从C++到Rust，类指针类型，不普通篇

原创 Ajonbin AJonbin的杂货铺 2024年01月17日 23:23 美国

普通篇讲了常用的引用reference和智能指针Box<T>。

不普通篇讲讲原始指针raw pointer。

原始指针raw pointer就是C++中的指针pointer。

Rust中保留了raw pointer主要有以下几个原因：

- 和C语言保持兼容。Rust一个设计要点就是能和其他编程语言相互调用。保留raw pointer就是为了和C兼容。
- 底层内存操作功能。某些情况下，直接管理和操作内存是必要的。raw pointe就提供了这种灵活性。
- 所有C/C++程序员都希望保留raw pointer（我猜的）

Raw pointer 有两种

- *const T -- 指向的值不可变
- *mut T -- 指向的值可变

需要注意的是，你只能在**unsafe{}**代码块中才能de-reference解引用原始指针raw pointer。

```
1 #[derive(Debug)]
2 struct Cubic {
3     length: u32,
4     width: u32,
5     height: u32,
6 }
7
```


我们还是继续用普通篇里Cubic这个结构。

接着我们要用raw pointer通过修改内存值，来改变cubic对象长宽高的值。

老三样，贴代码，贴结果，贴膏药

```
 8 fn main() {
 9     println!("Hello, raw pointer!");
10 
11     let mut small_cubic: Cubic = Cubic{length:10,width:20,height:30};
12     println!("Before modification Cubic: {:?}", small_cubic);
13     println!("Address of cubic: {:p}", &small_cubic);
14 
15     let raw_ptr_length: *mut u32 = &mut small_cubic.length as *mut u32;
16     println!{"raw_ptr_length = {:p}", raw_ptr_length};
17 
18     let raw_ptr_width: *mut u32 = raw_ptr_length.wrapping_add(1);
19     println!{"raw_ptr_width = {:p}", raw_ptr_width};
20 
21     let raw_ptr_height: *mut u32 = raw_ptr_width.wrapping_byte_offset(4);
22     println!{"raw_ptr_height = {:p}", raw_ptr_height};
23 
24     unsafe {
25         *raw_ptr_length = 11;
26         *raw_ptr_width = 22;
27         *raw_ptr_height = 33;
28     }
29 
30     println!("After modification Cubic: {:?}", small_cubic);
31 }
```



```
Hello, raw pointer!
Before modification Cubic: Cubic { length: 10, width: 20, height: 30 }
Address of cubic: 0x7ffd05ead5ac
raw_ptr_length = 0x7ffd05ead5ac
raw_ptr_width = 0x7ffd05ead5b0
raw_ptr_height = 0x7ffd05ead5b4
After modification Cubic: Cubic { length: 11, width: 22, height: 33 }
```


> mut关键字，意思是mutable，表明这个变量的值是可以修改的。

```
11     let mut small_cubic: Cubic = Cubic{length:10,width:20,height:30};
12     println!("Before modification Cubic: {:?}", small_cubic);
13     println!("Address of cubic: {:p}", &small_cubic);
```


第11-13行，创建了一个Cubic对象small_cubic，长宽高分别是10，20，30。

在声明small_cubic的时候，我们加了mut关键字，表示我们在后面的代码是可以修改small_cubic的值。

然后再打印了下small_cubic的地址。

```
Before modification Cubic: Cubic { length: 10, width: 20, height: 30 }
Address of cubic: 0x7ffd05ead5ac
```

通过打印，small_cubic的地址是0x7ffd05ead5ac。

```
15      let raw_ptr_length: *mut u32 = &mut small_cubic.length as *mut u32;
16      println!{"raw_ptr_length = {:p}", raw_ptr_length};
```

第15-16行，新建了变量raw_ptr_length。

它的类型是***mut u32**，一个指向u32的原始指针raw pointer，并且这个u32的值是可变的。

&mut small_cubic.length 是取small_cubic成员变量length的地址，并指明这个length是可变的。

as *mut u32是将之前通过&取到的地址转换为原始指针 *mut u32。

> as 是类型转换关键字

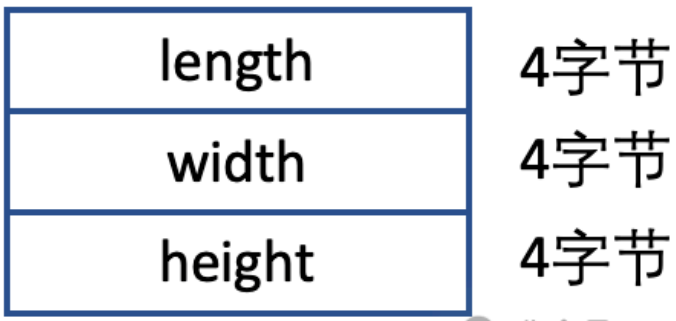
```
raw_ptr_length = 0x7ffd05ead5ac
```

从16行的打印结果可以看出small_cubic.length地址也是0x7ffd05ead5ac，和small_cubic的地址相同。当然也是符合预期的，length是Cubic的第一个成员变量，它的地址也就是cubic对象的地址。

```
18      let raw_ptr_width: *mut u32 = raw_ptr_length.wrapping_add(1);
19      println!{"raw_ptr_width = {:p}", raw_ptr_width};
```

第18行，新建了一个类型为*mut u32 的原始指针 raw_ptr_width，用来指向small_cubic的width成员变量。

Cubic对象的内存结构大致是这样的，u32是32位，4字节，所以width的地址就是length的地址+4字节。这对一个C++程序员来说是显而易见的。



Rust不能对raw pointer进行+操作，必须调用raw pointer的函数来进行地址偏移。

这里调用的是**wrapping_add(count)**函数。count就是要偏移的数量。但是这个数量是以指针指向对象的大小为单位的。

这里raw_ptr_length是*mut u32，它指向对象的类型是u32。u32的大小是32位，4个字节。所以wrapping_add(1)就是增加一个单位量，也就是4字节。

对于一个*mut u16，那么wrapping_add(1)增加的一个单位量，就是16位，2个字节。

```
raw_ptr_width = 0x7ffd05ead5b0
```

可以看到raw_ptr_width的值是0x7ffd05ead5b0 = 0x7ffd05ead5ac + 4

```
21      let raw_ptr_height: *mut u32 = raw_ptr_width.wrapping_byte_offset(4);
22      println!{"raw_ptr_height = {:p}", raw_ptr_height};
```

同样，第21行，新建了raw_ptr_height来指向height成员变量。

这里我们用了另一个函数**raw_ptr_width.wrapping_byte_offset(4)**。根据名字我们就可以知道这函数是以byte字节为单位来做地址偏移的，所以这里位移数量是4。

```
raw_ptr_height = 0x7ffd05ead5b4
```

可以看到raw_ptr_height的值是0x7ffd05ead5b4 = 0x7ffd05ead5b0 + 4

```
24      unsafe {
25          *raw_ptr_length = 11;
26          *raw_ptr_width = 22;
27          *raw_ptr_height = 33;
28      }
29
```


第24-27行就是通过地址来修改内存值了。

之前提过de-reference解应用必须在unsafe{}代码段里执行，不然编译会报错。

```
30      println!("After modification Cubic: {:?}", small_cubic);
31  }
```

```
After modification Cubic: Cubic { length: 11, width: 22, height: 33 }
```

然后我们就可以看到small_cubic的值被修改了，长宽高被改为了11，22，33。