Advanced Features of Rust Programming

Paweł Nowak

December 4, 2023

Zero Cost Abstraction

- Derived from C++ concept.
- Achieved through efficient iterators and other abstractions.
- Enables high performance without sacrificing safety or expressiveness.

Ownership Model

- Fundamental to Rust's memory safety guarantees.
- Ownership rules:
  - Each value in Rust has a single owner.
  - Only one owner at a time.
  - When the owner goes out of scope, the value is dropped.
- Prevents common errors like dangling pointers and memory leaks.

Algebraic Data Types

- Inspired by functional languages like Haskell.
- Combines different types in a single coherent unit.
- Commonly used with ‘enum‘ for defining custom data types.

Polymorphism

- Achieved in Rust through traits.
- Traits allow for shared behavior across different types.
- Similar to interfaces in other languages but more powerful.

Async Programming

- Integral part of Rust for efficient IO operations.
- Uses ‘async‘ and ‘await‘ syntax for easy-to-read asynchronous code.
- Enables concurrent execution without the complexity of threads.

Macros

- Macros in Rust are powerful and hygienic.
- Allow for metaprogramming and code generation.
- Hygienic nature ensures no unintended scope interference.

Cargo - The Rust Package Manager

- Cargo manages Rust projects, dependencies, and builds.
- Inspired by Node.js's npm.
- Simplifies project management and compilation process.

Conclusion

- Rust provides advanced features for safe, concurrent, and practical programming.
- Its unique approach to memory safety and management sets it apart.
- Continuously evolving with a strong focus on developer needs.