# 第四章 函数

本章要点：如何定义函数；传递参数的多种方式；调用本模块、外部模块、其他编程语言定义的函数；闭包的定义和使用。

通过前面几章内容的介绍，我们现在能够用Rust写一些简单的程序。本章我们将介绍函数。函数是Rust编程语言的重要组成部分。函数能够帮助我们封装过程，实现代码重用。结合Rust中的高阶函数、闭包、泛型、trait对象，可以让我们在Rust中轻松实现函数式编程，达到更高的抽象水平。

## 4.1 函数声明与定义

在一个能够生成可执行文件的Rust程序中，我们需要main函数作为该程序执行时的入口。一个main函数定义如ch4\_1.rs所示。

*//这是一个main函数实例*

fn main() {

    println!("demo for defining a main function");

}

ch4\_1.rs

### 4.1.1 函数的基本语法

在Rust中，我们用fn关键字表示函数定义的起始。之后是函数名、参数列表、返回值类型和函数体。在Rust中，函数名通常采用蛇形命名法：用下划线‘\_’分割单词，所有字母小写。参数列表用‘()’表示，内部用逗号‘,’分隔每个参数，每个参数以‘参数名:类型’的格式定义。参数列表可以为空。返回值类型用一个箭头指出，格式为‘-> 返回值类型’。函数体在‘{}’内，是表示函数执行过程的语句块。除函数体外，一个函数定义的其余部分被称作函数签名。一个函数的定义如ch4\_2.rs所示，这个例子定义了一个返回参数增加1的值的函数，参数和返回值的类型都为i32。

*//这是一个函数定义的实例*

fn plus\_one\_i32 (param1: i32) -> i32 {

    return param1 + 1;

}

ch4\_2.rs

在Rust中，我们用return语句来表示函数要返回的变量，如上面的ch4\_2.rs所示。我们还可以用一个表达式作为返回值，如ch4\_3.rs所示。

*//这是一个表达式作返回值的实例*

fn plus\_one\_i32 (param1: i32) -> i32 {

    param1 + 1

}

fn main() {

    println!("{}", plus\_one\_i32(0));

}

ch4\_3.rs

return语句能够让控制流提前返回，而表达式作返回值时没有这一功能，如ch4\_4.rs所示。

fn abs\_i32(x: i32) -> i32 {

    if x > 0 {

*//这里不能用表达式‘x’代替return语句*

*//因为表达式无法让控制流提前返回*

        return x;

    }

    -x

}

ch4\_3.rs

Rust允许函数签名中不标注返回值类型。此时，Rust会给这个函数赋予缺省返回值类型：空元组类型‘()’。ch4\_1中的main函数就是这样的一个例子。

Rust不支持函数声明

返回值类型的一致性e.g. impl Trait

多个返回值的实现

### 4.1.2 函数定义的Shadow机制

Rust不允许相同作用域内存在同名函数，但允许在不同的作用域中定义同名的函数。

不同作用域内的同名函数定义。

## 4.2 参数传递

### 4.2.1 按值传递参数

Rust中值传递的机制及原理

Ownership机制对值传递的影响

### 4.2.2 引用传递参数

Rust中引用传递的机制及原理

Ownership机制对引用传递的影响

带来副作用的&mut

### 4.2.3 生命周期参数

生命周期参数的用法与意义

生命周期参数的使用场景

生命周期参数的特殊使用情况

### 4.2.4 参数的模式匹配

通配符忽略参数

解构元组

### 4.2.5 Trait作为参数传递

## 4.3 函数调用

### 4.3.1 函数调用的基本语法

### 4.3.2 跨模块函数调用

### 4.3.3 外部函数接口绑定（FFI）

`extern`关键字的使用

FFI只能在unsafe代码块中使用

利用C实现变长参数列表

## 4.4 闭包

### 4.4.1 闭包简介与使用

闭包简介及使用

闭包的基本语法

闭包类型的自动推导

闭包的调用

每个闭包都有一个自己的类型

### 4.4.2 环境变量的捕获

Unboxed Closure

闭包的三个特性：Fn, FnOnce, FnMut

被捕获环境变量的所有权

move关键词

### 4.4.3 闭包作参数

### 4.4.4 闭包作返回值

## 4.5 发散函数

发散函数的定义与使用