Quem é o professor ?

Professor: Danilo Aparecido

Profissão: Programador / Empresário

Redes sociais procurar por:

"Danilo Aparecido - Torne-se um programador":

- Linkedin
- Facebook
- Youtube
- Tiktok
- Instagram

O que é Rust?

Origens e Motivação

O desenvolvimento de Rust começou por volta de 2006 por Graydon Hoare enquanto trabalhava na Mozilla.

A motivação para criar Rust surgiu da necessidade de uma linguagem que combinasse eficiência, paralelismo seguro e controle fino sobre recursos de memória, algo que as linguagens existentes não ofereciam de forma satisfatória.

O que é Rust?

Desenvolvimento e Crescimento

Em 2009, a Mozilla patrocinou o projeto, e Rust passou a ser um esforço colaborativo com mais desenvolvedores contribuindo.

Rust foi influenciado por várias linguagens de programação como C++, Haskell, OCaml e Erlang, buscando aprender com os pontos fortes e fracos de cada uma.

O que é Rust?

Primeiras Versões

O desenvolvimento inicial se concentrou em experimentar ideias e conceitos, o que levou a várias revisões na linguagem.

A primeira versão pré-alfa foi lançada em janeiro de 2012.

O que é Rust?

Lançamento da Versão Estável

A primeira versão estável, Rust 1.0, foi lançada em 15 de maio de 2015.

Este lançamento marcou um compromisso com a estabilidade da linguagem, garantindo que os futuros upgrades não quebrassem o código existente.

O que é Rust?

Versão atual 2024

A versão 1.75 foi lançada com diversas melhorias e novos recursos. Esta versão continua a enfatizar a segurança da memória e a eficiência na concorrência, características distintivas do Rust. Entre as novidades desta versão, estão:

- Esta versão aprimorou a usabilidade de elementos desabilitados através dos atributos de compilação condicional. Isso significa que o código que depende de certas condições para ser compilado ou executado tornou-se mais fácil de gerenciar, proporcionando uma experiência de codificação mais eficiente e transparente para os desenvolvedores.
- Rust 1.75 substituiu as restrições rígidas na avaliação de expressões constantes por avisos para cálculos de longa duração. Essa mudança torna o processo de compilação menos restritivo, ao mesmo tempo que mantém os desenvolvedores informados sobre potenciais problemas de desempenho em suas expressões constantes.

O que é Rust?

Versão atual 2024 - versão 1.75

- Novas verificações foram adicionadas ao Clippy, uma ferramenta de linting para Rust, e algumas dessas verificações foram movidas para o próprio compilador Rustc. Isso melhora a detecção de padrões de código problemáticos e ajuda a manter o código Rust limpo e eficiente.
- Foram feitas adições significativas à API do Rust, com a estabilização de novos métodos e implementações de características. Essas adições expandem as funcionalidades disponíveis para os desenvolvedores e aumentam a estabilidade e a maturidade da linguagem.

https://releases.rs/

O que é Rust?

Reconhecimento e Adoção

Rust ganhou rapidamente reconhecimento por sua abordagem inovadora à segurança de memória e concorrência.

A comunidade de desenvolvedores cresceu, atraída pela promessa de Rust de oferecer "concorrência sem medo".

O que é Rust?

Impacto e Aplicações Atuais

Rust tem sido adotado em uma variedade de domínios, desde sistemas operacionais até desenvolvimento web.

Empresas como Dropbox, Cloudflare, e até a própria Mozilla usam Rust para melhorar a segurança e o desempenho de seus produtos.

Por que Utilizar Rust ?

Segurança de Memória

A segurança de memória no Rust é uma das suas características mais notáveis, distanciando-o significativamente de linguagens como C++, que exigem uma gestão manual mais intensiva da memória e são mais suscetíveis a erros relacionados à memória, como referências nulas e estouros de buffer. Uma das principais diferenças que contribuem para essa segurança aprimorada é a ausência de um coletor de lixo (garbage collector) no Rust.

