链滴

Day 01 - 基础 rust

作者：KuMa
原文链接：https://ld246.com/article/1680026833741
来源网站：链滴

变量重影

- 使用 : + 类型 可以 申明变量的类型。

```
fn main() {
    let random: i64 = 20;
    let random = "我是刘博源"; // 重影变量
    println!("{}", random);
}
```

当一个变量可以进行重影（相当于重新申明 重新绑定一个新的变量） 类似于这样的：

```
int a = 2;
String a  = "213";
```

mut 与 重影 的区别是什么?

Rust中 变量默认是不可变的 需要加上 mut 才可以变成可变变量

```
let mut number = 1;
let number2 = 2;
number2 = 3; // 报错 因为 number2 不是可变变量
let number2 = 4; // 不报错  因为number2这里不是修改变量而是重影变量
number = 1324; // number1随便修改 都不会报错（但是你得注意类型）
number = " asdhi o" // 报错 因为类型不对
let mut number = "asdajsghd" // 不报错 因为类型虽然不对 但这是重影
```

数据类型

基础的有 整型（i8 i16 i32 i64 i128 isize）不同进制的整型 浮点型 布尔型 字符型 复合类型

复合类型

```
fn main() {
    let mut tup = (2, 4, "123123", 123.4);
    tup.0 = 3;
    print!("{}", tup.0);
}
```

复合类型 可以存在多个变量类型 并放在一个()中类似我们的元组，但是不同的是他不是用 a[i] 进行索 而是 a.i 进行索引

列表类型

```
fn main() {
    // let mut lst = [2, 4, 5, 6, 7, "2312"]; // 错误：不能存在多个不同类型的变量
    // lst[2] = "123"; // 错误：不能对lst固有的类型进行修改!

    let mut lst = [1,4,5,6];
    lst[2] = 4;
    print!("{}", lst[2]);
}
```

这是打印的方式 [2;5] 表示列表长度为5里面全是2

```
fn main() {
    let mut lst = [2; 5];
    print!("{:?}", lst);
}
// [2, 2, 2, 2, 2]
```

函数

```
fn add(number1: i64, number2: i64) -> i64 { // 形参 返回值 必须声明类型!
    return number1 + number2;
}

fn main() {
```

```
    let _number1: i64 = 2;
    let _number2: i64 = 5;
    println!("{}", add(_number1, _number2));
}
```

函数表达式

```
fn main() {
    let _number1: i64 = 2;
    let _number2: i64 = 5;
    let _add : i64 = {
        let tmp = _number2 + 3;
        _number1 + tmp //这里不能有分号 有分号的是语句 语句的返回值是()!
    }; // 此刻 _add = _number1 + tmp
    println!("{}", _add);
    //println!("{}", tmp); // 拿不到局部的值
}
```

流程控制

if

if语句：

```
fn main(){
    let a = -4;
    if a > 0{
        print!("是正数");
    }else {
        println!("是非正数");
    }
}
```

if 语句后面只能是 Bool 类型的变量。

当然我们还有类似的三元运算符:

```
fn main() {
    let a = 0;
    let b = if a > 0 { "是正数" } else if a == 0 { "是0" } else { "是负数" };

    if a > 0 {
        print!("是正数");
    } else {
        println!("是非正数");
    }

    print!("{}", b);
}
```

loop

```
fn main() {
    let mut number = 0;
    loop {
        print!("{}\n", number);
        number += 1;
        if number == 10000{
            break;
        }
    }
}
```

loop 是死循环的一种 直到他break之前会不断的执行

我们可以给Rust的loop设置一个循环标签:

```
fn main() {
    let mut number = 0;
    'Home: loop {
        print!("{}\n", number);
        number += 1;
        if number == 3 {
            let mut number2 = 5;
            'inner: loop {
                let mut ret: i32 = number + number2;
                print!("- {}\n", ret);
                number2 += 1;
                if ret > 5 {
                    break 'inner;
                }
            }
        }
        if number > 4 {
            break 'Home;
        }
    }
    print!("{}",number);
}
```

for

首先for可以用来简单的遍历

```
fn main() {
    let mut numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    for i in numbers {
        print!("{}\n", i);
    }
}
```

指定范围的遍历

```
fn main() {
    let mut  nums = 1..101;  // 从一到一百 包前不包后
    println!("{:?}", nums); // 1..101
```

```
    for i in nums{
        print!("{}\n", i); // 1 // 2 // 3 .... //100
    }
}
```

我们可以对范围类型的遍历进行倒置

```
fn main() {
    let mut  nums = 1..101;
    println!("{:?}", nums);
    for i in nums.rev() {
        print!("{}\n", i);
    }
}
```

