链滴

Rust 流程控制

作者：lingyundu
原文链接：https://ld246.com/article/1608367244511
来源网站：链滴

条件选择

if 条件选择是一个表达式（可以用来赋值），并且所有分支都必须返回相同的类型。

判断条件不必用小括号括起来，条件后跟的代码块必须用大括号括起来。

示例一：if 表达式

```
fn main() {
    let number = 3;

    if number != 0 {
        println!("number was something other than zero");
    }
}
```

示例二：if-else 表达式

```
fn main() {
    let number = 3;

    if number < 5 {
        println!("condition was true");
    } else {
        println!("condition was false");
    }
}
```

示例三：if-else-if 表达式

```
fn main() {
    let n = 5;

    if n < 0 {
        println!("{} is negative", n);
    } else if n > 0 {
        println!("{} is positive", n);
    } else {
        println!("{} is zero", n);
    }
}
```

循环语句

Rust 提供了三种循环：loop、while 和 for。

loop 循环

无限循环，一般配合break表达式使用。

示例：

```
fn main() {
```

```
    // 这是一个无限循环
    loop {
        println!("again!");
    }

    let mut counter = 0;

    let result = loop {
        counter += 1;
        // 设置循环终止条件，并返回一个结果
        if counter == 10 {
            break counter * 2;
        }
    };

    println!("The result is {}", result);
}
```

while 循环

示例:

```
fn main() {
    let mut number = 3;

    while number != 0 {
        println!("{}!", number);、
        number = number - 1;
    }

    println!("LIFTOFF!!!");
}
```

for 循环

示例:

```
fn main() {
    // `n` 的值为： 1, 2, ..., 100（`1..101` 等价于 `1..=100`）
    for n in 1..101 {
        if n % 15 == 0 {
            println!("fizzbuzz");
        } else if n % 3 == 0 {
            println!("fizz");
        } else if n % 5 == 0 {
            println!("buzz");
        } else {
            println!("{}", n);
        }
    }

    let a = [10, 20, 30, 40, 50];
    // 使用 for 循环遍历集合中的元素
```

```
    for element in a.iter() {
        println!("the value is: {}", element);
    }
}
```

模式匹配

Rust 通过 match 关键字提供了模式匹配功能，类似于 C 中的 switch，但功能更强大。

match 表达式由 match 关键字、用于匹配的值和一个或多个分支构成，每个分支包含一个**模式**（pattern）以及一个待执行的表达式。

```
match VALUE {
    PATTERN1 => EXPRESSION1,
    PATTERN2 => EXPRESSION2,
    PATTERN3 => EXPRESSION3,
    ...
    PATTERNn => EXPRESSIONn,
}
```

执行过程：依次将 PATTERN1，PATTERN2... 和 VALUE 进行比对，一旦匹配成功就执行该分支中 => 后面的表达式，并且不再继续匹配后面的分支。其中，PATTERNn 可以是字面量、元组、枚举、结体、通配符、函数等。

模式匹配的要求：

- 所有分支必须穷举所有的可能。
- 每个分支必须是一个表达式，且 **通常**这些表达式的返回值类型必须相同。

示例：

```
enum Direction {
    East,
    West,
    North,
    South,
}
fn main() {
    let d_west = Direction::West;
    let d_str = match d_west {
        Direction::East => "East",
        Direction::North | Direction::South => {
            panic!("South or North");
        }, // 该分支一定会触发 panic，返回值类型可以与其他分支不同
        _ => "West",
    };
    println!("{}", d_str);
}
```

解构

当待匹配的模式是复合类型时，可以提取复合类型值中的部分数据。

示例一：解构元组

```
fn main() {
    let triple = (0, -2, 3);

    println!("Tell me about {:?}", triple);

    match triple {
        // 解构元组中第2和第3个元素
        (0, y, z) => println!("First is `0`, `y` is {:?}, and `z` is {:?}", y, z),
        // 使用 `..` 忽略其他元素，仅匹配第1个元素
        (1, ..)  => println!("First is `1` and the rest doesn't matter"),
        // `_` 匹配所有值，通常将它放在最后，用来匹配剩余的所有可能值
        _      => println!("It doesn't matter what they are"),
    }
}
```

