unwrap_or_else(|_| panic!("Error connecting to {}", database_url))
}

use self::models::{NewStudent, Student};
// 创建学生
pub fn create_stu_m(conn: &mut PgConnection, title: &str, class_id: &i32, club_id: &i32) -> Student {
    use crate::schema::student;
    let new_student = NewStudent {
        title,
        class_id,
        club_id,
    };

    diesel::insert_into(student::table)
        .values(&new_student)
        .returning(Student::as_returning())
        .get_result(conn)
        .expect("Error saving new post")
}

// 学生列表
pub fn list_stu_m(conn: &mut PgConnection) {
    let results = student
        .limit(5)
        .select(Student::as_select())
        .load(conn)
        .expect("Error loading posts");

    println!("Displaying {} posts", results.len());
    for post in results {
        println!("{}", post.title);
        println!("----------\n");
        println!("{}", post.class_id);
    }
}
```

Q&A

TinTin

答疑讨论

THANKS

Twitter | YouTube | Discord