# 从C++到Rust，内存所有权管理Ownership，Move，转移所有权

原创 Ajonbin AJonbin的杂货铺 2024年03月05日 22:58 美国

之前谈了Rust中所有权Ownership，简单来说，就是每个值都有唯一的owner。

值的Ownership可以通过Move操作来转移给其他变量。

当变量A将它的值的所有权转移给变量B后，A就失去了对值的所有权，B将拥有这个值， 并且控制这个值的生命周期。A将被编译器认为是一个未初始化的变量，将不能再使用。

和C++不同，Rust默认语义就是Move。

也就是说当需要赋值的时候，Rust默认就是使用移动语义，而C++默认是拷贝语义。

赋值操作一般发生在下列情况：

- 给本地变量赋值
- 函数传递参数
- 函数返回值

下面来通过一些简单例子，来理解下Move语义。

**给变量赋值**

```rust
fn main(){
    let v1: Vec::<i32> = vec![1,2,3,4];

    let _v2 = v1;

    println!("{:?}", v1);
}
```



这段代码通俗易懂。先创建了一个变量v1，是一个Vec。然后，v1赋值为_v2，最后打印v1。

这样的逻辑和处理在C++中是完全没有问题的。当把v1赋值给_v2时，C++会调用v1的拷贝构造函数来创建_v2。这就是C++默认的拷贝语义。由于是拷贝，赋值之后，v1还是使用的。

但是Rust就完全不一样了。

由于Rust默认的是移动语义，在执行第4行

```
4       let _v2 = v1;
```

之后，v1的值，也就是vec![1,2,3,4]的ownership从v1转移Move到_v2了。

由于Rust中值的owner只能有一个，那么v1在交出所有权之后就被编译器认为是一个未被初始化uninitialized的变量。

那么这个时候，v1就不能再被使用了。

来看看这段代码的编译结果。

```
error[E0382]: borrow of moved value: `v1`
 --> tmp.rs:6:22
  |
2 |     let v1: Vec::<i32> = vec![1,2,3,4];
  |         -- move occurs because `v1` has type `Vec<i32>`, which does not implement the `Copy` trait
3 |
4 |     let _v2 = v1;
  |               -- value moved here
5 |
6 |     println!("{:?}", v1);
  |                      ^^ value borrowed here after move
  |
  = note: this error originates in the macro `$crate::format_args_nl` which comes from the expansion of the macro `println` (in Nightly builds, run with -Z macro-backtrace for more info)
help: consider cloning the value if the performance cost is acceptable
  |
4 |     let _v2 = v1.clone();
  |                  ++++++++

error: aborting due to previous error

For more information about this error, try `rustc --explain E0382`.
```



Rust编译器rustc的错误提示还是比较友好的。这里的错误是

> error[E0382]: borrow of moved value: `v1`

意思就是v1将ownership移出以后，println!()就不能再使用v1这个变量了。这里的提到了另一种所有权的操作， borrow。关于borrow后面聊。

这是一个在刚开始写Rust时经常会遇到的错误。大家都习惯了默认时拷贝语义，需要点时间熟悉默认移动语义。

### 函数传递参数

对于函数调用，传值还是传引用是一个问题。

和赋值一样，Rust在函数调用时，默认会转移参数的所有权。

```
1 fn take_vec(v: Vec::<i32>){
2     println!("{:?}", v);
3 }
4
5 fn main(){
6     let v1: Vec::<i32> = vec![1,2,3,4];
7
8     take_vec(v1);
9
10     println!("{:?}", v1);
11 }
```



来看看这段代码

第1-3行，定义了一个函数take_vec，它的参数是一个Vec::<i32>。

第6行，创建一个Vec::<i32>，变量为v1。

第8行，调用take_vec()函数，并将v1作为参数传入。

注意，调用函数时，Vec值[1,2,3,4]作为参数被传入，它的Owner从v1 Move转移到了函数入参v。

第10行，试图打印v1的值。这里就会出错。由于函数调用时发生的所有权的转移，v1已经被编译器认为是一个未初始化的值，不能再使用。

```
error[E0382]: borrow of moved value: `v1`
  --> tmp.rs:10:22
   |
6  |     let v1: Vec::<i32> = vec![1,2,3,4];
   |         -- move occurs because `v1` has type `Vec<i32>`, which does not implement the `Copy` trait
7  |
8  |     take_vec(v1);
   |              -- value moved here
9  |
10 |     println!("{:?}", v1);
   |                      ^^ value borrowed here after move
   |
note: consider changing this parameter type in function `take_vec` to borrow instead if owning the value
 isn't necessary
  --> tmp.rs:1:16
   |
1  | fn take_vec(v: Vec::<i32>){
   |    --------    ^^^^^^^^^^ this parameter takes ownership of the value
   |    |
   |    in this function
   = note: this error originates in the macro `$crate::format_args_nl` which comes from the expansion of
 the macro `println` (in Nightly builds, run with -Z macro-backtrace for more info)
help: consider cloning the value if the performance cost is acceptable
   |
8  |     take_vec(v1.clone());
   |                ++++++++

error: aborting due to previous error

For more information about this error, try `rustc --explain E0382`.
```



编译一下，通过编译错误也能知道错误是什么了。

### 函数返回值

同样，函数返回值也是移动语义。

```rust
fn return_vec() -> Vec::<i32>{
    let inner_v = vec![4,3,2,1];
    inner_v
}

fn main(){

    let outer_v = return_vec();

    println!("{:?}", outer_v);
}
```



看一个简单的例子，return_vec()里新建一个变量inner_v，然后返回innver_v。

注意在返回innver_v的时候，vec[4,3,2,1]的Owner从inner_v Move转移到外部变量outer_v。

如果要改变这种默认转移所有权的操作，就需要用到Copy和Clone这两个trait，下次聊。