从C++到Rust，类指针类型，普通篇

原创 Ajonbin AJonbin的杂货铺 2024年01月17日 00:00 美国

指针可以说C++中最强大最灵活的地方，当然也是出问题最多的地方。

本质上说指针就是一个内存地址。

对于Rust而言，用来表示这种内存地址的类型有三种：

- reference，引用
- Box<T>，类似于智能指针，在堆上分配内存
- raw pointer，原始指针，类似C++的指针

其实这些和C++的指针概念上大致相似。

由于Rust语言本身安全特性，使用reference和box一般不会发生C++中常见的指针使用错误，比如段错误，内存泄漏等问题。

当然代价就是失去了C++指针的灵活度和自由度。

Raw pointer是个例外，它就为了这种自由度而特地保留的。使用raw pointer的话，Rust固有的安全保护机制就无法使用，所以使用raw pointer必须在unsafe代码段里。

先来看看reference和box

今天用个新工具来演示代码Rust Playground。这是个在线编译运行Rust的网站，作为演示用起来很方便。

> https://play.rust-lang.org/

先来看看reference，上代码

```rust
fn main() {
    println!("Hello, memory!");

    let var_u16_1: u16 = 16;
    let var_u32_2: u32 = 32;
    let ref_u16_1: &u16 = &var_u16_1;
    let ref_u32_2: &u32 = &var_u32_2;

    println!("Address of var_u16_1 is: {:p}", &var_u16_1);
     println!("Address of var_u32_2 is: {:p}", &var_u32_2);

     println!("Value of ref_u16_1 is: {:p}", ref_u16_1);
     println!("Value of ref_u32_2 is: {:p}", ref_u32_2);

     println!("Value of ref_u16_1 is: {}", ref_u16_1);
     println!("Value of ref_u32_2 is: {}", ref_u32_2);

     println!("Value of ref_u16_1 is: {}", *ref_u16_1);
     println!("Value of ref_u32_2 is: {}", *ref_u32_2);
}
```


再看运行结果

```
Hello, memory!
Address of var_u16_1 is: 0x7ffedcba79e2
Address of var_u32_2 is: 0x7ffedcba79e4
Value of ref_u16_1 is: 0x7ffedcba79e2
Value of ref_u32_2 is: 0x7ffedcba79e4
Value of ref_u16_1 is: 16
Value of ref_u32_2 is: 32
Value of ref_u16_1 is: 16
Value of ref_u32_2 is: 32
```


最后来解释下

```rust
    let var_u16_1: u16 = 16;
    let var_u32_2: u32 = 32;
```

第4-5 行新建了两个变量 var_u16_1 和 var_u32_2，分别是16位无符号和32位无符号整型。

```
6     let ref_u16_1: &u16 = &var_u16_1;
7     let ref_u32_2: &u32 = &var_u32_2;
```

第6-7行新建了两个引用变量

ref_u16_1类型是&u16，就是一个指向u16的reference，它的值是&var_u16_1，也就是var_16_1的地址。

ref_u32_2类型是&u32，就是一个指向u32的reference，它的值是&var_u32_2，也就是var_u32_2的地址。

引用reference就是在变量或类型前加上&操作符，类似于C++的取地址符，实际也差不多，都是取地址的意思。

```
9     println!("Address of var_u16_1 is: {:p}", &var_u16_1);
10     println!("Address of var_u32_2 is: {:p}", &var_u32_2);
11
12     println!("Value of ref_u16_1 is: {:p}", ref_u16_1);
13     println!("Value of ref_u32_2 is: {:p}", ref_u32_2);
14
```


第9-13行，对reference进行了打印。一种是直接取&，另一种是打印reference 变量。

```
Address of var_u16_1 is: 0x7ffedcba79e2
Address of var_u32_2 is: 0x7ffedcba79e4
Value of ref_u16_1 is: 0x7ffedcba79e2
Value of ref_u32_2 is: 0x7ffedcba79e4
```


