一.简介与优点

rust特点：

Rust

- 性能
- 安全性
- 开发者友好
- 健壮的生态系统 Cargo

Why

Why Rust?

Rust 可编写更快、更可靠的软件，兼具高层次的易用性与低层次的控制力

- 内存安全，且不牺牲性能
- “编译通过就能正常运行”
- 优秀且不断增长的生态系统
- 令人愉悦的语法和强大的语言特性

用途广泛，包括命令行工具、Web服务、嵌入式设备等

提供现代开发工具，如 Cargo、Rustfmt、RustRover、rust-analyzer (VSCode plugin)

二.HELLO WORLD

在windows上面编译后还有一个.pdb文件，里面包含调试信息

Rust 程序解剖

- 定义函数：**fn main() { }**
  - 没有参数，没有返回
- main 函数很特别：它是每个 Rust 可执行程序最先运行的代码
- 打印文本：**println!("Hello, world!");**
  - Rust 的缩进是 4 个空格而不是 tab
  - println! 是一个 Rust macro（宏）
    - 如果是函数的话，就没有！
  - “Hello World”是字符串，它是 println! 的参数
  - 这行代码以；结尾

三，cargo

Cargo

- Cargo 是 Rust 的构建系统和包管理工具
  - 构建代码、下载依赖的库、构建这些库...

使用 Cargo 创建项目

- 创建项目：cargo new hello_cargo

```
D:\rust\hello_world>cd ..

D:\rust>cargo new hello_cargo
     Creating binary (application) `hello_cargo` package
note: see more `Cargo.toml` keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html
```

文件目录结果：

各个部分介绍：

- 创建项目：cargo new hello_cargo
  - 项目名称也是 hello_cargo
  - 会创建一个新的目录 hello_cargo
    - Cargo.toml
    - src 目录
      - main.rs
    - 初始化了一个新的 Git 仓库，.gitignore
      - 可以使用其它的 VCS 或不使用 VCS：cargo new 的时候使用 --vcs 这个 flag

Cargo.toml

- TOML（Tom's Obvious, Minimal Language）格式，是 Cargo 的配置格式
- [pacakge]，是一个区域标题，表示下方内容是用来配置包（package）的
  - name，项目名
  - version，项目版本
  - authors，项目作者
  - edition，使用的 Rust 版本
- [dependencies]，另一个区域的开始，它会列出项目的依赖项。
- 在 Rust 里面，代码的包称作 crate。

运行项目：cargo run　　检查代码：cargo check

```
D:\rust\guessing_game>cargo run
   Compiling guessing_game v0.1.0 (D:\rust\guessing_game)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 4.25s
     Running `target\debug\guessing_game.exe`
Hello, world!
```

四，一次猜数的测试

main.rs

D:\rust\guessing_game\src\main.rs

```rust
use std::io;
//引用需要的库

fn main() {
    println!("猜数！");

    println!("猜测一个数");

    //let mut foo = 1;
    //let bar = foo; //immutable不可变的量

    //foo = 2; //mutable变量

    let mut guess = String::new();

    io::stdin().read_line(&mut guess).expect("无法读取行");
    //read_line返回值是ok(里面包含结果的值)或者err（里面包含错误的原因）

    println!("你猜的数是：{}", guess);
}

```

运行测试：

```
D:\rust\guessing_game>cargo run
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.01s
     Running `target\debug\guessing_game.exe`
猜数！
猜测一个数
58
你猜的数是： 58
```

5.生成神秘数字

要生成随机数需要下载相应的rand库，这个依赖在cargo.toml里面的dependencies里可以添加

Cargo.toml

```toml
[package]
name = "guessing_game"
version = "0.1.0"
edition = "2024"

[dependencies]
rand = "0.3.14"
```

命令行的build命令 cargo build

```
D:\rust\guessing_game>cargo build
   Compiling winapi v0.3.9
   Compiling libc v0.2.171
   Compiling rand v0.4.6
   Compiling rand v0.3.23
   Compiling guessing_game v0.1.0 (D:\rust\guessing_game)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 1.55s
```

更新库文件版本命令 cargo update

```
D:\rust\guessing_game>cargo update
    Updating crates.io index
     Locking 0 packages to latest Rust 1.86.0 compatible versions
note: pass `--verbose` to see 1 unchanged dependencies behind latest
```

