

02 上手体验：猜数字游戏

这一节，我们将通过一个简单的猜数字小游戏来体验 Rust 的魅力。由于我们还没有讲解 Rust 的详细语法、类型系统等，所以这个小游戏中的代码可能不容易看懂。不过没关系，先直观感受一下，跟着把代码敲完能正确运行就可以了。

这个游戏的逻辑是，随机生成 1-100 之间的一个整数。然后让用户猜测一个数字，如果用户猜测的数字比生成的数字大，则反馈用户猜大了；如果用户猜测的数字比生成的数字小，则反馈用户猜小了。直到用户猜测的数字和生成的数字一样大时，告诉用户猜中了并且结束游戏。

使用伪代码描述一下这个小游戏：

```
1 生成一个随机数
2 LOOP
3     让用户输入一个数值
4     IF 输入的内容无法转换成有效的数值 THEN
5         打印信息并终止
6     ENDIF
7     将用户输入的字符转换成一个数值类型
8     IF 用户输入的数值和生成的随机数相等 THEN
9         打印信息并退出，游戏结束
10    ELSE
11        打印信息，进入下一轮循环
12    ENDIF
13 ENDLOOP
```

创建项目

首先，使用 `cargo` 命令创建项目：`cargo new guessing_game`。项目创建完成之后，使用编辑器打开这个项目。

读取输入

编辑 `main.rs` 文件，输入如下内容：

```

1 use std::io;
2
3 fn main() {
4     println!("Guess the number!");
5     println!("Please input your guess.");
6
7     let mut guess = String::new();
8     io::stdin()
9         .read_line(&mut guess)
10        .expect("Failed to read line");
11
12     println!("You guessed: {}", guess);
13 }
```

暂时先不关心如何生成随机数，上面这段代码中，先在命令行窗口中输出两句话，然后让用户随便输入一些内容，回车，再把用户输入的内容打印出来，然后程序终止。如果使用 Node.js 代码实现，则类似下面的代码：

对应的 Node.js 代码

```

1 const readline = require('node:readline/promises');
2
3 (async function() {
4     const rl = readline.createInterface({
5         input: process.stdin,
6         output: process.stdout
7     });
8
9     console.log("Guess the number");
10
11     const guess = await rl.question("Please input your
12     guess.\n");
13     rl.close();
14
15     console.log(`You guessed: ${guess}`);
16 })();
```

`fn main()` 是 Rust 可执行项目（二进制项目，Binary Package）的入口函数，这个是比较常见的习惯；

行 4、行 5 以及行 12 都出现了 `println!`，顾名思义，这个是在标准输出（stdout）中输出内容用的。但是这个看起来像是函数的，以感叹号（!）结尾的，在 Rust 中叫做“宏”（macro）。宏这个概念在 C/C++ 中都有，它在编译的时候，根据宏的规则，将代码中的宏替换成正常的函数调用或者对应的代码。Rust 中的 `println!` 大致等同于 Node.js 中的 `console.log`。

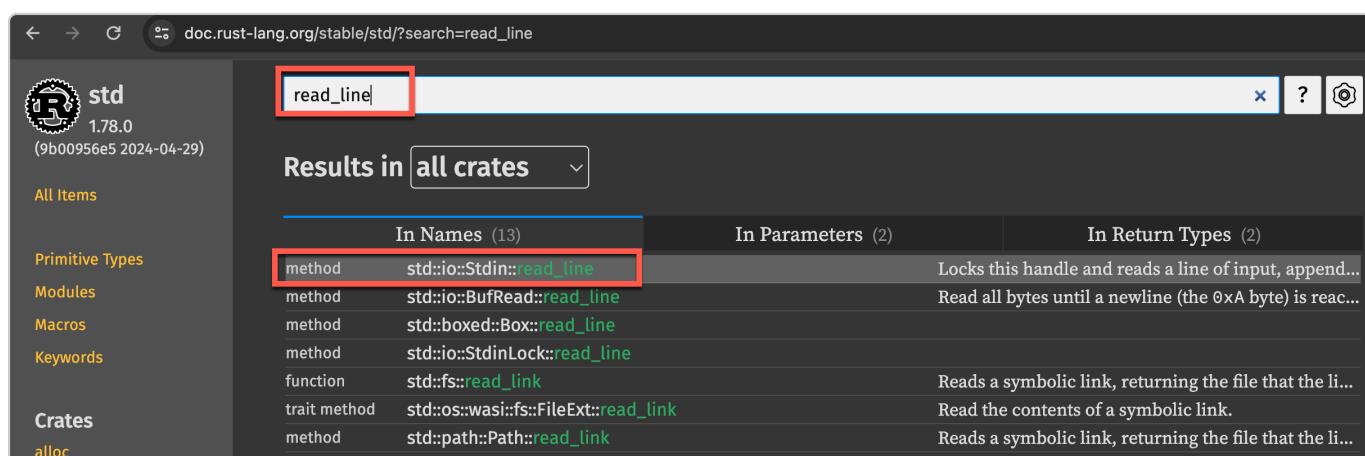
行 7 使用 `let` 关键字定义了一个名为 `guess` 的变量，并且通过 `mut` 关键字表明该变量的值是可变的（mutable）且其对应的初始值为 `String::new()` 的返回值。等同于 Node.js 中的 `let guess = ""`。

