Rust

Alunos: Artur Nascimento do Carmo

Igor Lara de Lima Caldas

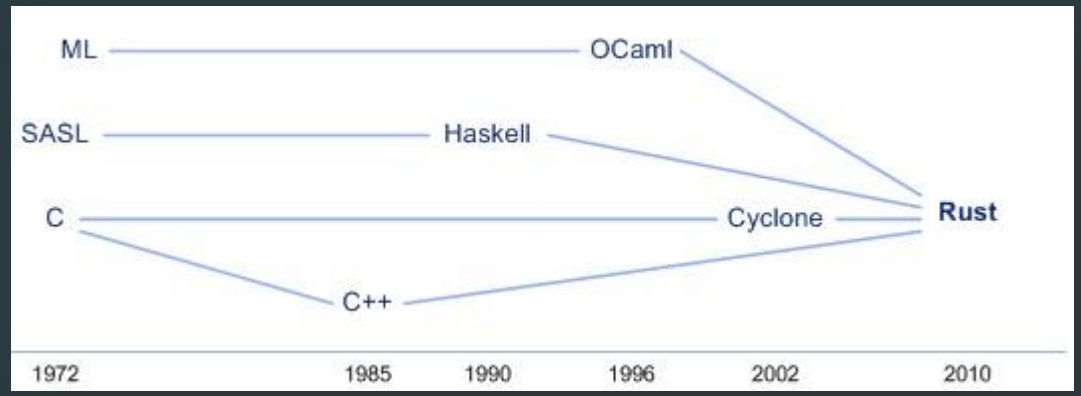
Disciplina: Estrutura de Linguagens

Professor: Francisco Figueiredo Goytacaz Sant’Anna

Origem

A linguagem cresceu em 2006, vinda de um projeto pessoal de um funcionário da Mozilla , Graydon Hoare. Mozilla começou a patrocinar o projeto em 2009 e o anunciou em 2010. Projetada para ser segura , concorrente e prática.

Influências: Alef C#, C++, Cyclone, Erlang, Haskell, Limbo, Newsqueak, OCaml, Scheme, Standard ML, Swift.



Características

- Segurança sem coletor de lixo (linguagens como Java, GO)
- Multiparadigma
- Pattern matching
- Multithreaded
- Funções de alta ordem (closures)
- Polimorfismo, combinando interfaces parecidas com javae classes parecidas com haskell
- Erro tem dois tipos (recuperável e não recuperável)
- Tipagem forte e estática

Exemplos de Código

Em Rust, podemos definir um tipo de dados como:

- Signed: que guarda valor negativo ou positivo
- Unsigned: guarda apenas valores positivos

```
fn main() {
    let result :i32  = 10;
    let age:u32 = 20;
    let sum:i32 = 5-15;
    let mark:isize = 10;
    let count:usize = 30;

}
```

```
let foo = 5; // immutable
let mut bar = 5; // mutable
```

Trait

Rust

```rust
trait CalculoArea {
    fn area(&self) -> f64;
}
struct Circulo {
    raio: f64,
}
impl CalculoArea for Circulo {
    fn area(&self) -> f64 {
        std::f64::consts::PI * (self.raio * self.raio)
    }}
struct Quadrado {
    lado: f64,}
impl CalculoArea for Quadrado {
    fn area(&self) -> f64 { self.lado * self.lado }}
fn imprime<T: CalculoArea>(forma: T) {
    println!("A area é: {}", forma.area());}
fn main() {
    let c = Circulo {
        raio: 7f64,
    };
    let s = Quadrado {
        lado: 6f64,
    };

    imprime( forma: c);
    imprime( forma: s);
}
```

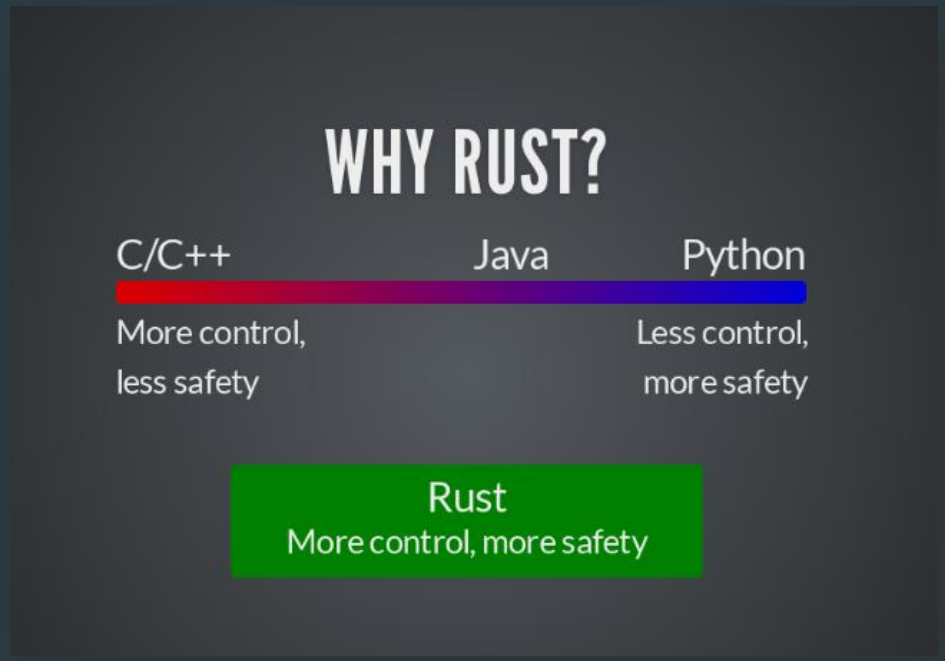
Java

```java
interface Forma {
    double pi = 3.14;
    double calculaArea(double x, double y);
}
class Circulo implements Forma{
    @Override
    public double calculaArea(double x, double y) {
        return (pi*(x*x));
    }
}
class Retangulo implements Forma{
    @Override
    public double calculaArea(double x, double y){
        return (x*y);
    }
}
class Calculo{
    public static void main(String[] args) {
        Circulo a = new Circulo();
        Retangulo b = new Retangulo();
        System.out.println("Area do circulo: "+ a.calculaArea( x: 7, y: 7));
        System.out.println("Area do retangulo: "+b.calculaArea( x: 7, y: 7));
    }
}
```

Segurança de memória

Rust possui dois pilares importantes em sua segurança que influenciam em sua expressividade, o *Ownership* e o *Borrowing*.

Os dois são especificidades de Rust que permitem a maior segurança no uso de dados e um melhor controle dos mesmos.



Ownership

Para que não exista a necessidade de um coletor de lixo, Rust utiliza o *Ownership*. A memória é gerenciada com um conjunto de regras que o compilador verifica em tempo de compilação. Após a transferência de recursos o proprietário anterior não tem mais controle sobre os dados passados. Com isso, é evitada a criação de ponteiros pendentes.

Borrowing

O sistema de *Borrowing* existe para que seja possível utilizar dados sem tomar posse sobre eles, passando-os por referência é possível que se tenha uma ligação, entretanto, não seu controle total tendo em vista que um objeto Emprestado, para que exista, necessita da existência do objeto referenciado, caso contrário, teremos um erro.

Bibliografia

- https://doc.rust-lang.org/book/
- https://en.wikipedia.org/wiki/Rust_(programming_language)
- https://www.tutorialspoint.com/rust/index.htm
- https://www.ime.usp.br/~gold/cursos/2015/MAC5742/reports/rust.pdf