链滴

Rust 函数

作者：lingyundu

原文链接：https://ld246.com/article/1608280454404

来源网站：链滴

Rust 支持多种编程范式，但更偏向于函数式，函数在 Rust 中是“一等公民”，函数可以作为数据在序中进行传递。跟 C、C++ 一样， Rust 也有一个唯一的程序入口 main 函数。

示例：程序入口 main 函数

```
fn main() {
    println!("Hello, world!");
}
```

Rust 使用 fn 关键字来声明和定义函数，使用 snake case 风格来命名函数，即所有的字母小写并使下划线分隔单词。函数可以有参数，并且每个函数都有返回值。

函数参数

参数的声明方式：参数名 + 冒号 + 参数类型

示例：

```
// 定义一个无参函数
fn function() {
    println!("A function with no parameters.");
}

// 定义一个有参函数
fn hello(name: &str) {
  println!("Hello, {}.", name);
}
```

函数返回值

在 Rust 中所有函数都有返回值，main 函数也不例外，main 函数的返回值是 ()（一个空的元组）。在 Rust 中，当一个函数返回 () 时，可以省略返回值类型的声明。main 函数的完整形式如下：

```
fn main() -> () {
    //statements
  }
```

() 通常被称为 unit 或者 unit type，它其实类似于 C/C++、Java、C#中的 void 类型。

若函数有其他类型返回值，则需使用 -> 显式标明返回值类型。像 C/C++ 或 Java 等语言在函数中需用 return 语句返回一个值，Rust 与它们不一样，Rust 默认将函数中最后一个表达式的结果作为返回。

示例：定义一个有返回值的函数

```
fn main() {
    let x = plus_one(5);

    println!("The value of x is: {}", x);
}

fn plus_one(x: i32) -> i32 {
    x + 1
```

```
}
```

Rust 也有 return 关键字，不过一般用于提前返回。

示例：在函数中使用 return

```
fn main() {
  let a = [1,3,2,5,9,8];
  println!("There is 7 in the array: {}", find(7, &a));
  println!("There is 8 in the array: {}", find(8, &a));
}
fn find(n: i32, a: &[i32]) -> bool {
  for i in a {
    if *i == n {
      return true;
    }
  }
  false // 这里也可以改为 `return false;`，但这就不是 Rust 的编程风格了
}
```

在 Rust 中还有一种“没有返回值”的函数，称之为**发散函数**（diverging function）。其实，它根就不返回，它使用感叹号 ! 作为返回类型。

示例：

```
fn main() {
  println!("hello");
  diverging();
  println!("world");
}
fn diverging() -> ! {
  panic!("This function will never return");
}
```

发散函数一般都以 panic! 宏调用或其他调用其他发散函数结束，所以，调用发散函数会导致当前线崩溃。

高阶函数

高阶函数与普通函数的不同在于，它可以使用一个或多个函数作为参数，可以将函数作为返回值。既函数可以作为参数和返回值，那么函数也应该有一种相对应的数据类型，那就是：函数指针类型。

函数指针类型

函数指针类型使用 fn 关键字定义，在编译时该类型指向一个已知函数参数和返回值类型，但函数体知的函数。

示例：

```
// 函数定义
fn inc(n: i32) -> i32 {
    n + 1
}
```

```
// 使用 `type` 给函数指针类型起一个别名
type IncType = fn(i32) -> i32;

fn main() {
    // 使用函数指针类型 `fn(i32) -> i32`
    let func: fn(i32) -> i32 = inc;
    println!("3 + 1 = {}", func(3));

    // 使用函数指针类型的别名 `IncType`
    let func: IncType = inc;
    println!("4 + 1 = {}", func(4));
}
```

函数作为参数

函数作为参数，其声明与普通参数一样。

示例：高阶函数

```
fn main() {
  println!("3 + 1 = {}", process(3, inc));
  println!("3 - 1 = {}", process(3, dec));
}
fn inc(n: i32) -> i32 {
  n + 1
}
fn dec(n: i32) -> i32 {
  n - 1
}
// process 是一个高阶函数，它有两个参数，一个类型为 `i32`，另一个类型为 `fn(i32)->i32`
fn process(n: i32, func: fn(i32) -> i32) -> i32 {
  func(n)
}
```

函数作为返回值

函数作为返回值，其声明与普通函数的返回值类型声明一样。

示例：

```
fn main() {
    let a = [1,2,3,4,5,6,7];
    let mut b = Vec::<i32>::new();
    for i in &a {
        b.push(get_func(*i)(*i));
    }
    println!("{:?}", b);
 }

 // 若传入的参数为偶数，返回 `inc`，否则返回 `dec`
 fn get_func(n: i32) -> fn(i32) -> i32 {
    fn inc(n: i32) -> i32 {
```

```
        n + 1
    }
    fn dec(n: i32) -> i32 {
        n - 1
    }
    if n % 2 == 0 {
        inc
    } else {
        dec
    }
}
```

相关资料

Rust Programming Language

The Rust Reference

Rust By Example

RustPrimer