Em linguagens com coletor de lixo, como Java ou C#, o sistema automaticamente recupera a memória que não está mais em uso, o que pode ser conveniente, mas às vezes resulta em uma sobrecarga de desempenho devido à pausas para a coleta de lixo. Rust, por outro lado, usa um sistema de propriedade e empréstimo, que garante a segurança de memória em tempo de compilação. Isso significa que o Rust pode realizar a gestão de memória de forma eficiente e segura, sem a necessidade de um coletor de lixo.

Por que Utilizar Rust ?

Segurança de Memória

Este sistema de propriedade assegura que cada pedaço de dados na memória tem um proprietário claro, e apenas um. Quando o proprietário sai do escopo, os dados são automaticamente descartados. Além disso, o sistema de empréstimo do Rust garante que as referências aos dados sejam gerenciadas de forma que não possam ocorrer condições de corrida ou outros bugs relacionados à concorrência.

A combinação dessas características permite que Rust ofereça segurança de memória sem comprometer o desempenho. Isso é particularmente importante em sistemas de baixo nível, onde a gestão eficiente da memória é crucial, e em aplicações onde a previsibilidade e o desempenho constante são essenciais. Assim, Rust oferece o melhor dos dois mundos: segurança de memória robusta sem o custo adicional de desempenho associado aos coletores de lixo tradicionais

Por que Utilizar Rust ?

Concorrência Segura

Rust facilita a programação concorrente sem riscos de condições de corrida, algo desafiador em outras linguagens como C++.

Por que Utilizar Rust ?

Desempenho

Rust é comparável ao C++ em termos de desempenho. Ambas as linguagens permitem controle detalhado da memória e têm capacidades de execução eficiente. Rust se destaca pela ausência de um coletor de lixo, o que contribui para um desempenho consistente e previsível.

Por que Utilizar Rust ?

Casos de Uso e Exemplos no Mundo Real

Rust é versátil, sendo usado em desenvolvimento de sistemas operacionais, aplicativos web, e jogos. Ele suporta paradigmas de programação procedural, paralela, funcional e OOP.

Por que Utilizar Rust ?

Empresas e Projetos

Grandes empresas e projetos têm adotado Rust devido à sua segurança e desempenho. Isso inclui o desenvolvimento de sistemas operacionais, servidores web, motores de navegadores e jogos.

Por que Utilizar Rust ?

Comunidade e Ecossistema

A comunidade Rust está em constante crescimento, com uma base ativa de desenvolvedores que contribuem para o desenvolvimento e suporte da linguagem.

Por que Utilizar Rust ?

Ferramentas e Bibliotecas

Rust possui um ecossistema rico em ferramentas e bibliotecas, embora ainda não tão extenso quanto o de C++. O gerenciador de pacotes Cargo, por exemplo, simplifica a gestão de dependências e configuração de projetos.

Por que Utilizar Rust ?

Curva de Aprendizado

Rust tem uma curva de aprendizado inicialmente mais íngreme devido ao seu sistema de propriedade e verificação em tempo de compilação. No entanto, isso leva a um código mais seguro e robusto.

Há uma variedade de recursos disponíveis para aprender Rust, incluindo documentação oficial, tutoriais, cursos online e comunidades ativas como a Rust BR.

Por que Utilizar Rust ?

Documentação Oficial

https://rust-br.github.io/rust-book-pt-br/title-page.html

Conclusão

Em conclusão, Rust se destaca como uma linguagem de programação poderosa e eficiente, oferecendo uma combinação única de segurança, desempenho e facilidade de uso.

É uma escolha excelente para projetos onde a segurança e a eficiência são cruciais, e para desenvolvedores que desejam aprimorar suas habilidades em uma linguagem moderna e em constante crescimento.

High level language
Low level language

Rust é classificado como uma linguagem de programação de "baixo nível" por várias razões, embora também ofereça muitas características de linguagens de "alto nível"

High level language
Low level language

Controle direto sobre o hardware: Rust proporciona um controle muito granular sobre os recursos do sistema, como memória e processamento. Isso permite que os programadores otimizem o desempenho para aplicações específicas, uma característica comum em linguagens de baixo nível.

