从C++到Rust，直接读写内存，修改Vec

原创 Ajonbin AJonbin的杂货铺 2024年02月22日 22:30 美国

上次讲了通过原始指针Raw Pointer来查看Vec的内存布局，今天接着看怎么用Raw Pointer来直接修改Vec的值。

- 修改Vec某个元素的值
- 增加一个元素

还是先上代码，再慢慢分析

```rust
use std::mem;

fn main(){
    let mut v = Vec::new();
    v.push(11);
    v.push(22);
    println!("Original Vec: {:?}",v);

    let ptr_e0 = &v[0] as *const i32;
    let u64_e0 = ptr_e0 as u64;
    let u64_e1 = u64_e0 + 0x4;
    let ptr_e1 = u64_e1 as *mut i32;
    unsafe {*ptr_e1 = 0;}
    println!("After modify 2nd element: {:?}", v);


    let u64_e2 = u64_e0 + 0x8;
    let ptr_e2 = u64_e2 as *mut i32;
    unsafe {*ptr_e2 = 33;}
    println!("After add one new element: {:?}", v);


    let ptr_v: u64 = unsafe{
        mem::transmute(&v)
    };
    let ptr_v_length = ptr_v + 8*2;
    let p_length = ptr_v_length as *mut u64;
    unsafe {*p_length = 3;}
    println!("After update the capacity: {:?}", v);
}
```


输出结果是这样的

Standard Output

```
Original Vec: [11, 22]
After modify 2nd element: [11, 0]
After add one new element: [11, 0]
After update the capacity: [11, 0, 33]
```

第4-7行，红色框

```
4     let mut v = Vec::new();
5     v.push(11);
6     v.push(22);
7     println!("Original Vec: {:?}",v);
```


跟之前一样，创建一个vector，添加两个元素，11和22，所以v的类型是Vec<i32>。

红色框的输出是

```
Original Vec: [11, 22]
```

第9-14行，橙色框

```
 9     let ptr_e0 = &v[0] as *const i32;
10     let u64_e0 = ptr_e0 as u64;
11     let u64_e1 = u64_e0 + 0x4;
12     let ptr_e1 = u64_e1 as *mut i32;
13     unsafe {*ptr_e1 = 0;}
14     println!("After modify 2nd element: {:?}", v);
```


第9行：先用&v[0]取到v的第一个元素的地址，&v[0]的类型是&i32，把它强制转换为一个指向i32常量的原始指针*const i32。

第10行：由于是64位系统，指针是64位的值。把指针转换成一个u64。

注意，&i32是不能直接通过as关键字转换成u64的，Rust编译器不允许这么转换，所以这里的转换过程是***&i32 --> *const i32 --> u64***。

第11行：我们将得到的u64加上4字节的偏移，也就是v第一个元素的地址+4字节，就得到了v第二个元素的地址，用u64表示。

第12行：然后将得到的第二个元素的u64地址值转换成原始指针*mut i32。由于我们要修改第二个元素的值，所以这里指针是mut，不是const。

第13行：通过解引用de-reference指针，我们将第二个元素的值从22改为0。由于解引用原始指针是不安全的，所以必须在unsafe{}代码段中使用。

通过打印可以看出，v的第二个元素已经被改成了0。

```
After modify 2nd element: [11, 0]
```

第17-20行，蓝色框

```
17      let u64_e2 = u64_e0 + 0x8;
18      let ptr_e2 = u64_e2 as *mut i32;
19      unsafe {*ptr_e2 = 33;}
20      println!("After add one new element: {:?}", v);
```


由于v的capacity是4，但是只有两个元素，长度length是2，所以v里有两块空闲的i32。

我们就利用和上面是相同的操作来增加一个元素33。

唯一的区别是指针偏移量是8，2个i32的长度。

看一下结果打印

```
After add one new element: [11, 0]
```

出乎意料的是，v的元素并没有变化，还是[11, 0]。

难道是修改内存没有用吗？

当然不是，我们回想一下Vec的内存布局，Vec有一个属性是length，我们只是在堆上第三个元素的位置上修改了值，但是length还是2，所以v还只是一个拥有2个元素的vector，所以从打印结果来看并没有变化。

那么下一步就是要修改v的length。

第23-29行，绿色框

```
23      let ptr_v: u64 = unsafe{
24          mem::transmute(&v)
25      };
26      let ptr_v_length = ptr_v + 8*2;
27      let p_length = ptr_v_length as *mut u64;
28      unsafe {*p_length = 3;}
29      println!("After update the capacity: {:?}", v);
```


第23-25行：通过transmute()函数，直接将v的地址转换成u64来表示地址值。

第26行：将v的首地址偏移2个u64的大小，也就是16个字节，就可以得到v的成员变量length的地址值。

第27行：将length地址u64值转换成Raw Pointer *mut 64。因为要修改它的值，所以这里也是mut。

第28行：通过de-reference将length的值改为3。

```
After update the capacity: [11, 0, 33]
```

这下就可以看到v的第3个元素了，它的值是之前设置的33。

这两篇演示了怎么利用raw point来直接读写内存。

一般情况下都不太会用到这些操作。但对于嵌入式或是一些IO设备的操作还是有可能会使用到了。

直接读写内存Rust也可以，就是有些麻烦。

上一篇：从C++到Rust，直接读写内存，vec内存布局