НОВОЕ ПОКОЛЕНИЕ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ – RUST

NEW GENERATION OF PROGRAMMING LANGUAGES - RUST

УДК 004.432.3

Кайшев Денис Анатольевич, магистрант, Московский государственный технологический университет «СТАНКИН», г. Москва

Kayshev Denis Anatolevich, Graduate Student, Moscow State Technological University STANKIN, Moscow, e-mail: topenkoff@gmail.com

Аннотация

В статье рассматривается Rust как один из языков нового поколения. Были рассмотрены основные концепции языка, а также ряд особенностей, отличающих от других существующих языков.

Annotation

This article looks at Rust as one of the next generation languages. The basic concepts of the language were considered, as well as a number of functions that distinguish them from other languages, the advantages and disadvantages were considered.

Ключевые слова: Мультипарадигменность, функциональное программирование, обобщенное программирование, императивное программирование

Keywords: Multi-paradigm, functional, generic, imperative

Rust – это мультипарадигменный язык общего назначения, с объектной системой в основе которой лежит понятие типаж. Данный язык берет свое начало в 2006, разработкой которого занимался сотрудник компании Mozila Грэйдон

Хор. В последствии компания Mozila начала спонсировать проект. Впоследствии с 2010 года развитием языка занимается open-source сообщество. Для Rust характерна сильная статическая типизация, поддержка параметрического полиморфизма, а также большая часть функционального программирования. Данный язык является тьюринг полным, а это означает, что на нем можно написать любую вычислимую функцию. Так же данный язык является компилируемым. Компилятор написан так же на языке Rust, бэкендом которого является LLVM для генерации машинного кода.

Владение является центральной особенностью языка Rust. Хотя эту особенность легко объяснить, она весьма сильно повлияла на остальную часть языка.

Все программы во время выполнения используют память компьютера и используют разные подходы для управления своей памятью. В одних языках программирования для этой цели используют систему сборки мусора (garbage collection, GC), постоянно следящую за памятью программы, которая больше не используется программой. В других языках программист должен сам явно запрашивать и освобождать память. Rust же использует третий подход: память управляется с помощью системы владения с набором правил, которые компилятор проверяет только во время компиляции программы. Ни одно из правил владения не замедляет выполнение программы:

- каждое значение имеет переменную, которая называется владельцем значения,
- у значения может быть только один владелец в один момент времени,
- когда владелец покидает область видимости, значение удаляется.

Данные правила позволяют эффективно работать с памятью, так как сам язык декларирует, что вся память, выделенная для переменных, которые используются в ходе выполнения программы, будет очищена после ее завершения. Реализована так называемая Move-семантика: по умолчанию Rust «переносит» указатель на объект в куче новому владельцу при присваивании, делая старую переменную недействительной.

В языке также присутствует такая сущность как типаж. Типаж сообщает компилятору Rust о функциональности, которой обладает определённый тип и которой он может поделиться с другими типами. Можно использовать типажи, чтобы определять общее поведение абстрактным способом. Можно использовать типажи для ограничения обобщённого типа: указать, что обобщённым типом может быть любой тип который реализует определённое поведение.

Поведение типа определяется теми методами, которые мы можем вызвать у данного типа. Различные типы разделяют одинаковое поведение, если мы можем вызвать одни и те же методы у этих типов. Определение типажей - это способ сгруппировать сигнатуры методов вместе для того, чтобы описать общее поведение, необходимое для достижения определённой цели.

Безопасное и эффективное управление многопоточным программированием - ещё одна из основных целей Rust. Системы владения и система типов являются мощным набором инструментов, помогающих управлять безопасностью памяти и проблемами многопоточного параллелизма! Используя владение и проверку типов, многие ошибки многопоточности являются ошибками времени компиляции в Rust, а не ошибками времени выполнения. Поэтому вместо того, чтобы тратить много времени на попытки воспроизвести точные обстоятельства, при которых возникает ошибка многопоточности во время выполнения, некорректный код будет отклонён с ошибкой.

В Rust есть разные умные указатели, определённые в стандартной библиотеке, которые обеспечивают функциональность выходящую за рамки ссылок. Умный указатель — идиома косвенного обращения к памяти, которая широко используется при программировании на языках высокого уровня. В Rust есть разные умные указатели, определённые в стандартной библиотеке, которые обеспечивают функциональность выходящую за рамки ссылок. Одним из примеров, который мы рассмотрим в этой главе, является тип умного указателя reference counting (подсчёт ссылок). Этот указатель позволяет иметь

несколько владельцев данных с помощью отслеживания количества владельцев и очистки данных, когда владельцев не осталось.

Одним из примеров, который мы рассмотрим в этой главе, является тип умного указателя reference counting (подсчёт ссылок). Этот указатель позволяет иметь несколько владельцев данных с помощью отслеживания количества владельцев и очистки данных, когда владельцев не осталось.

В Rust, который использует концепцию владения и заимствования, дополнительная разница между ссылками и умными указателями заключается в том, что ссылки являются указателями, которые только заимствуют данные; против того, что во многих случаях умные указатели владеют данными на которые они указывают.

Дизайн языка Rust получил вдохновение из многих существующих языков и техник, и одним из существенных влияний является функциональное программирование. Программирование в функциональном стиле часто включает использование функций в качестве значений, передаваемых в аргументах, возвращаемых из других функций, назначаемых переменным для последующего выполнения и так далее.

Если говорить про асинхронность, то на данный момент в стандартной библиотеке отсутствует поддержка такого функционала. Но существуют две концепции стороннего развития языка - tokio-rs и async_std. Это библиотеки, которые добавляют сущности, необходимые для асинхронного программирования – event loop, futures.

Таким образом Rust является многофункциональным языком, который позволяет реализовывать программы как высокого уровня – веб-приложения, так и низкоуровневые приложения – драйвера, системные утилиты и т.д.

Литература

1. Rust docs. URL: https://doc.rust-lang.org/book/ (31.05.2021)
2. The Rustonomicon URL: https://doc.rust-lang.org/nomicon/
3. Learn Rust URL: https://www.rust-lang.org/learn
4. Rust URL: https://github.com/rust-lang/rust

5. Rust vs C: https://evrone.com/rust-vs-c

Literature

1. Rust docs. URL: https://doc.rust-lang.org/book/ (31.05.2021)
2. The Rustonomicon URL: https://doc.rust-lang.org/nomicon/ (31.05.2021)
3. Learn Rust URL: https://www.rust-lang.org/learn (31.05.2021)
4. Rust URL: https://github.com/rust-lang/rust (31.05.2021)
5. Rust vs C: https://evrone.com/rust-vs-c (31.05.2021)