High level language
Low level language

Gerenciamento manual de memória: Embora Rust introduza o sistema de propriedade para gerenciar a memória de forma segura e evitar erros comuns em C e C++ (como dangling pointers e memory leaks), os programadores ainda têm a capacidade de gerenciar a memória explicitamente. Isso é algo tipicamente associado a linguagens de baixo nível, que oferecem controle direto sobre a alocação e desalocação de memória.

High level language
Low level language

Sem coleta de lixo: Rust não usa um coletor de lixo (garbage collector - GC) como linguagens de alto nível, como Java ou C#. Isso significa que não há uma pausa indeterminada na execução do programa para coleta de lixo, permitindo um controle mais previsível sobre o desempenho e o uso de recursos.

High level language
Low level language

Abstração zero: Rust segue a filosofia de "zero-cost abstractions", onde as abstrações introduzidas pela linguagem não devem ter um custo de execução maior do que se você tivesse escrito seu código em uma linguagem de baixo nível. Isso significa que você pode escrever código de alto nível sem sacrificar o desempenho, uma característica desejável em programação de sistemas e aplicações de performance crítica.

Gerenciamento de memória

Data Races

Um data race ocorre em programas concorrentes quando dois ou mais threads acessam a mesma variável de memória simultaneamente, e pelo menos uma das acessos é uma escrita. Se esses acessos não são sincronizados (ou seja, não há mecanismos para garantir a ordem ou exclusividade dos acessos), podem ocorrer comportamentos inesperados e bugs difíceis de reproduzir. Data races são um problema comum em programação multithread, onde a ordem de execução das threads não é determinística e pode levar a resultados inconsistentes ou corrupção de dados.

Gerenciamento de memória

Memory Leak

Um memory leak (vazamento de memória) ocorre quando uma aplicação aloca memória mas não a libera adequadamente após o uso. Com o tempo, a memória que não é mais necessária permanece alocada, reduzindo a quantidade de memória disponível para outras partes do programa ou outros programas. Isso pode levar à degradação do desempenho e, eventualmente, à falha do programa por falta de recursos de memória. Memory leaks são especialmente problemáticos em programas de longa execução, como servidores ou aplicações de desktop.

Gerenciamento de memória

Stack Overflow

Um stack overflow é um erro que ocorre quando o espaço de memória da pilha (stack) de um programa é excedido. A pilha é uma região de memória que armazena variáveis locais, parâmetros de funções e endereços de retorno. Um overflow pode acontecer por motivos como recursão profunda sem condição de término adequada, ou a alocação de grandes quantidades de dados na pilha. Quando o limite de memória da pilha é ultrapassado, pode resultar em um crash do programa.

Gerenciamento de memória

Outros conceitos importantes:

Deadlock: Situação em programação concorrente onde dois ou mais processos ou threads estão esperando uns pelos outros para liberar recursos, resultando em um impasse onde nenhum dos processos pode avançar.

Heap Overflow: Similar ao stack overflow, mas ocorre na heap, uma área de memória usada para alocação dinâmica. Acontece quando um programa tenta alocar mais memória na heap do que o máximo disponível, podendo resultar em falhas do programa ou comportamento imprevisível.

Use-After-Free: Erro que ocorre quando um programa tenta acessar ou modificar memória que já foi liberada. Isso pode levar a comportamentos indefinidos, incluindo corrupção de dados, falhas e vulnerabilidades de segurança.

Gerenciamento de memória no Rust

Deadlocks, Heap Overflow, e Use-After-Free

Deadlocks não são diretamente evitados pelo Rust, pois eles são mais sobre a lógica de bloqueio e espera entre threads. No entanto, Rust oferece tipos abstratos de alto nível para gerenciamento de concorrência que podem ajudar a reduzir a possibilidade de deadlocks.

Heap Overflow é um tipo de problema de gerenciamento de memória que Rust ajuda a mitigar através de verificações de limites em tempo de execução para acessos a vetores e outras estruturas de dados, mas a alocação excessiva de memória ainda é possível se não for devidamente gerenciada pelo desenvolvedor.