示例二：解构枚举

```
#[allow(dead_code)]
enum Color {
    Red,
    Blue,
    Green,
    RGB(u32, u32, u32),
    HSV(u32, u32, u32),
    HSL(u32, u32, u32),
    CMY(u32, u32, u32),
    CMYK(u32, u32, u32, u32),
}

fn main() {
    let color = Color::RGB(122, 17, 40);

    println!("What color is it?");

    match color {
        Color::Red   => println!("The color is Red!"),
        Color::Blue  => println!("The color is Blue!"),
        Color::Green => println!("The color is Green!"),
        Color::RGB(r, g, b) =>
            println!("Red: {}, green: {}, and blue: {}!", r, g, b),
        Color::HSV(h, s, v) =>
            println!("Hue: {}, saturation: {}, value: {}!", h, s, v),
        Color::HSL(h, s, l) =>
            println!("Hue: {}, saturation: {}, lightness: {}!", h, s, l),
        Color::CMY(c, m, y) =>
            println!("Cyan: {}, magenta: {}, yellow: {}!", c, m, y),
        Color::CMYK(c, m, y, k) =>
            println!("Cyan: {}, magenta: {}, yellow: {}, key (black): {}!", c, m, y, k),
    }
}
```

示例三：解构结构体

```rust
fn main() {
    struct Foo {
        x: (u32, u32),
        y: u32,
    }

    let foo = Foo { x: (1, 2), y: 3 };

    match foo {
        Foo { x: (1, b), y } => println!("First of x is 1, b = {},  y = {} ", b, y),

        // 可以对字段（field）重命名
        Foo { y: 2, x: i } => println!("y is 2, i = {:?}", i),
        // Foo { y: 2, x } => println!("y is 2, i = {:?}", x),

        // 可以使用 `..` 忽略一些字段（field）
        Foo { y, .. } => println!("y = {}, we don't care about x", y),
    }
}
```

示例四：匹配范围

```rust
fn main() {
    let x = 1;
    match x {
        1 ..=10 => println!("一到十"),
        _ => println!("其它"),
    }

    let c = 'w';
    match c {
        'a' ..='z' => println!("小写字母"),
        'A' ..='Z' => println!("大写字母"),
        _ => println!("其他字符"),
    }
}
```

示例五：多重匹配

```rust
fn main() {
    let x = 1;
    // 匹配多个模式时，使用 `|` 分隔
    match x {
        1 | 2 => println!("一或二"),
        _ => println!("其他"),
    }
}
```

后置条件

一个 if 表达式可以被放在 match 的模式之后，被称为 match guard。

示例：

```
fn main() {
    let x = 4;
    let y = false;
    match x {
        4 | 5 if y => println!("yes"),
        _ => println!("no"),
    }
}
```

绑定

可以使用 @ 符号将模式中匹配的值绑定一个名称，详细说明见Identifier patterns。

示例：

```
fn age() -> u32 {
    15
}

fn main() {
    println!("Tell me what type of person you are");

    match age() {
        0             => println!("I haven't celebrated my first birthday yet"),
        n @ 1  ..= 12 => println!("I'm a child of age {:?}", n),
        n @ 13 ..= 19 => println!("I'm a teen of age {:?}", n),
        // Nothing bound. Return the result.
        n             => println!("I'm an old person of age {:?}", n),
    }
}
```

if let

在某些情况下，可以使用 if let 对模式匹配进行简化。

示例：

```
fn main() {
    let number = Some(7);
    let letter: Option<i32> = None;

    match number {
        Some(i) => {
            println!("Matched {:?}!", i);
        }
        _ => {} // 此分支什么都不做，但是不能省略，因为模式匹配要列举所有可能
    };

    // 可以将上面的模式匹配该为下面这种形式
    if let Some(i) = number {
        println!("Matched {:?}!", i);
    }
```

```
    // 如果需要对未匹配的情况进行处理，还可以添加 `else` 分支
    if let Some(i) = letter {
        println!("Matched {:?}!", i);
    } else {
        println!("Didn't match a number. Let's go with a letter!");
    }
}
```

while let

与 if let 类似，使用 while let 也可以对模式匹配进行简化，只关注要匹配的值，其它的值不用做显式理。

示例：

```
fn main() {
    let mut optional = Some(0);

    loop {
        match optional {
            Some(i) => {
                if i > 9 {
                    println!("Greater than 9, quit!");
                    optional = None;
                } else {
                    println!("`i` is `{:?}`. Try again.", i);
                    optional = Some(i + 1);
                }
            }
            _ => {
                break;
            }
        }
    }

    // 上面loop中的模式匹配可以改为下面这种形式
    while let Some(i) = optional {
        if i > 9 {
            println!("Greater than 9, quit!");
            optional = None; // 这是循环结束条件
        } else {
            println!("`i` is `{:?}`. Try again.", i);
            optional = Some(i + 1);
        }
    }
}
```

相关资料

The Rust Programming Language

The Rust Reference