`String::new()` 是调用类型 `String` 上的关联函数（Associated Function）`new`。在 Rust 中，关联函数类似于 JavaScript 中的静态函数，它和类型有关，和类型的具体实例（instance）无关。例如，JavaScript 中，`String.fromCharCode` 和具体的字符串实例无关，它是 `String` 这个类的静态方法。

行 8 调用了 `io` 模块中的 `stdin()` 函数。`io` 模块的全名叫做 `std::io`，但是为了代码简洁，我们在行 1 使用 `use` 关键字引入 `std::io` 模块，这样在下面的代码中，直接使用 `io` 即可。Rust 中的 `use` 类似于 Node.js 代码中，我们使用 `require` (CJS) 或者 `import` (ESM) 将其他的库引入到当前代码。

`io::stdin()` 函数的返回值是一个 `Stdin` 结构体（Struct），它有一个方法叫做 `read_line`，用来从标准输入读取一行输入内容。这里的一行是指用户键入回车字符之前的所有内容。等同于 Node.js 中的 `rl.question`。

在浏览器中访问 <https://docs.rs/std>，即可打开 Rust 标准库的文档，搜索 `read_line`：



The screenshot shows the Rust documentation search interface. The URL in the address bar is https://docs.rs/std?search=read_line. The search term 'read_line' is entered in the search input field, which is highlighted with a red box. The results are displayed under the heading 'Results in all crates'. The first result is 'In Names (13)':

| | In Names (13) | In Parameters (2) | In Return Types (2) |
|--------------|--|--|---------------------|
| method | <code>std::io::Stdin::read_line</code> | Locks this handle and reads a line of input, append... | |
| method | <code>std::io::BufRead::read_line</code> | Read all bytes until a newline (the 0xA byte) is reac... | |
| method | <code>std::boxed::Box::read_line</code> | | |
| method | <code>std::io::StdinLock::read_line</code> | | |
| function | <code>std::fs::read_link</code> | Reads a symbolic link, returning the file that the li... | |
| trait method | <code>std::os::wasi::fs::FileExt::read_link</code> | Read the contents of a symbolic link. | |
| method | <code>std::path::Path::read_link</code> | Reads a symbolic link, returning the file that the li... | |
| method | <code>std::fc::File::read_link</code> | | |

The sidebar on the left includes links for 'std' version 1.78.0, 'All Items', 'Primitive Types', 'Modules', 'Macros', 'Keywords', and 'Crates' (with 'alloc' listed).

点击进入 `read_line` 的文档：

[[-](#)] `pub fn read_line(&self, buf: &mut String) -> Result<usize>`

[source](#)

Locks this handle and reads a line of input, appending it to the specified buffer.

For detailed semantics of this method, see the documentation on [BufRead::read_line](#).

Examples

```
use std::io;

let mut input = String::new();
match io::stdin().read_line(&mut input) {
    Ok(n) => {
        println!("{} bytes read", n);
        println!("{}", input);
    }
    Err(error) => println!("error: {}", error),
}
```

You can run the example one of two ways:

- Pipe some text to it, e.g., `printf foo | path/to/executable`
- Give it text interactively by running the executable directly, in which case it will wait for the Enter key to be pressed before continuing

Rust 的文档非常好用，它通常会在文档中附带一些示例代码让我们参考。所以要学会使用文档这个强有力的武器。

从文档中可以看出 `read_line` 函数有 2 个参数，分别是：`&self` 和 `&mut String`。第一个参数 `&self` 我们先不管它，第二个参数中用到了 `&mut`。`&`（读作 [ref]）表示这是一个引用（Reference），`&mut`（读作 [ref mju:t]）表示是一个可变引用。这里我们又一次见到了 `mut` 这个关键字。

