Системне Програмування

З використанням мови програмування **Rust.**
Fundamentals. Arrays

Олексій Рихальський

12.09.2024

Масиви

Масиви є однією з базових структур даних
в будь-якій мові програмування, включаючи Rust.
Вони дозволяють зберігати фіксовану кількість елементів
одного типу, що робить їх дуже корисними
при роботі з множинами даних,
які не змінюють свій розмір під час виконання програми.

Масиви

Масив в програмуванні — це структура даних,
яка дозволяє зберігати множину елементів одного типу
під одним іменем.

Кожен елемент у масиві має свій індекс,
який використовується для доступу до нього.
Масиви зручні для роботи з фіксованими наборами даних,
де потрібно легко отримати доступ до елементів за їх індексом.

Основні характеристики масивів

Розмір: Кількість елементів у масиві визначається при його створенні і не змінюється.

Тип елементів: Усі елементи масиву мають однаковий тип (наприклад, числа, рядки або інші типи даних).
Індексація: Доступ до елементів здійснюється за індексом, зазвичай починаючи з нуля (хоча в деяких мовах програмування індексація може починатися з одиниці).

Декларація масиву

```
let xs: [i32; 5] = [10, 20, 30, 40, 50];

let xs : [i32; 5] =    [10, 20, 30, 40, 50];

let xs : [bool; 2] = [true, false];

let xs : [?; 0] = []; // will not compile
```

Доступ до елементів масиву

```
let xs :[i32; 5] = [10, 20, 30, 40, 50];

let k :i32 = xs[0];
dbg!(k); // 10

let k :i32 = xs[5]; // will not compile
```

Розподіл масиву

```rust
let xs : [i32; 7] = [10, 13, 17, 25, 37, 51, 60];
let (l : &[i32], r : &[i32]) = xs.split_at( mid: 3);

println!("{:?}", l); // [10, 13, 17]
println!("{:?}", r); // [25, 37, 51, 60]
```

Доступ до елементів масиву

```
let xs : [i32; 5] = [10, 20, 30, 40, 50];
let x : Option<&i32> = xs.first();
let y : Option<&i32> = xs.last();
println!("{:?}", x);
println!("{:?}", y);
```

```
let xs: [u32;0] = [];
let x : Option<&u32> = xs.first();
let y : Option<&u32> = xs.last();
println!("{:?}", x);
println!("{:?}", y);
```

Довжина та доступ до елементів масиву

```rust
let xs : [i32; 5] = [10, 20, 30, 40, 50];
println!("{:?}", xs.len());

let z : Option<&[i32; 3]> = xs.last_chunk::<3>();
println!("{:?}", z);

let z : Option<&[i32; 6]> = xs.last_chunk::<6>();
println!("{:?}", z);
```

Доступ до елементів масиву

```rust
let xs : [i32; 5] = [10, 20, 30, 40, 50];
println!("{:?}", xs.len()); // 5

let z : Option<&[i32; 3]> = xs.last_chunk::<3>();
println!("{:?}", z); // Some([30, 40, 50])

let z : Option<&[i32; 6]> = xs.last_chunk::<6>();
println!("{:?}", z); // None
```

Модифікація елементів масиву

```
let xs : [i32; 5] = [10, 20, 30, 40, 50];

xs[0] = 5; // will not compile

let mut xs : [i32; 5] = [10, 20, 30, 40, 50];

xs[0] = 5;
println!("{:?}", xs);
```

```
[5, 20, 30, 40, 50]
```

Деякі функції для роботи з масивами

```
let qs : [&str; 4] = ["a", "b", "c", "d"];
let z : String = qs.join( sep: "_");
println!("{:?}", z); // "a_b_c_d"
```

```
let c : bool = qs.contains( x: &"a");
println!("{:?}", c); // true
let c : bool = qs.contains( x: &"z");
println!("{:?}", c); // false
```

Ітерація по елементах масиву

```
let xs : [i32; 7] = [10, 13, 17, 25, 37, 51, 60];

for x : i32 in xs {
    print!("{} ", x);
}
// 10 13 17 25 37 51 60
```

```
let xs : [i32; 7] = [10, 13, 17, 25, 37, 51, 60];

let it : Iter<i32> = xs.iter();
it.for_each(|x : &i32| print!("{} ", x));
// 10 13 17 25 37 51 60
```

Ітерація по елементах масиву з індексами

```rust
let xs : [i32; 7] = [10, 13, 17, 25, 37, 51, 60];

for (idx : usize, x : &i32) in xs.iter().enumerate() {
    print!("[{}]->{}, ", idx, x);
}
// [0]->10, [1]->13, [2]->17, [3]->25, [4]->37, [5]->51, [6]->60
```

Швидкість

Хоча масиви можуть бути менш гнучкими порівняно з векторами (що змінюють розмір), вони мають перевагу у швидкості, тому що їх розмір завжди відомий на етапі компіляції.
Це дозволяє компілятору оптимізувати роботу з масивами.

Висновок

Масиви в Rust — це ефективний спосіб зберігання набору елементів одного типу, якщо розмір відомий на етапі компіляції. Вони забезпечують швидкий доступ до елементів і гарантують безпеку через перевірку меж. Проте, якщо потрібна гнучкість у роботі з динамічною кількістю елементів, варто використовувати вектори, або інші колекції.
Масиви є важливою частиною екосистеми Rust, забезпечуючи як високу продуктивність, так і безпеку в управлінні пам’яттю.

Код з лекцій,
презентації Keynote,
PDF-файли
знаходяться на GitHub:

https://github.com/djnzx/rust-course