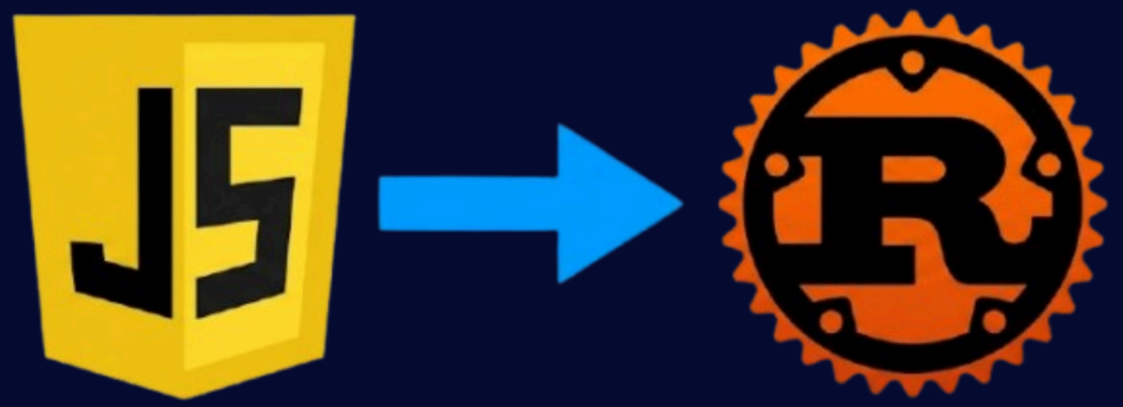
Compiladores e paradigmas de programação

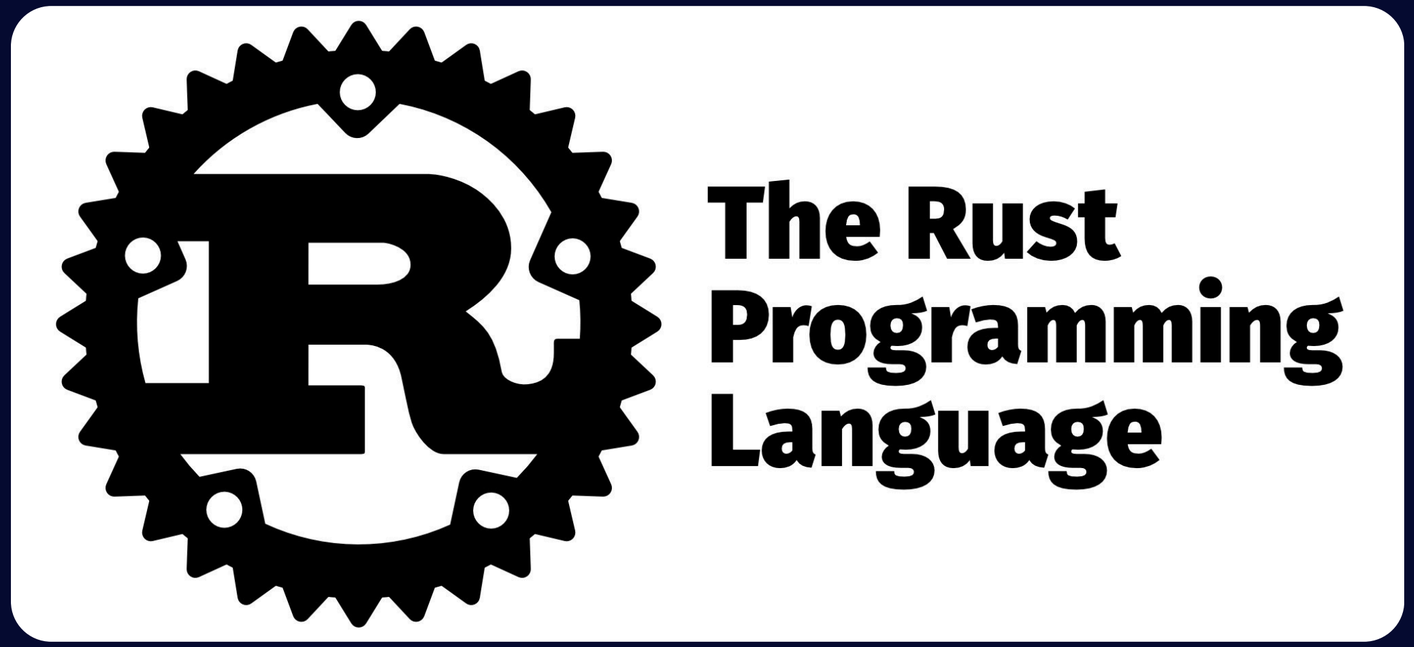
TRANSPILADOR
JAVASCRIPT -> RUST

Erica Kathlen, Igor Dias, João Marcelo, João Guilherme, Rafael Alexander

INTRODUÇÃO

- **O que é:** Transpilador Source-to-Source (JavaScript ES6 para Rust).
- **A Proposta:** Unir a facilidade de escrita do JS com a performance do Rust.
- **O Problema Resolvido:**
  - Eliminação da dependência do Node.js ou Navegador.
  - Criação de **Binários Nativos** e autocontidos (Single Binary).
  - Portabilidade total (roda em qualquer lugar).