行 10 中出现了一个在其他语言中很少见到的 `expect`。由于 `read_line` 函数返回的结果是 `Result<usize>`，表示如果读取成功，则返回读取的字节数量，如果读取失败，则返回错误信息。`expect` 表示，如果读取失败了，那么打印 "Failed to read line"。

那么上面的代码可以省略 `expect` 吗？来试一下，把代码中的 `expect` 去掉：

```

1 use std::io;
2
3 fn main() {
4     println!("Guess the number!");
5     println!("Please input your guess.");
6
7     let mut guess = String::new();
8     io::stdin()
9         .read_line(&mut guess);
10
11    println!("You guessed: {}", guess);
12 }
```

然后使用 `cargo run` 来尝试编译并运行这个项目，会发现虽然编译通过了，但是会收到一条警告信息：

```

1 warning: unused `Result` that must be used
2 --> src/main.rs:8:5
3 |
4 8 | /     io::stdin()
5 9 | |         .read_line(&mut guess);
6  | |_____^
7  |
8  = note: this `Result` may be an `Err` variant, which should be
handled
9  = note: `#[warn(unused_must_use)]` on by default
10 help: use `let _ = ...` to ignore the resulting value
11 |
12 8 |     let _ = io::stdin()
13 |     ++++++
14
15 warning: `guessing_game` (bin "guessing_game") generated 1
warning
16     Finished `dev` profile [unoptimized + debuginfo] target(s)
in 0.00s
17     Running `target/debug/guessing_game`
18 Guess the number!
19 Please input your guess.
```

```
20 23
21 You guessed: 23
```

Rust 建议你对调用函数的返回值进行检查，因为这个返回值可能是一个错误，而不是预期的返回。

生成一个随机数

Rust 的标准库中没有生成随机数的相关类型或者函数，那么怎么办呢？现在就需要去 <https://crates.io> 查找有没有可用的库了。

<https://crates.io> 是 Rust 的库的集中地，类似 npmjs.com 之于 Node.js，Maven Repository 之于 Java。crate 这个词的本意是货箱、板条箱，而 cargo 的本意是货物，而且是大宗的，比如货机、货轮所运输的货物。在本课程中就保留这些词，不做翻译。

在 crates.io 中搜索 rand：

The screenshot shows the crates.io website interface. At the top, there is a dark header bar with the 'crates.io' logo, a search input field containing 'rand', a magnifying glass icon, and links for 'Browse All Crates' and 'Log in with GitHub'. Below the header, the main content area has a light background. The title 'Search Results for 'rand'' is displayed prominently. Underneath, it says 'Displaying 1-10 of 1372 total results' and 'Sort by Relevance'. The first result listed is 'rand v0.8.5', which includes a download icon, an 'All-Time' download count of 316,855,509, a 'Recent' download count of 34,340,462, and an 'Updated' timestamp of 'about 2 months ago'. Below the result are links for 'Homepage', 'Documentation', and 'Repository'.

这个 `rand` crate 就是我们要用来生成随机数的第三方库。有两种方式可以把这个库加入到我们项目中来。

方式一：直接编辑 `Cargo.toml` 文件

我们可以直接编辑 `Cargo.toml` 文件，在 `[dependencies]` 段落中增加一行：

```

1 [package]
2 name = "guessing_game"
3 version = "0.1.0"
4 edition = "2021"
5
6 [dependencies]
7 rand = 0.8.5

```

然后保存文件即可。

方式二：使用 `cargo add` 命令

打开命令行窗口，进入到项目目录，然后执行 `cargo add rand` 即可将这个库加入到项目中来。

以上两种方式没有区别，可以都尝试着用一下。

接下来，我们在 `main.rs` 中加入生成随机数的代码：

```

1 use std::io;
2
3 use rand::Rng;
4
5 fn main() {
6     println!("Guess the number!");
7
8     let secret_number = rand::thread_rng().gen_range(1..=100);
9     println!("The secret number is: {}", secret_number);
10
11    println!("Please input your guess.");
12
13    let mut guess = String::new();
14    io::stdin()
15        .read_line(&mut guess)
16        .expect("Failed to read line");
17
18    println!("You guessed: {}", guess);
19 }

```

行 8、行 9 是我们新加入的代码。

行 8 表示使用新加入的 `rand crate` 生成一个 1 到 100 之间的随机数。`1..=100` 在 Rust 中表示一个 `RangeInclusive` 类型的值。`1..=100` 表示包含 100，`1..100` 表示不包含 100。类似的，在 Node.js 中对应的代码就是 `1 + Math.floor(Math.random() * 100);`。