Use-After-Free é largamente prevenido pelo sistema de propriedade e verificador de empréstimos de Rust. Rust garante que os dados só podem ser acessados enquanto é válido, e o compilador impede que os programadores criem referências a dados que foram desalocados.

Programação Funcional

Entender os Conceitos Fundamentais

- **Imutabilidade**: Dados imutáveis não podem ser modificados após sua criação. Isso ajuda a evitar efeitos colaterais e torna o código mais previsível.

- **Funções Puras:** São funções que, para os mesmos inputs, sempre retornarão o mesmo output e não têm efeitos colaterais.

- **Expressões sobre Instruções:** Em programação funcional, você expressa o "o quê" ao invés do "como", focando em expressões que calculam valores em vez de instruções que alteram o estado do programa.

Programação Funcional

Utilizar Recursos e Ferramentas da Linguagem

- **Funções de Alta Ordem:** São funções que podem receber outras funções como argumentos ou retorná-las como resultado. Elas são fundamentais para criar abstrações poderosas e reutilizáveis.

- **Closures:** Funções que podem capturar e utilizar variáveis do escopo onde foram criadas.

- **Estruturas de Dados Imutáveis:** Preferir o uso de estruturas de dados que não podem ser alteradas após sua criação.

- **Pattern Matching:** Uma forma de verificar e destrinchar valores de acordo com um padrão. É particularmente útil para trabalhar com tipos algébricos de dados, como enums em Rust.

Programação Funcional

Praticar Técnicas Comuns da Programação Funcional

- **Recursão**: Muitas vezes, em programação funcional, loops são expressos através de recursão. É importante entender como implementar recursão de forma eficiente, especialmente a recursão de cauda (tail recursion), que é otimizada em muitas linguagens para evitar estouro de pilha.

- **Map, Filter, e Reduce**: São operações fundamentais para trabalhar com coleções de dados de maneira declarativa.

- **Composição de Funções:** A capacidade de combinar duas ou mais funções para criar uma nova função.

Concorrência e Paralelismo

Rust é uma linguagem de programação que oferece suporte robusto tanto à concorrência quanto ao paralelismo, projetada para garantir segurança de memória e thread sem o uso de um coletor de lixo. Isso é alcançado através de várias abstrações e características da linguagem que permitem aos desenvolvedores escrever código seguro e eficiente em ambientes concorrentes e paralelos.

Concorrência e Paralelismo

Concorrência

A concorrência em Rust é principalmente tratada através do modelo de propriedade e do sistema de tipos. O Rust usa conceitos como *ownership* (propriedade), *borrowing* (empréstimo), e *lifetimes* (tempo de vida) para prevenir condições de corrida em tempo de compilação. Além disso, a biblioteca padrão do Rust oferece vários primitivos para trabalhar com concorrência de forma segura, como:

- Threads: Rust permite criar threads usando a função `std::thread::spawn`, permitindo a execução de código em paralelo. O modelo de propriedade do Rust garante que o acesso aos dados seja gerenciado de forma segura entre threads.
- Canais: Para comunicação entre threads, Rust oferece canais (usando `std::sync::mpsc`), permitindo o envio de mensagens de forma segura entre threads.
- Mutexes e Arc: Rust fornece Mutexes (Mutual Exclusion) para garantir que apenas uma thread possa acessar um recurso por vez. Além disso, o tipo `Arc` (Atomic Reference Counting) é usado para compartilhar a propriedade de dados entre threads de forma segura.

Concorrência e Paralelismo

Paralelismo

O paralelismo em Rust é facilitado por bibliotecas externas como Rayon, que permitem a execução de operações em paralelo de forma mais eficiente. Rayon adiciona métodos paralelos para iteradores, permitindo que as operações de dados sejam automaticamente divididas em múltiplas threads. Isso torna o paralelismo mais acessível e seguro, aproveitando o sistema de tipos do Rust para evitar erros comuns em programação paralela.

Concorrência e Paralelismo

Async

Tokio é uma biblioteca assíncrona de Rust focada em entrada/saída (I/O) e sistemas de rede. Ela permite a execução de múltiplas tarefas de I/O simultaneamente, sem bloquear, através de um modelo de programação assíncrona. Tokio é construído em torno do modelo de "futuros" e "tarefas", permitindo que você escreva código que pode esperar por operações de I/O para completar sem parar a thread em que está sendo executado.

