Rust. Особенности языка

Нгуен Тхи Хань Хуен
Королева Дарья

Срезы (Slices)

Срез — это ссылка на участок данных, например, часть массива или строки.

```rust
fn main() {
    let arr = [1, 2, 3, 4, 5];
    let slice = &arr[1..4]; //
    println!("{:?}", slice);
}
```

```rust
let s = "Привет";
let slice = &s[0..6]; // "При" - первые 6 байт (каждый символ кириллицы = 2 байта)

// ПАНИКА во время выполнения:
// let bad_slice = &s[0..5]; // Будет паника, потому что режем символ пополам
// Rust гарантирует, что срезы строк всегда на корректных границах символов UTF-8
```

Box

Box - это умный указатель на расположенное в куче значение типа T

```
// ОШИБКА КОМПИЛЯЦИИ! Компилятор не знает размер List.
enum List {
    Cons(i32, List), // Сколько памяти выделить?
    Nil,
}
```

Позволяет разместить данные в куче, а сам указатель — в стеке

```
enum List {
    Cons(i32, Box<List>), // Теперь размер фиксирован: i32 + указатель
    Nil,
}

let list = Cons(1, Box::new(Cons(2, Box::new(Nil))));
// В стеке: [значение=1, указатель] -> [значение=2, указатель] -> Nil
// В куче:    {значение=2, указатель}    {Nil}
```

Сырые указатели (raw pointers)

Rust предоставляет два типа низкоуровневых указателей:

*const T — неизменяемый указатель

*mut T — изменяемый указатель

```
let mut x = 10;
let x_ptr = &mut x as *mut i32;
```

&mut x — обычная безопасная ссылка

as *mut i32 преобразует её в сырой указатель

Небезопасные блоки unsafe

unsafe позволяет выполнять операции, которые компилятор Rust обычно запрещает:

- разыменование сырых указателей,
- вызов небезопасных функций,
- изменение статических переменных,
- использование extern функций из C,
- реализация небезопасных трейтов.

```
unsafe {
    println!("Absolute value of -3 according to C: {}", abs(-3));
}
```

Работа в блоке unsafe

```rust
fn main() {
    let mut num = 10;

    // Создаем сырые указатели в безопасном коде
    let r1 = &num as *const i32;
    let r2 = &mut num as *mut i32;

    // Безопасный код может продолжаться здесь

    unsafe {
        // Внутри unsafe берем на себя ответственность
        println!("r1 = {}", *r1); // 10
        *r2 = 20; // Модифицируем память через *mut
        println!("r2 = {}", *r2); // 20
        println!("num = {}", num); // 20 (это та же переменная)
    }
    // После блока unsafe мир снова безопасен.
}
```

I/O

Базовый ввод-вывод - макросы println! и print!

```rust
fn main() {
    // Простой вывод
    println!("Hello, Rust!"); // Добавляет перевод строки

    // Форматирование как в Python
    let name = "Alice";
    let age = 30;
    println!("Name: {}, Age: {}", name, age);

    // Позиционные аргументы
    println!("{1} is {0} years old", age, name);

    // Именованные аргументы
    println!("{name} is {age} years old", name=name, age=age);

    // Форматирование чисел
    let pi = 3.1415926535;
    println!("Pi is {:.2}", pi); // "Pi is 3.14"
    println!("Binary: {:b}, Hex: {:x}", 42, 42);

    // Вывод без перевода строки
    print!("Loading... ");
    // что-то происходит
    println!("Done!");
}
```

Чтение из стандартного ввода (stdin)

```rust
use std::io;

fn main() {
    let mut input = String::new();

    println!("Введите ваше имя:");
    io::stdin()
        .read_line(&mut input)
        .expect("Не удалось прочитать строку");

    println!("Привет, {}!", input.trim());
}
```

write!

```rust
use std::io::{self, Write};

fn main() -> io::Result<()> {
    let mut buffer = Vec::new(); // Можно писать в вектор, как в файл

    write!(&mut buffer, "Привет, мир!\n")?;
    write!(&mut buffer, "Второе сообщение\n")?;

    // Теперь buffer содержит байты строк
    println!("Записано {} байт", buffer.len());

    // Выведем содержимое buffer как строку
    let output = String::from_utf8(buffer).unwrap();
    print!("{}", output);

    Ok(())
}
```

Работа с файлами

Структура File представляет открытый файл (она является обёрткой над файловым дескриптором) и даёт возможность чтения/записи этого файла.

```rust
use std::fs::File;
use std::io::{self, Read};

fn read_file_to_string(path: &str) -> io::Result<String> {
    let mut file = File::open(path)?;
    let mut contents = String::new();
    file.read_to_string(&mut contents)?;
    Ok(contents)
}
```

```rust
use std::fs::File;
use std::io::{self, Write};

fn write_string_to_file(path: &str, data: &str) -> io::Result<()> {
    let mut file = File::create(path)?;
    file.write_all(data.as_bytes())?;
    Ok(())
}
```

Пример копирования файла

```rust
use std::fs::File;
use std::io::{self, Read, Write};

fn copy_file(src: &str, dst: &str) -> io::Result<()> {
    let mut src_file = File::open(src)?;
    let mut dst_file = File::create(dst)?;

    let mut buffer = [0; 4096]; // Буфер 4 КБ

    loop {
        let bytes_read = src_file.read(&mut buffer)?;
        if bytes_read == 0 {
            break;
        }
        dst_file.write_all(&buffer[..bytes_read])?;
    }

    println!("Файл скопирован из {} в {}", src, dst);
    Ok(())
}

fn main() {
    if let Err(error) = copy_file("input.txt", "output.txt") {
        eprintln!("Ошибка копирования: {}", error);
    }
}
```