从C++到Rust，references，宏和多重引用

原创 Ajonbin AJonbin的杂货铺 2024年03月26日 21:28 美国

之前讲了点操作符自动引用和解引用。

今天来看看上次遗留下来的问题，看看在println!()中应用是怎么处理引用的。

先来看一个简单的例子。

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

    println!("r_v = {:?}", r_v);
    println!("*r_v = {:?}", *r_v);
}
```


我们新建了引用r_v来指向v，然后在调用println!()，分别用了r_v和*r_v。一个是引用本身，一个是解引用后的结果。

```
Standard Output

r_v = [1, 2, 3, 4]
*r_v = [1, 2, 3, 4]
```


输出结果是相同的。

当我们传入的是不同的类型，一个是引用&Vec<i32>，另一个是Vec<i32>。println!()是怎么处理引用的呢？

你自然会想到，println!()会自动解引用，和点操作符一样。

对的，但又没全对。

这要从println!()说起。

println!()我们已经用过很多次了，但是还没有详细讲过它。今天就来简单讲讲。

println!()是一个宏。结尾处的感叹号!就是Rust中宏的标志。之前讲到的异常panic!()也是一个宏。

Rust的宏和C++中的宏#define很相似，都是用来定义一段代码，然后会在编译的时候被展开，替换成宏定义的代码。

回到开始的问题，要想知道引用在println!()是怎么使用的，就要知道println!()展开的代码是怎样的。

在2023年前的Rust版本中，可以通过

```
rustc -Zunpretty=expanded
```

来展开宏。

```
println!("{}", x);
```

展开宏后，可以得到下面的代码。

```
    io::_print(fmt::Arguments::new_v1(
        &["", "\n"],
        &[fmt::ArgumentV1::new(&x, fmt::Display::fmt)],

    ));
```


注意println!()中用的&x，x的引用。

目前最新的std版本1.77.0中，宏println!调用了宏format_args_nl!。而format_args_nl宏已经作为编译器实现的一部分，不能再看到其展开代码了。

下面是println的源码，可以看一眼。

```
macro_rules! println {
    () => {
        $crate::print!("\n")
    };
    ($($arg:tt)*) => {{
        $crate::io::_print($crate::format_args_nl!($($arg)*));
    }};
}
```



```
macro_rules! format_args_nl {
    ($fmt:expr) => {{ /* compiler built-in */ }};
    ($fmt:expr, $($args:tt)*) => {{ /* compiler built-in */ }};
}
```


再回到我们的例子中，我们分别用r_v和*r_v来调用println!()。

由于宏println!()会把参数取引用，而r_v又是一个引用&Vec<i32>，那么在调用println!()时

*r_v 会变成&(*r_v)，是一个引用，&Vec<i32>。

而r_v会变成&(r_v)，是一个引用的引用，&&Vec<i32>。

这就引出了Rust中reference另一个特性，当使用多重引用时，解引用时，Rust会自动找到最后实际的值。

再来个例子

```rust
struct Point {
    x: i32,
    y: i32,
}

fn main(){
    let point = Point { x: 10, y: 100 };
    let r: &Point = &point;
    let rr: &&Point = &r;
    let rrr: &&&Point = &rr;
    let rrrr: &&&&Point = &rrr;

    println!("x={}, y={}", rrrr.x, rrrr.y);
}
```


代码里新建了变量rrrr，它是一个指向point的4重引用。

在使用点操作符解引用的时候，会自动解引用到最后一层。

Standard Output

```
x= 10, y=100
```


上一篇：从C++到Rust，references，dereference，. operator