# 从C++到Rust，不要Move，要默认的Clone

原创 Ajonbin AJonbin的杂货铺 2024年03月12日 22:16 美国

对于C++来说，如果成员变量包含指针变量的类型，那就需要实现拷贝构造函数或是重载赋值操作符来实现复制对象。

同样，对于Rust来说，当某个类型包含指针类型变量的时候，赋值此类型的变量就不能再用Copy Trait这种按位拷贝的方式，必须通过Clone Trait来实现深拷贝。

上次谈到通过用#[derive]的方式获得默认的Copy和Clone实现。

那么我们自己定义的类型如果包含了指针类型，是否可以用Clone Trait默认实现呢？还需不需要自己写代码实现呢？

让我们动手来试试。

我们先来定义一个3D点的类型。

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

struct Point1D{
    x: i32,
}
struct Point2D{
    p1d: Point1D,
    y: Rc<i32>,
}
struct Point3D{
    p2d: Box::<Point2D>,
    z: Box::<i32>,
}
```



这里我们尽量把简单的类型复杂化。

一个普通的Point3D点，我们定义了Point1D和Point2D，作为Point3D的成员变量类型，并通过Box指针和Rc指针来引用变量的值。

这样在我们自己定义的结构中就有不同指针类型，看看Clone Trait的默认实现能为我们做什么。

```
15 fn main(){
16     let rc_y = Rc::new(20);
17 
18     let p1 = Point3D{
19         p2d: Box::new(Point2D{
20             p1d: Point1D{
21                 x: 10,
22             },
23             y: rc_y,
24         }),
25         z: Box::new(30),
26     };
27     let p2 = p1.clone();
28     println!("p1 = {:?}, z = {:p}, y = {:p}", p1, p1.z, p1.p2d.y);
29     println!("p2 = {:?}, z = {:p}, y = {:p}", p2, p2.z, p2.p2d.y);
30 }
```



在main()函数，我们先时创建了一个新的Point3D p1，然后通过调用p1.clone() 复制一个新的Point3D p2。最后在打印p1和p2。

目前这段代码是编译不通过的，因为Point3D的clone()函数还没有实现。

我们先让Point3D通过#[derive]来获得Clone Trait的默认实现。Point1D和Point2D通过#[derive(Debug)]获得默认的打印实现，但不实现Clone。

```rust
#[derive(Debug)]
struct Point1D{
    x: i32,
}

#[derive(Debug)]
struct Point2D{
    p1d: Point1D,
    y: Rc<i32>,
}

#[derive(Debug, Clone)]
struct Point3D{
    p2d: Box::<Point2D>,
    z: Box::<i32>,
}
```



编译之后，报错，Point2D没有实现Clone。

```
Standard Error

   Compiling playground v0.0.1 (/playground)
error[E0277]: the trait bound `Point2D: Clone` is not satisfied
  --> src/main.rs:16:5
   |
14 | #[derive(Debug, Clone)]
   |                 ----- in this derive macro expansion
15 | struct Point3D{
16 |     p2d: Box::<Point2D>,
   |     ^^^^^^^^^^^^^^^^^^^ the trait `Clone` is not implemented for `Point2D`
   |
```



接着Point2D加上#[derive(Debug, Clone)]，Point1D保持不变。再编译。

```
   Compiling playground v0.0.1 (/playground)
error[E0277]: the trait bound `Point1D: Clone` is not satisfied
  --> src/main.rs:10:5
   |
8  | #[derive(Debug, Clone)]
   |                 ----- in this derive macro expansion
9  | struct Point2D{
10 |     p1d: Point1D,
   |     ^^^^^^^^^^^^ the trait `Clone` is not implemented for `Point1D`
   |
```



报错，Point1D也加上#[derive(Debug, Clone)]。

Standard Output

```
p1 = Point3D { p2d: Point2D { p1d: Point1D { x: 10 }, y: 20 }, z: 30 }, z = 0x55629dffba10, y = 0x55629dffb9e0
p2 = Point3D { p2d: Point2D { p1d: Point1D { x: 10 }, y: 20 }, z: 30 }, z = 0x55629dffba50, y = 0x55629dffb9e0
```

这下编译通过了。p2的内容和p1的内容一样。

同时，可以看到p1.z和p2.z的指针是不同的Box<i32>地址。

而p1.y和p2.y是Rc，所以它们的地址是一样的。

通过这个小小的例子，我们可以看到：

Clone Trait的默认实现是通过调用成员变量的Clone来实现的。

对于Box和Rc，Rust默认的Clone实现就可以实现复制，并不需要我们另外写代码来实现。当然前提是指针指向的类型已经实现了Clone。

下次讲讲如何实现自己的Clone。

上一篇：从C++到Rust，不要Move，要Copy和Clone