为了知道生成的随机数是多少，在行 9 把它打印到屏幕上。

对应的 Node.js 代码

使用 Node.js 生成随机数就比较简单了，因为 `Math` 对象自带了：

```

1 const readline = require('node:readline/promises');
2
3 (async function() {
4     console.log("Guess the number");
5     const secretNumber = 1 + Math.floor(Math.random() * 100);
6     console.log(`The secret number is: ${secretNumber}`);
7
8     const rl = readline.createInterface({
9         input: process.stdin,
10        output: process.stdout
11    });
12
13     const guess = await rl.question("Please input your
14    guess.\n");
15     rl.close();
16
17     console.log(`You guessed: ${guess}`);
18 })();

```

将输入的字符串转换成数字

在“读取输入”一节的代码中，我们使用 `String` 类型的一个变量来接收输入的值。但是上一节生成的随机数是一个数值类型。要想将输入的值和生成的随机数进行比较，需要将输入的字符串转换成数字。

```

1 use std::io;

```

```

2
3 use rand::Rng;
4
5 fn main() {
6     println!("Guess the number!");
7
8     let secret_number = rand::thread_rng().gen_range(1..=100);
9     println!("The secret number is: {}", secret_number);
10
11    println!("Please input your guess.");
12
13    let mut guess = String::new();
14    io::stdin()
15        .read_line(&mut guess)
16        .expect("Failed to read line");
17
18    let guess: u32 = guess.trim().parse().expect("Please type a
19 valid number");
20    println!("You guessed: {}", guess);
21 }
```

行 18 是新加入的代码。

仔细阅读上面的代码，你会发现一个问题，明明在行 13 声明了类型为 `String` 的变量 `guess`，为什么在行 18 可以再次声明 `u32` 类型的变量 `guess` 呢？这种行为在 Rust 中被称为变量遮蔽（**Shadowing Variable**）。

为什么行 18 中讲字符串转换成数值之前要先 `trim()` 呢？因为在 `read_line` 方法中读取到的字符串，除了包含可见的、我们输入的字符之外，还在字符串尾部包含一个换行符（newline character）`"\n"`，所以要调用 `trim()` 函数去掉结尾的换行符再进行转换，否则一个带有换行符的字符串是无法正确转换成一个数值的。

这里我们再一次遇到 `expect`，很好理解，因为不是所有的输入字符串都可以成功的转换成一个数值的。

那么，Rust 如何知道我们需要把字符串类型的 `guess` 转换成什么类型呢？答案就在于行 18 中的 `let guess: u32`。因为 Rust 具有类型推断的能力（Type Inferencing）。而在 Node.js 中，需要使用 `parseInt` 函数把一个字符串转换成一个整数。

经过上面的代码，我们得到了一个数值类型的 `guess`。

对应的 Node.js 代码

```

1  const readline = require('node:readline/promises');
2
3  (async function() {
4      console.log("Guess the number");
5
6      const secretNumber = 1 + Math.floor(Math.random() * 100);
7      console.log(`The secret number is: ${secretNumber}`);
8
9      const rl = readline.createInterface({
10          input: process.stdin,
11          output: process.stdout
12      });
13
14      const guess = await rl.question("Please input your
15      guess.\n");
16
17      rl.close();
18
19      const guessNumber = parseInt(guess);
20      if (isNaN(guessNumber)) {
21          console.log("Please type a valid number");
22          return;
23      }
24  })();

```

需要注意的，在 Rust 中，将一个字符串转换成数值，是要求这个字符串中所有的字符都参与转换之后，能够转换成一个有效的数值；而 JavaScript 的 `parseInt` 函数是不要求全部字符都参与转换的。

加入比较逻辑

到现在为止，我们生成了一个随机数，也成功的将从标准输入中读取到的字符串转换成了一个数值。接下来，就要进行比较了。

```

1  use std::io;
2
3  use rand::Rng;
4
5  fn main() {
6      println!("Guess the number!");
7
8      let secret_number = rand::thread_rng().gen_range(1..=100);
9      println!("The secret number is: {}", secret_number);
10
11     println!("Please input your guess.");
12
13     let mut guess = String::new();
14     io::stdin()
15         .read_line(&mut guess)
16         .expect("Failed to read line");
17
18     let guess: u32 = guess.trim().parse().expect("Please type a
valid number");
19     println!("You guessed: {}", guess);
20
21     match guess.cmp(&secret_number) {
22         std::cmp::Ordering::Less => println!("Too small!"),
23         std::cmp::Ordering::Equal => println!("Bingo! You
win!"),
24         std::cmp::Ordering::Greater => println!("Too large"),
25     }
26 }
```