RUST

Linguagem de sistemas moderna focada em performance e confiabilidade. Gera binários nativos otimizados e opera sem a necessidade de uma Máquina Virtual, sendo ideal para criar softwares seguros e eficientes.

Tipagem Estática e Forte

Tipos definidos na compilação; proíbe conversões implícitas de dados.

Sem Garbage Collector

Gestão de memória determinística, sem pausas durante a execução.

Ownership

Regra de "dono único": a memória é liberada automaticamente ao fim do escopo.

Borrow Checker

Analisa o código e impede conflitos de acesso à memória antes de compilar.

DIFERENÇAS CONCEITUAIS

JavaScript e Rust são linguagens bastante diferentes: uma prioriza a produtividade através da permissividade, a outra prioriza a performance através do rigor.

01

Flexibilidade vs Rigidez

JS é permissivo e dinâmico, enquanto o Rust é estrito e exige definições estáticas para garantir segurança.

02

Gerenciamento de Memória

JS depende de Garbage Collector (automático), enquanto o Rust usa Ownership (determinístico e sem pausas).

03

Coerção de Tipos

JS realiza conversões implícitas (Tipagem Fraca), enquanto o Rust proíbe operações entre tipos distintos (Tipagem Forte).

DESAFIOS DA TRANSPILAÇÃO

DESAFIO 1

Tipagem Dinâmica

O Rust precisa saber o tipo exato de cada variável na compilação. Como o JavaScript permite que uma variável mude de Número para Texto a qualquer momento, torna-se impossível fazer uma tradução direta para os tipos primitivos do Rust.

DESAFIO 2

Mutabilidade Compartilhada

O JavaScript permite que várias partes do código modifiquem o mesmo objeto simultaneamente. O Borrow Checker do Rust proíbe estritamente isso para evitar conflitos de memória, bloqueando a compilação de códigos que tentam imitar esse padrão do JS.

DESAFIO 3

Operadores Flexíveis

O JavaScript aceita somar tipos diferentes (ex: 10 + "20"), fazendo conversões automáticas. O Rust é rígido e impede operações matemáticas entre tipos distintos, exigindo que essa lógica de "adivinhação" do JS seja recriada manualmente.

JSVALUE

```rust
pub enum JsValue {
    Undefined,
    Null,
    Number(f64),
    String(String),
    Boolean(bool),
    Array(Vec<JsVar>),
    Object(HashMap<String, JsVar>),
    Function(JsFunc),
}
```

USO DE RC E REFCELL

```rust
pub type JsVar = Rc<RefCell<JsValue>>;
```

01

RC

É um ponteiro inteligente que mantém a contagem de “Donos” de um dado.

02

RefCell

Uma estrutura que permite alterar um dado mesmo quando se tem apenas uma referência imutável a ele

FUNÇÕES PERSONALIZADAS

```rust
pub fn add(a: &JsVar, b: &JsVar) -> JsVar {
    let val_a = a.borrow();
    let val_b = b.borrow();
    match (&*val_a, &*val_b) {
        (JsValue::Number(n1), JsValue::Number(n2)) => new_num(n1 + n2),
        (JsValue::String(s1), JsValue::String(s2)) => new_str(&format!("{}{}", s1, s2)),
        (JsValue::String(s1), JsValue::Number(n2)) => new_str(&format!("{}{}", s1, n2)),
        (JsValue::Number(n1), JsValue::String(s2)) => new_str(&format!("{}{}", n1, s2)),
        _ => new_undefined(),
    }
}
```

PROCESSO DE TRANSPILAÇÃO

01

Análise (Parsing)

Conversão do código fonte bruto em uma Árvore de Sintaxe Abstrata (AST) via Babel.

02

Travessia (Visitor)

Varredura recursiva (top-down) da árvore para identificar as estruturas lógicas

03

Tradução (Code Gen)

Mapeamento semântico de cada nó JS para sua função equivalente no Runtime Rust

04

Injeção (Linking)

Acoplamento da biblioteca runtime.rs no código transpilado

05

Compilação (Rustc)

Otimização final e geração do binário nativo executável

EXEMPLOS PRÁTICOS

OBRIGADO PELA ATENÇÃO!

Transpilador disponível no nosso repositório!

https://github.com/JMarcelooo/Compiladores