显然，直接引用和引用变量的值是一样的。

注意到由于var_u16_1和var_u32_2都是栈上的局部变量，它们的地址差了2个字节，也就是var_u16_1的大小。

格式{:p}是专门用来打印地址值的。

```
15      println!("Value of ref_u16_1 is: {}", ref_u16_1);
16      println!("Value of ref_u32_2 is: {}", ref_u32_2);
17
18      println!("Value of ref_u16_1 is: {}", *ref_u16_1);
19      println!("Value of ref_u32_2 is: {}", *ref_u32_2);
20  }
```


第15-19行，打印出引用所指向的地址上变量的值。标准做法是用*操作符de-reference取到变量值，就像18-19行。在打印的时候可以省略*操作符，println!会自动de-reference。

```
Value of ref_u16_1 is: 16
Value of ref_u32_2 is: 32
Value of ref_u16_1 is: 16
Value of ref_u32_2 is: 32
```


接着来讲讲Box<T>。

Box和reference不太一样，reference可以理解为取地址符，只是取得变量的地址，对于变量内存在堆上还是栈上并没有要求。

Box<T>一般用Box::new(T)在堆heap上创建一个类型为T的变量。

Box<T>是一个智能指针，当它生命周期结束后，内存就会被释放。

取名叫Box就是为了强调它对于堆上内存的管理权和控制权，像个盒子一样，把堆上的内存包在里面。

看一段最简单的代码

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

fn main() {
    println!("Hello, box!");

    let box_1: Box<Cubic> = Box::new(Cubic{length: 10, width: 20, height: 30});
    let box_2: Box<u32> = Box::new(100);

    println!("box_1 is: {:p}", box_1);
    println!("box_2 is: {:p}", box_2);

    println!("value of box_1 is: {:?}", *box_1);
    println!("value of box_2 is: {}", *box_2);

    println!("value of box_1 is: {:?}", box_1);
    println!("value of box_2 is: {}", box_2);

    println!("Size of Box<Cubic>: {} bytes", std::mem::size_of::<Cubic>());
}
```


结果

```
Hello, box!
box_1 is: 0x56197fa329d0
box_2 is: 0x56197fa329f0
value of box_1 is: Cubic { length: 10, width: 20, height: 30 }
value of box_2 is: 100
Size of Box<Cubic>: 12 bytes
```



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


第1-6行，定义了一个结构Cubic，有3个成员变量，长宽高。都是u32类型。#[derive(Debug)]表示Cubic继承了默认的Debug 特性trait的实现，这样就可以用格式{:?}打印出Cubic的内容。

```
11      let box_1: Box<Cubic> = Box::new(Cubic{length: 10, width: 20, height: 30});
12      let box_2: Box<u32> = Box::new(100);
```

第11-12行，创建两个Box智能指针，box_1创建了一个新的Cubic变量，长宽高分别是10，20，30。box_2创建了一个u32变量，值是100。

记住这两个变量都是在堆heap上的。

```
14      println!("box_1 is: {:p}", box_1);
15      println!("box_2 is: {:p}", box_2);
```


第14-15行打印了box的值，也就是变量的地址。

```
box_1 is: 0x56197fa329d0
box_2 is: 0x56197fa329f0
```

注意到box的地址都是0x5619xx，和之前局部变量的地址0x7ffexx是明显不同的。

```
17      println!("value of box_1 is: {:?}", *box_1);
18      println!("value of box_2 is: {}", *box_2);
19
20      println!("value of box_1 is: {:?}", box_1);
21      println!("value of box_2 is: {}", box_2);
```


第17-21行打印出Box所指向内存的变量的值。同样，标准做法是使用*操作符来取变量值。println!也可以省略。

```
value of box_1 is: Cubic { length: 10, width: 20, height: 30 }
value of box_2 is: 100
```

今天就到这，下次讲raw pointer。