从行 21 开始，是新加入的代码。出乎意料的是，这里我们没有见到其他语言中比较大小常用的运算符：`<`, `>`, `=`，而是使用了一个新的关键字 `match`。这是 Rust 中强大的模式匹配（Pattern Matching）。在模式匹配中，每个分支被称做一个“arm”。

在上面的例子中，`guess.cmp` 返回的结果是一个枚举（Enumeration），表示 `guess` 和 `secret_number` 的比较结果。很显然，两个数值比较的结果无外乎就是大于、小于和等于，我们只需根据对应的结果打印对应的消息就可以了。

对应的 Node.js 代码

```
1 const readline = require('node:readline/promises');
2
3 (async function() {
4     const secretNumber = 1 + Math.floor(Math.random() * 100);
5     console.log(`The secret number is: ${secretNumber}`);
6
7     console.log("Guess the number");
8
9     const rl = readline.createInterface({
10         input: process.stdin,
11         output: process.stdout
12     });
13
14     const guess = await rl.question("Please input your
15     guess.\n");
16     rl.close();
17
18     console.log(`You guessed: ${guess}`);
19
20     const guessNumber = parseInt(guess);
21     if (isNaN(guessNumber)) {
22         console.log("Please type a valid number");
23         return;
24     }
25
26     if (guessNumber < secretNumber) {
27         console.log("Too small!");
28     }
29
30     if (guessNumber > secretNumber) {
31         console.log("Too large!");
32     }
33 })();
```

```

33     if (guessNumber === secretNumber) {
34         console.log("Bingo! You win!");
35     }
36 })();

```

加入循环

上面的代码已经完成了我们这个猜数字小游戏的核心逻辑了，但是随机出来的数字，我们只有一次猜测的机会，显然被猜中的几率太小了。所以，我们加入循环，直到用户猜中，否则一直让用户猜。

```

1 use std::io;
2
3 use rand::Rng;
4
5 fn main() {
6     println!("Guess the number!");
7
8     let secret_number = rand::thread_rng().gen_range(1..=100);
9     println!("The secret number is: {}", secret_number);
10
11     loop {
12         println!("Please input your guess.");
13
14         let mut guess = String::new();
15         io::stdin()
16             .read_line(&mut guess)
17             .expect("Failed to read line");
18
19         let guess: u32 = guess.trim().parse().expect("Please
type a valid number");
20         println!("You guessed: {}", guess);
21
22         match guess.cmp(&secret_number) {
23             std::cmp::Ordering::Less => println!("Too small!"),
24             std::cmp::Ordering::Equal => {
25                 println!("Bingo! You win!");
26                 break;
27             }
28         }
29     }
30 }

```

```

27     },
28     std::cmp::Ordering::Greater => println!("Too
29         large"),
30     }
31 }
```

使用 `loop` 关键字，循环读取用户输入，直到用户猜中为止。行 26 的 `break` 中止循环。

对应的 Node.js 代码

```

1 const readline = require('node:readline/promises');
2
3 (async function() {
4     const secretNumber = 1 + Math.floor(Math.random() * 100);
5     console.log(`The secret number is: ${secretNumber}`);
6
7     for(;;) {
8         console.log("Guess the number");
9
10        const rl = readline.createInterface({
11            input: process.stdin,
12            output: process.stdout
13        });
14
15        const guess = await rl.question("Please input your
16        guess.\n");
17        rl.close();
18
19        console.log(`You guessed: ${guess}`);
20
21        const guessNumber = parseInt(guess);
22        if (isNaN(guessNumber)) {
23            console.log("Please type a valid number");
24            return;
25        }
26
27        if (guessNumber < secretNumber) {
```

```
27         console.log("Too small!");
28     }
29
30     if (guessNumber > secretNumber) {
31         console.log("Too large!");
32     }
33
34     if (guessNumber === secretNumber) {
35         console.log("Bingo! You win!");
36         break;
37     }
38 }
39 })();
```

至此，猜数字游戏完成。

回顾

本节我们上手完成了一个猜数字的小游戏，简单回顾一下：

1. 读取用户输入
2. 引入第三方库：rand crate
3. 数据类型转换
4. Shadowing Variable
5. 模式匹配（Pattern Matching）
6. loop 循环与 break 中止循环