探讨Rust智能指针二

Box、Vec<T> | String、Cell | RefCell、Rc | Arc、RwLock | Mutex

苏林

回顾上次公开课内容

1、什么是智能指针.
2、如何理解"智能", “智能在何处"?
3、智能指针工作机制.
4、以递归为例, 和大家一起聊了聊Box在递归中的使用

Trait

Deref trait 和 Drop trait

1、可以自动解引用, 提升开发体验.
=> A、*x 手动解引用的方式, 等价于 *(x.deref())
=> B、x.fun(), fun(&mut x) => x.deref()

2、可以自动化管理内存, 安全无忧.

今天公开课内容

1、理解 Cell<T> 与 RefCell<T>
2、理解 Rc<T> 与 Arc<T>
3、Mutex

理解 Cell<T> 与 RefCell<T>

为了理解Cell与RefCell, 我们先来了解什么是 可变性 -> 引出内部可变性

```
1  fn main() {
2      let x : i32 = 1;
3      x += 1;
4      println!("The value of x is {}.", x);
5  }
```

理解 Cell<T> 与 RefCell<T>

为了理解Cell与RefCell, 我们先来了解什么是 内部可变性?

```rust
fn add_and_print(x: &mut i32) {
    *x += 1;
    println!("The value of x in add_and_print is {}.", x);
    // mutable reference to x is dropped
}

fn main() {
    let mut x: i32 = 1; // Declare a mutable variable x
    add_and_print(&mut x); // Modify x in function
    x += 1; // Modify x again
    println!("The value of x in main is {}.", x);
}
```

理解 Cell<T> 与 RefCell<T>

为了理解Cell与RefCell, 我们先来了解什么是 内部可变性?

```rust
#[derive(Debug)]
struct XStruct<'a> {
    x: &'a mut i32,
}

fn main() {
    let mut x: i32 = 1;
    let x_struct = XStruct {x: &mut x};
    println!("The value of x_struct is {:?}.", x_struct);
    println!("The value of x is {:?}.", x);
}
```

理解 Cell<T> 与 RefCell<T>

为了理解Cell与RefCell, 我们先来了解什么是 内部可变性?

```rust
#[derive(Debug)]
struct XStruct<'a> {
    x: &'a mut i32,
}

fn main() {
    let mut x :i32 = 1; // Declare a mutable variable x
    let x_struct = XStruct { x: &mut x }; // Pass a mutable reference to x

    x += 1; // Modify x...

    println!("The value of x_struct is {:?}.", x_struct);
    println!("The value of x is {:?}.", x);
}
```

理解 Cell<T> 与 RefCell<T>

RefCell的使用

```rust
use std::cell::RefCell;

#[derive(Debug)]
struct XStruct<'a> {
    x: &'a RefCell<i32>,
}

fn add_and_print(x_struct: &XStruct) {
    *x_struct.x.borrow_mut() += 1;
    println!("The value of x_struct in add_and_print is {:?}.", x_struct);
}

fn main() {
    let ref_cell_x : RefCell<i32> = RefCell::new( value: 1);
    let x_struct = XStruct { x: &ref_cell_x };
    add_and_print(&x_struct);

    *ref_cell_x.borrow_mut() += 1;

    println!("Final value of x_struct is {:?}.", x_struct);
    println!("Final value of x is {:?}.", ref_cell_x);
}
```

理解 Cell<T> 与 RefCell<T>

Cell的使用

```rust
use std::cell::Cell;

#[derive(Debug)]
struct XStruct<'a> {
    x: &'a Cell<i32>,
}
fn add_and_print(x_struct: &XStruct) {
    let x : i32 = x_struct.x.get();
    x_struct.x.set( val: x+1);
    println!("The value of x_struct in add_and_print is {:?}.", x_struct);
}

fn main() {
    let cell_x : Cell<i32> = Cell::new( value: 1);
    let x_struct = XStruct { x: &cell_x };
    add_and_print(&x_struct);

    cell_x.set( val: cell_x.get()+1);

    println!("Final value of x_struct is {:?}.", x_struct);
    println!("Final value of x is {:?}.", cell_x);
}
```

理解 Cell<T> 与 RefCell<T>

应该什么时候使用 RefCell 和 Cell?

理解 Rc<T> 与 Arc<T>

1、每个值value, 都有一个所有者owner.
2、同一时间, 一个值只能有一个所有者owner.
3、当所有者owner离开作用域, 对应的值会自动 drop.

被多个变量共享，又无法用引用来解决, 无法确定生命周期.

Rc -> reference count 引用计数, java垃圾回收机制引用计数.

Arc -> A(atomic) -> atomic reference count 原子操作的意思, 线程安全.

```rust
use std::rc::Rc;

fn main() {
    let a = Rc::new(String::from("Hello World!"));
    println!("ref count is {}", Rc::strong_count(&a));
    let b = Rc::clone(&a);
    println!("ref count is {}", Rc::strong_count(&a));
    {
        let c = Rc::clone(&a);
        println!("ref count is {}", Rc::strong_count(&a));
        println!("ref count is {}", Rc::strong_count(&c));
    }
    println!("ref count is {}", Rc::strong_count(&a));
}
```

理解 Rc<T> 与 Arc<T>

```rust
use std::sync::{Arc, Mutex};
use std::thread;

fn main() {
    let counter = Arc::new(Mutex::new(0));
    let mut handles = vec![];

    for _ in 0..10 {
        let counter = Arc::clone(&counter);
        let handle = thread::spawn(move || {
            let mut num = counter.lock().unwrap();

            *num += 1;
        });
        handles.push(handle);
    }

    for handle in handles {
        handle.join().unwrap();
    }

    println!("Got Result: {}", *counter.lock().unwrap());
}
```

理解 Rc<T> 与 Arc<T>

底层实现

QA环节

加群一起交流Rust & Datafuse

Datafuse Rust语言中文社区 知数堂 zhishuteng.com