Concorrência sem medo - Rust

Thilo Koch

MAC 5742: Introdução à Computação Paralela e Distribuída
Professor Alfredo Goldman Vel Lejbman
IME USP

slides: http://www.ime.usp.br/~tiko/apresentacao-rust.pdf

12 de junho de 2015

Introdução

- Projeto da Mozilla,
- versão 1.0 em maio 2015,
- inicialmente para programação paralela e distribuída,
- agora mais como concorrente de C/C++,
- combina controle de baixo nível sobre desempenho e conveniências de alto nível,
- abstrações com custos zero (tempo de compilação),
- garantias de segurança de memória (*unique pointers*) - forçadas pelo compilador!,
- foco na programação paralela.

Hello World

Instalação:

```
$ curl -sf -L https://static.rust-lang.org/rustup.sh | sh
```

- Gerar esqueleto

```
$ cargo new hello_world --bin
$ cd hello_world
$ tree .
.
 - Cargo.toml
 - src
     - main.rs
```

- Cargo.toml

```
[package]
name = "hello_world"
version = "0.1.0"
authors = ["tiko <tiko@ime.usp.br>"]
```

- main.rs

```
// This is the main function
fn main() {
    println!("Hello World!");
}
```

- Compilar com rustc ("complainer")

```
$ rustc hello.rs
```

- Executar

```
$ cargo run
     Running `target/debug/hello_world`
Hello, world!
```

Unique pointers

Mudabilidade faz parte da *signature*

```
let a = 1;       // immutable per default
let a mut = 1;   // mutable
```

Ownership

- uma alocação de memória pertence ao escopo
- quando a execução sai do escopo a memória é desalocada
  - C

    ```
    {
        int *x = malloc(sizeof(int));
        *x = 5;
        free(x);
    }
    ```

  - Rust

    ```
    {
        let x = Box::new(5);
    }
    ```

Rust não tem destrutores!

Borrowing

```
fn main() {
    let x = Box::new(5);
    add_one(x);
    println!("{}", x);
}

fn add_one(mut num: Box<i32>)
{
    *num += 1;
}
```

→ Resulta em erro:

```
error: use of moved value: ‘x‘
   println!("{}", x);
```

```
fn main() {
    let mut x = 5;
    add_one(&mut x);
    println!("{}", x);
}

fn add_one(num: &mut i32) {
    *num += 1;
}
```

Computa porque é apenas um empréstimo.

Referências

É possível pedir emprestado:

- várias referências para variáveis somente para leitura

```
fn add(x: &i32, y: &i32) -> i32 {
    *x + *y
}

fn main() {
    let x = Box::new(5);
    println!("{}", add(&*x, &*x));
    println!("{}", add(&*x, &*x));
}
```

- apenas um referências para variáveis mudáveis para escrita (mas não ao mesmo tempo e não em combinação com referências de leitura)

```
fn increment(x: &mut i32) {
    *x += 1;
}

fn main() {
    let mut x = Box::new(5);
    increment(&mut x);
    increment(&mut x);
    println!("{}", x);
}
```

- existem 7 tipos de ponteiros / referências (inclusive *threadsafe*).

Concorrência

- Canais com semântica de mandar mensagem como carta
- `send` transfere a *ownership*

```
let mut vec = Vec::new();
// do some computation
send(&chan, vec);
print_vec(&vec);

->
Error: use of moved value ‘vec‘
```

- existem mutex e locks (data e não código) dentro da biblioteca `std`
- outras ferramentas estão em desenvolvimento (bibliotecas)

Açúcar sintático

- baseado em expressões

```
let y = if x == 5 { 10 } else { 15 };
```

- comentários em markdown para a documentação (`rustdoc`)
- tuplas (lista ordenada com tamanho fixo)

```
let x: (i32, &str) = (1, "hello");
fn next_two(x: i32) -> (i32, i32) { (x + 1, x + 2) }
```

- *match*

```
match x {
    1 | 2   => println!("one or two"),
    3       => println!("three"),
    4 .. 9  => println!("[4,9]")
    _       => println!("anything"),
}
```

- iteradores (*iterators*) para strings

```
let a = [0, 1, 2, 3, 4];
for e in a.iter() {
    println!("{}", e); // Prints 1, 2, 3
}
```

- Macros (por exemplo `println!`)

Aplicações

- Browser engine (Servo) - projeto da Mozilla
- Skylight - analisador de desempenho para aplicações em Rails
- Módulos para o kernel do Linux
- Muitas aplicações e bibliotecas ainda em andamento (p2p, web framework, games, ...)

Fim

- **The Rust Programming Language** http://doc.rust-lang.org/stable/book

Muito obrigado!

Contato: tiko@ime.usp.br