随机数游戏

我们先写一个可以进行输入的代码：

```
use std::io; // 加入io库 std 是一个标准库

fn main() {
    println!("猜数游戏"); // 加入提示
    let mut val = String::new(); // 创建一个新的字符串 但是字符串是空的
    println!("请写入一个数字: ");

    io::stdin().read_line(&mut val).expect("错误!"); // 等待用户进行输入 并将 val的引用交给read_li
e read_line然后在进行对val的修改
    print!("你输入的数字是: {val}");
}
```

use 会预导入一个 prelude 库 除此之外我们需要显式的导入比如 use std::io;

引用默认也是不可变的 所以我们传入read_line的时候不是 read_line(&val) 而是read_line(&mut val)

依赖添加

我们要知道 rust默认本身是很轻便的需要三方库的支持 如何支持三方库？ 使用Cargo的toml就可以
比如我们添加一个随机数的库

```
[dependencies]
rand = "0.8.5"
```

然后我们进行cargo build就可以了 。（有idea就不管~）

然后我们就可以在代码中创建一个随机数了

```
use std::io;
use rand::Rng;

fn main() {
    println!("猜数游戏");
    let mut val  = String::new();
```

```
    println!("请写入一个数字: ");
    let rand_num = rand::thread_rng().gen_range(1..101);
    io::stdin().read_line(&mut val).expect("错误!");
    print!("你输入的数字是: {val} 我猜的数字是{rand_num}");
}
```

Ordering是 std cmp提供的一个枚举 表示两个对象的大小 有 Less 小于 Greater 大于 Equal 等于

然后我们的字符串类型有个cmp方法可以比较self和参数引用变量的大小，他会返回Ordering类型的举。

接着还有就是match表达式, match表达式的使用方法是:

```
match (...){
    A => xxx,
    B => xxxx,
}
```

当括号里的值与A B这些值匹配的时候就会后面的代码 AB这些值我们称之为ARM 一个Match存在多ARM

```
use rand::Rng;
use std::cmp::Ordering;
use std::io;

fn main() {
    println!("猜数游戏");
    let mut val = String::new();
    println!("请写入一个数字: ");
    let rand_num = rand::thread_rng().gen_range(1..101);
    io::stdin().read_line(&mut val).expect("错误!");
    print!("你输入的数字是: {val} 我猜的数字是{rand_num}");
    let val: u32 = val.trim().parse().expect("请输入一个可以被转换成int的数字!");
    match val.cmp(&rand_num) {
        Ordering::Less => print!("太小了"),
        Ordering::Greater => print!("太大了"),
        Ordering::Equal => print!("You Are Win"),
    }
}
```

trim是去除字符串两端的空白

parse是将字符串强制转化类型的数字类型

然后我们可以使用之前的流控语句进行更新:

```
use rand::Rng;
use std::cmp::Ordering;
use std::io;

fn main() {
    println!("猜数游戏");
```

```
    let rand_num = rand::thread_rng().gen_range(1..101);

    'guess_block: loop {
        println!("请写入一个数字: ");
        let mut val = String::new();
        io::stdin().read_line(&mut val).expect("错误!");
        print!("你输入的数字是: {val} 我猜的数字是{rand_num}");
        let val: u32 = val.trim().parse().expect("请输入一个可以被转换成int的数字!");
        match val.cmp(&rand_num) {
            Ordering::Less => print!("太小了"),
            Ordering::Greater => print!("太大了"),
            Ordering::Equal => {
                print!("You Are Win");
                break 'guess_block;
            }
        }
    }
}
```

我们还需呀知道 parse这种方法会返回一个Result类型的枚举 有成功的Ok 和失败的Err， 其中ok有带了一个num 表示parse处理后返回数字

```
use rand::Rng;
use std::cmp::Ordering;
use std::io;

fn main() {
    println!("猜数游戏");

    let rand_num = rand::thread_rng().gen_range(1..101);

    'guess_block: loop {
        println!("请写入一个数字: ");
        let mut val = String::new();
        io::stdin().read_line(&mut val).expect("错误!");
        print!("你输入的数字是: {val} 我猜的数字是{rand_num}");

        let val: u32 = match val.trim().parse() {
            Result::Ok(num) => num,
            Result::Err(_) => {
                print!("你输入的变量不对!");
                continue;
            }
        };

        match val.cmp(&rand_num) {
            Ordering::Less => print!("太小了"),
            Ordering::Greater => print!("太大了"),
            Ordering::Equal => {
                print!("You Are Win");
                break 'guess_block;
            }
        }
    }
```

```
}
```