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

    for i in 1..5 {
        println!("{}", i);
    }
}
```

4、闭包

闭包是匿名的函数，它们可以捕获其创建环境的变量。在下面的示例代码中，闭包被用来过滤偶数。

```
fn main() {
    let numbers = vec![1, 2, 3, 4, 5];
    let even_numbers = numbers.iter().filter(|&x| x % 2 == 0).collect::<Vec<_>>();
    println!("{:?}", even_numbers);
}
```

5、推导式

通过#[derive(...)]属性，Rust可以自动为结构体或枚举生成标准的方法实现，比如：Debug、Eq、PartialEq等。在下面的示例代码中，我们为Person结构体类型使用了#[derive(Debug)]属性，因此编译器为我们自动生成了Debug Trait的实现。在main函数中，我们使用{:?}格式化占位符来打印person变量的调试信息。

```
#[derive(Debug)]
struct Person {
    name: String,
    age: u8,
}
```

```
fn main() {
    let person = Person {
        name: ”Mike”.to_string(),
        age: 24,
    };

    // 输 ’outer: for i in 0..5 {
        ’inner: for j in 0..5 {
            if i == 2 && j == 2 {
                // 跳出外层循环
                break ’outer;
            }

            println!(”({}, {})”, i, j);
        }
    }
}
```

10、Deref转换

Deref转换允许通过*操作符自动解引用实现了Deref Trait的类型，从而访问其内部的值。这是一种隐式的类型转换，使得代码更加简洁。

```
use std::ops::Deref;

struct CustomBox<T> {
    value: T,
}

impl<T> Deref for CustomBox<T> {
    type Target = T;
    fn deref(&self) -> &Self::Target {
        &self.value
    }
}

fn main() {
    let my_box = CustomBox { value: 66 };
    // 通过Deref转换，可以直接解引用MyBox实例，从而访问其内部的value字段
    println!(”{}”, *my_box);
```

}

```

原文链接: https://juejin.cn/post/7379055493270503443
```