此命令会绕开lock文件，在toml的dependencies里面找到未更新得库版本，进行更新后，再写入lock文件中

6.比较神秘数字和猜测数字

```rust
match guess.cmp(&secret_number) {
    Ordering::Less => println!("猜小了"),  //arm,从上到下匹配
    Ordering::Greater => println!("猜大了"),
    Ordering::Equal => println!("猜对了"),
}
```

1.match会匹配cmp这个函数返回的Ordering枚举类型的值，并进行从上到下的依次匹配

2.Match里面包含多个arm，每个arm后面有其特定匹配模式，如果匹配的上，就执行相应arm里面的代码

```rust
let guess: u32 = guess.trim().parse().expect(msg: "请输入一个数字");
//parse返回的类型是result类型
```

：u32 指定其类型

trim （）：去掉两端的空白，空格和回车等

parse （）：解析成整数，失败的话会执行expect，输出括号中的话

7.多次猜测

```rust
loop{
    println!("猜测一个数");

    let mut guess: String = String::new();

    io::stdin().read_line(buf: &mut guess).expect(msg: "无法读取行");
```

```
D:\rust\guessing_game>cargo run
   Compiling guessing_game v0.1.0 (D:\rust\guessing_game)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 1.28s
     Running `target\debug\guessing_game.exe`
猜数游戏！
```

```rust
println!("猜测一个数");

let mut guess: String = String::new();

io::stdin().read_line(buf: &mut guess).expect(msg: "无法读取行");
//read_line返回值是ok(里面包含结果的值)或者err（里面包含错误的原因）

let guess: u32 = guess.trim().parse().expect(msg: "请输入一个数字");
//parse返回的类型是result类型

println!("你猜的数是：{}", guess);

match guess.cmp(&secret_number) {
Ordering::Less => println!("猜小了"),  //arm,从上到下匹配
Ordering::Greater => println!("猜大了"),
Ordering::Equal => {println!("猜对了");
    break;
    },
}
}
} fn main
```

```
D:\rust\guessing_game>cargo run
   Compiling guessing_game v0.1.0 (D:\rust\guessing_game)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 1.28s
     Running `target\debug\guessing_game.exe`
猜数游戏！
神秘数字是:61
猜测一个数
56
你猜的数是： 56
猜小了
猜测一个数
98
你猜的数是： 98
猜大了
猜测一个数
61
你猜的数是： 61
猜对了
```

使用loop{}来进行循环操作，然后在arm的equal匹配的执行代码块中加入break语句即可跳出循环

```rust
let guess: u32 = match guess.trim().parse() {
    Ok(num: u32) => num,
    Err(_) => continue,
};
```

因为parse结果返回的result类型，是枚举类型，result有两个变体，ok和err，然后正好可以用match来针对枚举的类型匹配

这里为了防止输入字符串类型时直接程序崩溃，用match来替代expect的作用，在用户输入不符合要求时重新进入开头的循环中

8.变量与可变性

Let 声明变量，默认是不可变的，前面加mut即成为可变化的变量

变量与常量

- 常量（constant），常量在绑定值以后也是不可变的，但是它与不可变的变量有很多区别：
  - 不可以使用 mut，常量永远都是不可变的
  - 声明常量使用 const 关键字，它的类型必须被标注
  - 常量可以在任何作用域内进行声明，包括全局作用域
  - 常量只可以绑定到常量表达式，无法绑定到函数的调用结果或只能在运行时才能计算出的值
- 在程序运行期间，常量在其声明的作用域内一直有效
- 命名规范：Rust 里常量使用全大写字母，每个单词之间用下划线分开，例如：
  - MAX_POINTS
- 例子：const MAX_POINTS: u32 = 100_000;

隐藏

Shadowing（隐藏）

- 可以使用相同的名字声明新的变量，新的变量就会 shadow（隐藏）之前声明的同名变量
  - 在后续的代码中这个变量名代表的就是新的变量
  - （例子）
- shadow 和把变量标记为 mut 是不一样的：
  - 如果不使用 let 关键字，那么重新给非 mut 的变量赋值会导致编译时错误
  - 而使用 let 声明的同名新变量，也是不可变的
  - 使用 let 声明的同名新变量，它的类型可以与之前不同

```rust
fn main() {
    let spaces = "    ";
    let spaces = spaces.len();

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

这里就可以是不同类型

好处：可以灵活转变变量类型，不必声明多个变量