Concorrência e Paralelismo

Segurança

A segurança é uma das principais vantagens do Rust em concorrência e paralelismo. O compilador Rust impõe regras estritas em tempo de compilação que eliminam muitos tipos de erros comuns em programas concorrentes e paralelos, como condições de corrida, deadlocks e acessos a memória não sincronizados.

Concorrência e Paralelismo

Conclusão

Rust oferece uma abordagem moderna e segura para escrever programas concorrentes e paralelos. Suas abstrações de alto nível e verificação em tempo de compilação ajudam a prevenir muitos dos problemas comuns associados à programação concorrente e paralela, tornando-a uma escolha atraente para projetos que exigem alta performance e segurança.

Projetos Web

- HTML e CSS
- JavaScript
- Conceitos de Rede (HTTP/HTTPS, DNS, REST, RESTFull, GraphQL, ElasticSearch, Banco de Dados Etc.)
- Controle de Versão
- Segurança Web (XSS, CSRF, SQL injection)
- Desenvolvimento e Operações (DevOps) AWS ou outro provider
- Frameworks Web e Ferramentas
- Fundamentos de WebAssembly (opcional) - Iteração do Rust com JS

Projetos Web

Os principais frameworks para desenvolvimento de aplicações web em Rust são projetados para maximizar a performance e segurança, tirando proveito das características únicas da linguagem. Aqui estão alguns dos frameworks mais notáveis

Projetos Web

Actix Web

Descrição: Actix Web é um poderoso, pragmático, e extremamente rápido framework web para Rust. É conhecido por sua alta performance e modelo de ator para lidar com concorrência.

Características: Suporte a WebSocket, servidor HTTP/2, sistema de plugins, e uma arquitetura extensível. Actix Web é uma ótima escolha para projetos que exigem alta performance e segurança.

Projetos Web

Rocket

Descrição: Rocket é um framework web para Rust que se destaca por sua simplicidade e expressividade. Requer a versão nightly do Rust, mas compensa isso com uma API muito amigável e um guia de aprendizado excelente.

Características: Suporte a declarações de tipo para validação de request, sistema de templates integrado, e um forte foco na experiência do desenvolvedor. Rocket é ideal para quem está começando com Rust e desenvolvimento web.

Projetos Web

Tide

Descrição: Tide é um framework web minimalista e extensível em Rust, construído sobre o runtime async do Rust, o `async-std`. É projetado para ser simples de usar, enquanto ainda é poderoso e flexível.

- **Características**: Middleware fácil de usar, roteamento simples, e suporte a WebSockets. Tide é uma boa escolha para desenvolvedores que procuram uma abordagem mais simples e direta para o desenvolvimento web em Rust.

Projetos Web

Warp

Descrição: Warp é um framework web para Rust que se baseia na composição de filtros para criar APIs web. É construído sobre o Hyper, uma biblioteca de baixo nível HTTP em Rust, oferecendo uma abordagem única para o roteamento e manipulação de requisições.

Características: Composição de filtros para uma maior reusabilidade de código, excelente performance, e suporte a WebSockets. Warp é ideal para projetos que necessitam de uma abordagem modular e flexível.

Projetos Web

Axum

Descrição: Axum é um framework web escrito em Rust que se concentra em ergonomia e modularidade. Construído sobre o Tokio, Axum tira proveito das características async do Rust para oferecer uma experiência de desenvolvimento produtiva.

Características: Focado em segurança de tipos, fácil de compor, e incentiva a criação de aplicações web concorrentes e seguras. Axum é uma escolha excelente para projetos que buscam tirar proveito do modelo async do Rust.

Projetos Web

Cada um desses frameworks tem seu conjunto único de características e benefícios, tornando-os adequados para diferentes tipos de projetos web em Rust. A escolha entre eles dependerá das necessidades específicas do seu projeto, sua familiaridade com a linguagem Rust, e a complexidade da aplicação que você está desenvolvendo.