Rust. Конструкции потока управления и обработка ошибок

Нгуен Тхи Хань Хуен
Королева Дарья

Условные конструкции

```rust
fn main() {
    let x = 5;
    if x > 0 {
        println!("Положительное");
    } else {
        println!("Неположительное");
    }
}
```

в rust if возвращает значение

```rust
let number = if x % 2 == 0 { "чётное" } else { "нечётное" };
println!("{}", number);
```

Циклы: while, loop, for

```rust
// Бесконечный цикл
loop {
    println!("Вечный цикл");
    break; // выход вручную
}

// Условный цикл
let mut i = 0;
while i < 5 {
    println!("i = {}", i);
    i += 1;
}

// Итерация по коллекции
for x in 0..5 {
    println!("{}", x);
}
```

В Rust цикл `for` основан на итераторах

Циклы

```rust
fn main() {
    let (mut a, mut b) = (100, 52);
    let result = loop {
        if a == b {
            break a;
        }
        if a < b {
            b -= a;
        } else {
            a -= b;
        }
    };
    println!("{result}");
}
```

Для loop break может принимать опциональное выражение, которое становится значением выражения loop.

Для незамедлительного перехода к следующей итерации используется ключевое слово **continue**.

Циклы

```rust
fn main() {
    'outer: for x in 1..5 {
        println!("x: {x}");
        let mut i = 0;
        while i < x {
            println!("x: {x}, i: {i}");
            i += 1;
            if i == 3 {
                break 'outer;
            }
        }
    }
}
```

continue и break могут помечаться метками (**labels**)

Итераторы

```rust
trait Iterator {
    type Item;
    fn next(&mut self) -> Option<Self::Item>;
    // остальные методы
}
```

Главный метод итератора — next():

- возвращает Some(item), если элемент есть
- возвращает None, если элементы закончились

Пример

```rust
struct Counter {
    count: u32
}

impl Iterator for Counter {

    type Item = u32;

    fn next(&mut self) -> Option<Self::Item> {
        if self.count < 10 {
            self.count += 1;
            Some(self.count)
        } else {
            None
        }
    }
}

fn main() {
    let mut counter = Counter { count: 0 };
    for number in &mut counter {
        println!("Counter: {}", number);
    }
}
```

Трейт IntoIterator

Этот трейт позволяет использовать собственные типы данных в цикле for.

Он определяет метод into_iter(), который создаёт итератор из объекта.

```rust
struct MyCollection {
    data: Vec<i32>,
}

impl MyCollection {
    fn new() -> Self {
        MyCollection { data: Vec::new() }
    }

    fn add(&mut self, value: i32) {
        self.data.push(value);
    }
}

impl IntoIterator for MyCollection {
    type Item = i32;                                // Тип элементов
    type IntoIter = std::vec::IntoIter<i32>;        // Итератор, создающийся из Vec

    fn into_iter(self) -> Self::IntoIter {
        self.data.into_iter()
    }
}

fn main() {
    let mut c = MyCollection::new();
    c.add(2);
    c.add(4);
    c.add(8);

    for item in c {
        println!("Item: {}", item);
    }
}
```

Match

```
let number = 3;
match number {
    1 => println!("Один"),
    2 | 3 => println!("Два или три"),
    4..=10 => println!("От 4 до 10"),
    _ => println!("Другое"),
}
```

- match проверяет все варианты (исчерпывающий анализ)
- Поддерживаются диапазоны и объединение значений
- match также возвращает значение

Блоки

```rust
fn main() {
    let z = 13;
    let x = {
        let y = 10;
        println!("y: {y}");
        z - y
    };
    println!("x: {x}");
}
```

В Rust блок { . . . } — это **выражение**. Если последнее выражение заканчивается ;, результирующим значением и типом является ()

Обработка ошибок в Rust

В Rust нет классического механизма исключений. Вместо этого ошибки выражаются типами.

- Ошибки компиляции
- `Result<T, E>`, `Option<T>` — то, что в других языках выбрасывается как исключение
- `panic!` — для критических ошибок, из-за которых программа завершается

Option<T>— отсутствие значения без null

```rust
fn divide(a: i32, b: i32) -> Option<i32> {
    if b == 0 {
        None
    } else {
        Some(a / b)
    }
}

fn main() {
    match divide(10, 2) {
        Some(result) => println!("Результат: {}", result),
        None => println!("Деление на ноль!"),
    }
}
```

Result<T, E> — ошибки, которые нужно обработать

Result<T, E> = Ok(T) или Err(E)

```rust
use std::fs::File;

fn main() {
    let file = File::open("test.txt");
    match file {
        Ok(f) => println!("Файл открыт: {:?}", f),
        Err(e) => println!("Ошибка: {}", e),
    }
}
```

```rust
let f = File::open("test.txt")?;
```

panic! — ошибки, от которых нельзя продолжать работу

```rust
fn main() {
    assert!(2 + 2 == 4); // всё хорошо
    assert!(2 + 2 == 5); // паника
    panic!("Что-то пошло не так!");
}
```

Список литературы

- Rust Reference (официальная спецификация)
  - https://doc.rust-lang.org/reference/
- Rust by Example
  - https://doc.rust-lang.org/rust-by-example/