```rust
fn main() {
    let x = 5;
    let x = x + 1;

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

通过再次使用**let**声明同名的量，实现隐藏，此处打印的是后面声明的x（也就是之后所有引用的x都是后面定义的那个），第一个x在x+1中已经被使用

数据类型

1.标量类型：
- 整数类型
- 浮点类型
- 布尔类型
- 字符类型

整数类型：默认是i32

Length	Signed	Unsigned
8-bit	i8	u8

进制字面量：

Number literals	Example

整数类型：默认是i32

Length	Signed	Unsigned
8-bit	i8	u8
16-bit	i16	u16
32-bit	i32	u32
64-bit	i64	u64
128-bit	i128	u128
arch	isize	usize

进制字面量：

Number literals	Example
Decimal	98_222
Hex	0xff
Octal	0o77
Binary	0b1111_0000
Byte (u8 only)	b'A'

- isize 和 usize 类型的位数由程序运行的计算机的架构所决定：
  - 如果是 64 位计算机，那就是 64 位的

浮点类型

- Rust 有两种基础的浮点类型，也就是含有小数部分的类型
  - f32，32位，单精度
  - f64，64位，双精度
- Rust 的浮点类型使用了 IEEE-754 标准来表述
- f64 是默认类型，因为在现代 CPU 上 f64 和 f32 的速度差不多，而且精度更高。

布尔类型

- Rust 的布尔类型也有两个值：true 和 false
- 一个字节大小
- 符号是 bool

字符类型

- Rust 语言中 char 类型被用来描述语言中最基础的单个字符。
- 字符类型的字面值使用单引号
- 占用 4 字节大小
- 是 Unicode 标量值，可以表示比 ASCII 多得多的字符内容：拼音、中日韩文、零长度空白字符、emoji 表情等。
  - U+0000 到 U+D7FF
  - U+E000 到 U+10FFFF
- 但 Unicode 中并没有“字符”的概念，所以直觉上认为的字符也许与 Rust 中的概念并不相符

2.

3.3 复合类型

- 复合类型可以将多个值放在一个类型里。
- Rust 提供了两种基础的复合类型：元组（Tuple ）、数组

Tuple

- Tuple 可以将多个类型的多个值放在一个类型里
- Tuple 的长度是固定的：一旦声明就无法改变

```
fn main() {
    let tup: (i32, f64, u8) = (500, 6.4, 1);

    println!("{}, {}, {}, ", tup.0, tup.1, tup.2);
}
```

声明：小括号+逗号

访问：name.(n-1)

```
fn main() {
    let tup: (i32, f64, u8) = (500, 6.4, 1);

    let (x, y, z) = tup;

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

获取tuple的值:使用匹配模式解构

数组

- 数组也可以将多个值放在一个类型里
- 数组中每个元素的类型必须相同
- 数组的长度也是固定的

数组的用处

- 如果想让你的数据存放在 stack（栈）上而不是 heap（堆）上，或者想保证有固定数量的元素，这时使用数组更有好处
- 数组没有 Vector 灵活（以后再讲）。
  - Vector 和数组类似，它由标准库提供
  - Vector 的长度可以改变
  - 如果你不确定应该用数组还是 Vector，那么估计你应该用 Vector。

数组的类型

- 数组的类型以这种形式表示：[类型；长度]
  - 例如： let a: [i32; 5] = [1, 2, 3, 4, 5];

另一种声明数组的方法

- 如果数组的每个元素值都相同，那么可以在：
  - 在中括号里指定初始值
  - 然后是一个“；”
  - 最后是数组的长度
- 例如： let a = [3; 5]; 它就相当于：let a = [3, 3, 3, 3, 3];

内存安全性：

- 如果访问的索引超出了数组的范围，那么：
  - 编译会通过
  - 运行会报错（runtime 时会 panic）
    - Rust 不会允许其继续访问相应地址的内存